Typing Linear Constraints

SALVATORE RUGGIERI
Dipartimento di Informatica, Università di Pisa, Italy
FRED MESNARD
IREMIA, LIM, Université de la Réunion, France

We present a type system for linear constraints over the reals intended for reasoning about the input-output directionality of variables. Types model the properties of definiteness, range width or approximation, lower and upper bounds of variables in a linear constraint. Several proof procedures are presented for inferring the type of a variable and for checking validity of type assertions. We rely on theory and tools for linear programming problems, linear algebra, parameterized polyhedra and negative constraints. An application of the type system is proposed in the context of the static analysis of constraint logic programs. Type assertions are at the basis of the extension of well-moding from pure logic programming. The proof procedures (both for type assertion validity and for well-moding) are implemented and their computational complexity is discussed. We report experimental results demonstrating the efficiency in practice of the proposed approach.

Categories and Subject Descriptors: I.2.2 [**Artificial Intelligence**]: Automatic programming; I.2.3 [**Artificial Intelligence**]: Deduction and theorem proving

General Terms: Languages, Verification

Additional Key Words and Phrases: Linear constraints, polyhedra, constraint logic programming, well-moding, definiteness

1. INTRODUCTION

(Linear) constraint solving is naturally undirected, in the precise sense that there is no explicit set of input/output variables. As an example, let c be the linear constraint `F = 9/5 * C + 32` stating the Fahrenheit-Celsius conversion rule. A constraint solver provided with c and a Celsius value `C = 30` returns `F = 86` as the solved constraint. Conversely, a constraint solver provided with c and a Fahrenheit value `F = 77` returns `C = 25` as the solved constraint. This extends to intervals, lower bounds and upper bounds as well. Assume that the Celsius temperature is known with some approximation (e.g., due to measurement error of a sensor) `C =` `30` $\pm$ `1`, or, by adopting a different syntax, that `29 =< C, C =< 31`. A constraint solver provided with c and a Celsius interval `29 =< C, C =< 31` returns as the solved constraint `84.2 =< F, F =< 87.8`, or `F = 86` $\pm$ `1.8`.

Corresponding author's address: S. Ruggieri, Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy. e-mail: `ruggieri@di.unipi.it`. A preliminary version of the results presented in this paper appeared in [Ruggieri and Mesnard 2008].

Programming with constraints allows then for declaratively modelling a problem by generating the set of constraints among the variables at hand, and then letting a constraint solver find a solved form. Constraint generation can be as simple as feeding a constraint solver with a predetermined system of linear equalities and inequalities, as in the example above. Or it can assume more complex forms, requiring iteration, recursion or non-determinism. Constraint logic programming languages, for instance, adopt recursion, non-determinism and intertwined constraint generation & solving.

The objective of this paper is twofold. As the first main contribution, we propose a type system that allows for reasoning on the input-output directionality of variables occurring in a linear constraint. The types considered include definiteness of a variable, namely it assumes a unique value; lower bound; upper bound; and maximum range width. These types allow for verifying that the solved form returned by a constraint solver satisfies the intended directionality, e.g., that a certain variable is definite, or that it has a variability range within a maximum threshold, or that it is upper/lower bounded. Recalling the Fahrenheit-Celsius conversion example, we will be able to conclude that if `C` is definite, then in the solved form of c also `F` is definite; or that if `C` is known with an approximation of $\pm$ 1, then in the solved form of c we have that `F` is known with an approximation of $\pm$ 1.8. Formally, type assertions are introduced in order to derive types implied by a constraint and a set of typed variables. The semantics of a type assertion is provided as a first order formula over the theory of the reals. However, a geometric view based on parameterized polyhedra will be considered in order to reason on type assertions. Validity of type assertions is thoroughly investigated by devising several checking and inference procedures. We proceed incrementally by first considering lower and upper bound types: the inference procedure relies on solving homogeneous linear programming problems. Next, we add definiteness by reasoning on the set of implicit equalities of the underlying linear constraint. Third, we tackle range width by computing the Minkowski's form of a parameterized polyhedron. The procedures are sound and complete with respect to the appropriate subset of types. Moreover, the approach is extended to deal with constraints containing strict inequalities and disequalities. Finally, parameters are added to the languages of types in order to express range widths of parametric size. Recalling again the Fahrenheit-Celsius conversion example, we will be able to show that if `C` is known with an approximation of $\pm s$, then in the solved form of c we have that `F` is known with an approximation of $\pm 1.8s$.

The considered types can be useful for reasoning about variable directionality in programs and systems loosely coupled with a constraint solving library. Even more challenging is the case of programming languages that tightly integrate constraint solving. As another main contribution of this paper, we study how the type system can be adopted for reasoning over constraint logic programs, which represent a sophisticated scheme for programming with constraints. Modern constraint logic programming languages, such as Mercury [Somogyi et al. 1996; Becket et al. 2006], offer the notion of moding [Apt 1997] as program annotations allowing the programmer to specify the input-output behavior of predicate arguments. Modes are at the basis of compiler optimizations, program transformations and termination analyses. As an example, consider the `MORTGAGE` program over CLP($\mathcal{R}$).

```
(m1)  mortgage(P,T,R,B) ←        (m2)  mortgage(P,T,R,B) ←
         T = 0,                           T >= 1,
         B = P.                           NP = P + P * 0.05 - R,
                                          NT = T - 1,
                                          mortgage(NP,NT,R,B).
```

The query ← `mortgage(100, 5, 20, B)` is intended for calculating the balance of a mortgage of 100 units after giving back 20 units per year for a period of 5 years. The answer provides an exact value (i.e., a real number) for the required balance, namely `B = 17.12`. Using the moding terminology, we say that given definite values for principal, time and repayment, in every (non-deterministically computed) answer we obtain a definite value for the balance. However, this is only one mode we can query the program above. The query ← `3 <= T, T <= 5, mortgage(100, T, 20, B)` is intended for calculating the balance at the end of the third, fourth and fifth year. Principal and repayment are now definite, whilst time is (upper and lower) bounded. Again, for each of the three answers: `T=3, B=52.71`; `T=4, B=35.35`; and `T=5, B=17.12` we get a definite value for the balance. Intuitively, this mode is more general than the previous one, since definiteness of time has been replaced by boundedness. Finally, consider the query ← `0 <= B, B <= 10, 15 <= R, R <= 20, mortgage(P, 5, R, B)`. It is intended for calculating the principal that a person could be granted under the condition of repaying from 15 to 20 units per year, and with the requirement that after 5 years the final balance is of at most 10 units. The answer is `P=0.78*B+4.33*R`, which is not definite, but, since `B` and `R` are bounded, it is (upper and lower) bounded. This mode is not comparable to the previous ones, since we now provide a definite value for time and a range for balance and repayment, and we wish to compute a range for the principal of the answer. These examples give only a few hints about the flexibility of the constraint logic programming (CLP) scheme when compared to pure logic programming, where definiteness of variables corresponds to groundness, but upper and lower bounds have no direct counterpart. Type assertions are at the basis of moding programs in constraint logic programming languages with linear constraints over reals and rationals, as in CLP($\mathcal{R}$) [Jaffar et al. 1992; Holzbaur 1995], ECLiPSe, SICStus Prolog, SWI-Prolog, and many others. We conservatively extend the notion of well-moding [Apt 1997] from pure logic programming to CLP($\mathcal{R}$), proving useful properties in support of static analysis, including persistence, call pattern characterization and answer constraint characterization. Moreover, we show how to deal with programs mixing linear constraints and logic terms, and how the notion generalizes in the presence of parametric types.

Besides discussing the computational complexity of the procedures presented in the paper, we have implemented, in standard C++, all the type assertion checking and inference procedures, and the well-moding checking procedures. Experimental results are reported over a large collection of CLP programs to demonstrate the efficiency in practice of the proposed approach. Proof obligations generated by the notion of well-moding turn out to be a representative testbed for experimental evaluation of the type checking and inference procedures. The software developed, called `clpt`, is released as open source. The experimental evaluation of the designed procedures represents the third main contribution of this paper.

Organization of the paper. We start by recalling in Section 1.1 basic notions for linear algebra, linear programming and constraint logic programming. In Section 2 we introduce syntax and semantics of types, type declarations and type assertions. The problems of checking validity of type assertions and of inferring the most general types for variables are formally introduced. In Section 3, we rephrase the problems in a geometric view, which is the basis for designing several checking and inference procedures. Also, extensions to strict inequalities and disequalities, and to parametric types are reported. Finally, the computational complexity of the procedures is discussed. In Section 4, modes and well-moding for CLP($\mathcal{R}$) programs, possibly with logic terms, are introduced, with type assertions as the basic proof mechanism. The implementations of the type inference procedures and of the well-mode checking procedures are presented in Section 5 together with an extensive experimentation over several testbed programs. Related work and conclusions follow in Section 6 and Section 7. For readability reasons, all proofs of the paper are reported in Appendix A.

1.1 Preliminaries

We adhere to standard notations for linear algebra [Schrijver 1986], linear programming [Murty 1983] and (constraint) logic programming [Apt 1997; Jaffar and Maher 1994; Jaffar et al. 1998].

Linear algebra. $\mathcal{R}$ denotes the set of real numbers. Small capital letters ($\mathbf{a}$, $\mathbf{b}$, ...) denote column vectors, while capital letters ($\mathbf{A}$, $\mathbf{B}$, ...) denote matrices. $\mathbf{0}$ and $\mathbf{1}$ are column vectors with all elements equal to 0 and 1 respectively. $\mathbf{a}_i$ denotes the i^{th} element in $\mathbf{a}$, $\mathbf{A}_i$ the i^{th} column in $\mathbf{A}$, and $row(\mathbf{A}, i)$ the row vector consisting of the i^{th} row of $\mathbf{A}$. $\mathbf{a}^T$ (resp., $\mathbf{A}^T$) denotes the transposed vector (resp., matrix) of $\mathbf{a}$ (resp., $\mathbf{A}$). $\mathbf{c}^T\mathbf{x}$ denotes the inner product of the transposed vector $\mathbf{c}^T$ and $\mathbf{x}$. $\Sigma\mathbf{v}$ is the sum of all the elements in $\mathbf{v}$. $\mathbf{Ax} \leq \mathbf{b}$ denotes a system of linear inequalities (or, a linear system) over the variables in $\mathbf{x}$. We assume that the dimensions of vectors and matrices in inner products and linear systems are of the appropriate size. The solution set of points that satisfy a formula/linear system ψ over $\mathcal{R}^n$ is defined as $Sol(\psi) = \{\mathbf{x} \in \mathcal{R}^n \mid \psi(\mathbf{x})\}$. A polyhedron is the solution set of a linear system, namely $Sol(\mathbf{Ax} \leq \mathbf{b})$. Polyhedra are both closed and convex sets. The vectorial sum of two sets of vectors is defined as: $A + B = \{\mathbf{x} + \mathbf{y} \mid \mathbf{x} \in A, \mathbf{y} \in B\}$.

Linear programming. A linear programming problem consists of determining $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\}$, if it exists. The problem is infeasible when $\{\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\} = \emptyset$. If feasible, but $\{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\}$ has no upper bound, the problem is unbounded, and we write $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\} = \infty$. Otherwise, it is bounded. We write $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\} \in \mathcal{R}$ when the problem is feasible and bounded.

Constraint logic programming. The CLP Scheme defines a family of languages, CLP($\mathcal{C}$), that are parametric in the constraint domain $\mathcal{C}$. We are interested here in CLP($\mathcal{R}$), namely the constraint domain over the reals[1]. All results apply to the

[1]We assume that the underlying constraint solver is ideal, namely without rounding errors. For CLP($\mathcal{R}$) this is achieved by adopting exact arithmetic libraries. For CLP($\mathcal{Q}$) this is achieved by adopting rationals and exact integer arithmetic libraries.

constraint domain of rationals as well, namely to CLP($\mathcal{Q}$) programming languages. A primitive linear constraint is an expression $a_1 \cdot x_1 + \ldots a_n \cdot x_n \simeq a_0$, where $\simeq$ is in $\{\leq, =, \geq\}$, $a_1, \ldots, a_n$ are constants in $\mathcal{R}$ and $x_1, \ldots, x_n$ are variables. We will use the inner product form by rewriting it as $\mathbf{c}^T\mathbf{x} \simeq \alpha$. A linear constraint c is a sequence of primitive linear constraints, whose interpretation is their conjunction. We write $\mathcal{R} \models \psi$ to denote that the first order formula ψ is true in the domain of the reals [Shoenfield 1967]. A CLP($\mathcal{R}$) program is a finite set of clauses of the form $A \leftarrow c, B_1, \ldots, B_n$, where A is an atom, c a linear constraint, and $B_1, \ldots, B_n$ ($n \geq 0$) a sequence of atoms. We assume that atoms are in flat form, namely an atom is of the form $p(x_1, \ldots, x_n)$ where p is a predicate of arity n and $x_1, \ldots, x_n$ are (not necessarily distinct) variables. A query $\leftarrow c, B_1, \ldots, B_n$ consists of a linear constraint and a sequence of atoms.

2. BOUND TYPES FOR LINEAR CONSTRAINTS

2.1 Syntax and Semantics

We introduce a static typing for variables in linear constraints. The set of types $\mathcal{BT}$ is defined first.

Definition 2.1 (Types). *A* type *is an element of* $\mathcal{BT} = \{\star, \sqcup, \sqcap, \Box\} \cup \{\Box_r \mid r \in \mathcal{R}, r \geq 0\}$. *We use* ! *as a shorthand for* $\Box_0$. *Moreover, we denote by* $\mathcal{BT}_\Box$ *the subset* $\{\star, \sqcup, \sqcap, \Box\}$*, and by* $\mathcal{BT}_!$ *the subset* $\{\star, \sqcup, \sqcap, \Box, !\}$.

The intuitive meaning of a type is to label variables occurring in a constraint on the basis of the values that they can assume in the set of solutions of the constraint. ! is intended for typing variables that show at most one single value in every solution, a property known as *definiteness*; $\Box_r$ is intended for typing variables that assume a range of values of width at most r or, as an alternative interpretation, that are known with an approximation of $\pm r/2$; $\Box$ is intended for typing variables that assume a range of values of unknown width, yet lower and upper bounds for those values still exist; $\sqcup$ (resp., $\sqcap$) is intended for variables that have a lower bound (resp., an upper bound); and finally, $\star$ is to be used when no upper or lower bound can be stated. Let us introduce syntactic means to assert the type of variables.

Definition 2.2 (Types assertions). *An* atomic type declaration *(atd, for short) is an expression* $x : \tau$*, where* x *is a variable and* $\tau \in \mathcal{BT}$*. We define* $vars(x : \tau) = \{x\}$*, and say that* x *is typed as* τ.

A type declaration *is a sequence of atd's* $d_1, \ldots, d_n$*, with* $n \geq 0$*. We define* $vars(d_1, \ldots, d_n) = \cup_{i=1..n} vars(d_i)$.

A type assertion *is an expression* $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$*, where* $\mathbf{d}_1, \mathbf{d}_2$ *are type declarations and* c *is a linear constraint.*

Type declarations assign a type to variables. Such a typing is used in type assertions as an hypothesis (to the left of $\vdash$) or as a conclusion (to the right of $\rightarrow$). Intuitively, the type assertion $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ states that given the type declaration $\mathbf{d}_1$, the type declaration $\mathbf{d}_2$ holds under the linear constraint c.

Example 2.3. The type assertion $z :! \vdash y - x \leq z, y + x \leq z, -y - 2x \leq 5 - z \rightarrow y : \sqcap, x : \sqcup$ intuitively states that if z has a fixed value then the set of solutions of the

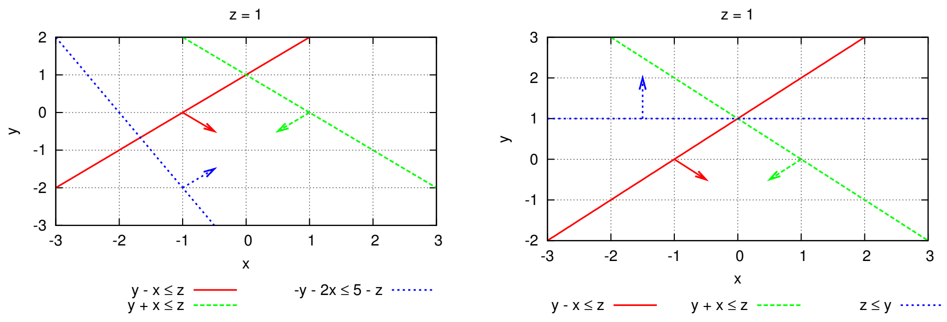

Fig. 1. Solutions of sample linear constraints for a fixed value of the z variable.

involved constraint is either empty or such that y has an upper bound and x has a lower bound. Figure 1 (left) shows graphically the set of solutions for $z = 1$.

The type assertion $z :! \vdash y - x \leq z, y + x \leq z, z \leq y \rightarrow y :!, x :!$ states that if z has a fixed value then either the set of solutions of the involved constraint is empty or both x and y assume a unique value in it. Figure 1 (right) shows graphically the set of solutions for $z = 1$.

For a type declaration $\mathbf{d}$, we write $\mathbf{d}|_{\mathbf{x}}$ to denote the subsequence of $\mathbf{d}$ consisting only of atd's typing variables in $\mathbf{x}$. Analogously, we write $\mathbf{d}|_\tau$ (resp., $\mathbf{d}|_\mathcal{B}$) to denote the subsequence consisting only of atd's typing variables as τ (resp., as any $\tau \in \mathcal{B} \subseteq \mathcal{BT}$). The intuition on the meaning of type assertions is formalized by the next definition. Notice that we are redundant about the type !.

Definition 2.4 (Semantics). *We associate with an atd $d = x : \tau$ a formula $\phi(d)$ over fresh variables $\upsilon(d)$, called parameters, as follows:*

$$\begin{array}{ll}
\phi(x :!) = x = a & \upsilon(x :!) = \{a\} \\
\phi(x : \Box_r) = a \leq x \wedge x \leq a + r & \upsilon(x : \Box_r) = \{a\} \\
\phi(x : \Box) = a \leq x \wedge x \leq b & \upsilon(x : \Box) = \{a, b\} \\
\phi(x : \sqcup) = a \leq x & \upsilon(x : \sqcup) = \{a\} \\
\phi(x : \sqcap) = x \leq b & \upsilon(x : \sqcap) = \{b\} \\
\phi(x : \star) = \texttt{true} & \upsilon(x : \star) = \emptyset.
\end{array}$$

ϕ and υ extend to type declarations as follows:

$$\phi(d_1, \ldots, d_n) = \wedge_{i=1..n} \phi(d_i) \qquad \upsilon(d_1, \ldots, d_n) = \cup_{i=1..n} \upsilon(d_i).$$

A type assertion $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is valid if for $\mathbf{v} = vars(c) \cup vars(\mathbf{d}_1) \cup vars(\mathbf{d}_2)$, the following formula is true in the domain of the reals:

$$\forall \upsilon(\mathbf{d}_1) \exists \upsilon(\mathbf{d}_2) \setminus \upsilon(\mathbf{d}_1) \forall \mathbf{v}. (\phi(\mathbf{d}_1) \wedge c) \rightarrow \phi(\mathbf{d}_2). \tag{1}$$

Notice that, for the $\upsilon()$ function defined so far, we have that $\upsilon(\mathbf{d}_2) \setminus \upsilon(\mathbf{d}_1) = \upsilon(\mathbf{d}_2)$, since we consider fresh parameters. Later on in Section 3.6, the syntax will be extended, and then the general formulation of (1) will be required.

Example 2.5. For the type assertion $z :! \vdash y - x \leq z, y + x \leq z, z \leq y \rightarrow y :!, x :!$, the formula to be shown is:

$$\forall a\ \exists b, c\ \forall x, y, z\ (z = a \wedge y - x \leq z \wedge y + x \leq z \wedge z \leq y) \rightarrow (y = b \wedge x = c).$$

The set of variables is fixed to $\mathbf{v} = \mathit{vars}(c) \cup \mathit{vars}(\mathbf{d}_1) \cup \mathit{vars}(\mathbf{d}_2)$ in order to take into account variables that appear in $\mathbf{d}_1$ or $\mathbf{d}_2$ but not in c. E.g., for the (valid) type assertion $x : \sqcap \vdash \mathtt{true} \rightarrow x : \sqcap$, we have the formula $\forall a\ \exists b\ \forall x\ x \leq a \rightarrow x \leq b$.

A natural ordering over types is induced by the semantics above. For instance, it is readily checked that $\mathbf{d} \vdash c \rightarrow x :!$ implies $\mathbf{d} \vdash c \rightarrow x : \square$ for any $\mathbf{d}$, c and x, or that $\mathbf{d} \vdash c \rightarrow x : \square_2$ implies $\mathbf{d} \vdash c \rightarrow x : \square_3$. Similar implications lead to define an order $\geq_t$ over types, which models type preciseness.

Definition 2.6. *The $\geq_t$ partial order over $\mathcal{BT}$ is defined as the reflexive and transitive closure of the following relation $\rightarrow$:*

$$\begin{tikzcd} & & \sqcap \arrow[dr] & \\ ! \arrow[r] & \square \arrow[ur] \arrow[dr] & & \star \\ & & \sqcup \arrow[ur] & \end{tikzcd}$$

$$! \rightarrow \square_r \rightarrow \square_s \rightarrow \square$$
for every $r < s$.

We write $\tau >_t \mu$ when $\tau \geq_t \mu$ and $\tau \neq \mu$. We define $\mathit{lub}(\emptyset) = \star$ and for $n > 0$:

$$\mathit{lub}(\{\tau_1, \ldots, \tau_n\}) = \mathit{min}\{\tau \mid \tau \geq_t \tau_i, i = 1..n\}.$$

Next, the $\geq_t$ relation is extended to type declarations.

Definition 2.7. *We write $\mathbf{d}_1 \geq_t \mathbf{d}_2$ if for every $x : \tau$ in $\mathbf{d}_2$ there exists $x : \mu$ in $\mathbf{d}_1$ such that $\mu \geq_t \tau$.*

Using the notation $\geq_t$, the intuition behind the ordering can be formalized in a monotonicity lemma.

Lemma 2.8 (Monotonicity). *Assume that $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is valid. If $\mathbf{d}'_1 \geq_t \mathbf{d}_1$, $\mathbf{d}_2 \geq_t \mathbf{d}'_2$ and $\mathcal{R} \models c' \rightarrow c$ for a linear constraint c', then $\mathbf{d}'_1 \vdash c' \rightarrow \mathbf{d}'_2$ is valid.*

Normal forms for type declarations are introduced by assigning to each variable the least upper bound of its types. When the least upper bound is $\star$, the type assignment provides no actual information and then it can be discarded. Normal forms are unique modulo reordering of atd's.

Definition 2.9. *We define $\mathit{nf}(\mathbf{d})$ as any type declaration $\mathbf{d}'$ such that $x : \tau$ is in $\mathbf{d}'$ iff $\tau = \mathit{lub}(\{\mu \mid x : \mu \text{ is in } \mathbf{d}\ \})$ and $\tau \neq \star$.*

Example 2.10. Notice that $x : \square \geq_t x : \sqcap, x : \sqcup$ holds, while $x : \sqcap, x : \sqcup \geq_t x : \square$ does not hold. Actually, $\geq_t$ does not capture semantic implication. We have to move to normal forms to conclude that $\mathit{nf}(x : \sqcap, x : \sqcup) = x : \square \geq_t x : \square$.

Normal forms precisely characterize validity when it only depends on type declarations, i.e. for the constraint `true`.

Lemma 2.11. $\mathbf{d}_1 \vdash \mathtt{true} \rightarrow \mathbf{d}_2$ *is valid iff* $\mathit{nf}(\mathbf{d}_1) \geq_t \mathit{nf}(\mathbf{d}_2)$.

Finally, transitivity of the $\vdash$ relation is readily checked.

Lemma 2.12 (Transitivity). *Assume that $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ and $\mathbf{d}_1, \mathbf{d}_2 \vdash c \rightarrow \mathbf{d}_3$ are valid. Then $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_3$ is valid.*

2.2 Checking and Inferring Type Assertions

Let us concentrate now on the problem of checking the validity of a type assertion. In principle, formulas as in (1) can be checked by real quantifier elimination methods [Dolzmann et al. 1998b; Renegar 1992]. Quantifier elimination traces back to Tarski's decision procedure [Van Den Vries 1988] for first order formula over real polynomials. The core of the procedure in the case of linear polynomials over reals is the Fourier-Motzkin projection method [Schrijver 1986]. Although in the worst case quantifier elimination is doubly exponential [Basu et al. 1996; Davenport and Heintz 1988], approaches efficient-in-practice have been proposed and successfully applied to theorem proving and program verification. We mention partial cylindrical algebraic decomposition as provided in the QEPCAD/QEPCAD-B systems [Collins and Hong 1991; Brown 2003] and available in the Mathematica tool [Strzebonski 2000] and virtual substitution of test terms [Dolzmann et al. 1998a] as provided in the REDLOG system [Dolzmann and Sturm 1997] and specialized for low-degree polynomials.

While quantifier elimination represents a direct solution to the checking problem and it allows for generalizing to the non-linear case, we observe that formulas in (1) represent a quite restricted class. We will be looking then for a specialized and efficient approach to check them. In addition, provided with a type declaration $\mathbf{d}$, a linear constraint c, and a set of variables $\mathbf{v}$, we are interested in the problem of inferring type declarations $\mathbf{d}'$ for variables in $\mathbf{v}$ such that $\mathbf{d} \vdash c \rightarrow \mathbf{d}'$ is valid. More precisely, we are interested in the largest $\mathbf{d}'$ (w.r.t. the $\geq_t$ order), if it exists. The inference problem cannot be directly tackled by real quantifier elimination.

Definition 2.13 (Checking and inference problems). *The* type checking problem *consists of deciding whether a given type assertion $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is valid.*

The type inference problem *consists of computing, given $\mathbf{d}_1$, c and a set of variables $\mathbf{v}$, a type declaration $\mathbf{d}_2$ with $vars(\mathbf{d}_2) \subseteq \mathbf{v}$ such that for every $\mathbf{d}'_2$ with $vars(\mathbf{d}'_2) \subseteq \mathbf{v}$: $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}'_2$ is valid iff $\mathbf{d}_2 \geq_t \mathbf{d}'_2$ holds.*

A procedure is sound for the type inference problem if the type declaration $\mathbf{d}_2$ returned by the procedure is such that $vars(\mathbf{d}_2) \subseteq \mathbf{v}$ and $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is valid[2]. For a subset of types $\mathcal{B} \subseteq \mathcal{BT}$, we say that a procedure is complete for $\mathcal{B}$ if $\mathbf{d}_2$ is such that $\mathbf{d}_2 \geq_t \mathbf{d}'_2$ for every valid $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}'_2$ with $vars(\mathbf{d}'_2) \subseteq \mathbf{v}$ and all types in $\mathbf{d}'_2$ belonging to $\mathcal{B}$. The next result shows that a solution exists for the type inference problem and there is a standard representation obtained as the normal form of any solution. Notice, however, that the proof of the lemma is non-constructive.

Lemma 2.14. *Given $\mathbf{d}_1$, c and a set of variables $\mathbf{v}$, there exists at least one solution $\mathbf{d}_2$ to the type inference problem. Moreover, for any two solutions $\mathbf{d}_2$ and $\mathbf{d}'_2$, we have that $nf(\mathbf{d}_2) = nf(\mathbf{d}'_2)$ (modulo reordering of atd's).*

Assume $\mathbf{d}_1$ be $x :!$, and c be $y = x, z \leq w$ and $\mathbf{v}$ be y, z. A solution to the inference problem is $y :!, z : \star$. By adding less precise typings we can obtain other

[2]As a consequence, $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}'_2$ is valid for every $\mathbf{d}'_2$ such that $\mathbf{d}_2 \geq_t \mathbf{d}'_2$.

```
Input: a type assertion $\mathbf{d}_1 \vdash c \to \mathbf{d}_2$

Step 0. $\mathbf{d} = \chi(\mathbf{d}_1, c, vars(\mathbf{d}_2))$.
Step 1. If $\mathbf{d} \geq_t \mathbf{d}_2$ Then output "Valid"
        Else output "Not valid".
```

Fig. 2. CHECK(χ) procedure.

solutions, such as $y : \Box, y :!, z : \star$. All of them have the same normal form $y :!$.

Example 2.15. Consider the type assertion $x : \Box_2 \vdash x - 1 \leq z \leq x \to z : \tau$. What is the most general type inferrable for τ? Since x ranges in $[a, a+2]$ for some $a \in \mathcal{R}$, by simply varying x we have that z ranges in the interval $[a-1, a+2]$, hence $\tau = \Box_3$ is the most accurate type inferrable for z. In other words, $\tau = \Box_3$ makes the assertion above valid (soundness), and $\Box_3 \geq_t \tau$ holds for every τ such that the type assertion above is valid (completeness).

An inference procedure returning $z : \Box$ is sound, albeit it cannot be complete for the full $\mathcal{BT}$ type system.

Notice that type inference can be tricky, especially when $\Box_r$ types are involved.

Example 2.16. For $x : \Box_2 \vdash -x \leq z \leq x \to z : \tau$, we can only infer $\tau = \Box$. In fact, assume that x ranges in $[a, a+2]$ for $a \geq 0$. Then z ranges in $[-a-2, a+2]$, whose width (i.e., $2(a+2)$) is unbounded since $a \geq 0$.

As another example, let c be $x - w \leq z, z \leq x + w$. From $x : \Box_2, w : \Box_3 \vdash c \to z : \tau, w : \mu$, we can only infer $\tau = \Box, \mu = \Box_3$. However, by adding $w \leq 0$ to c, we can infer $\tau = \Box_2, \mu = !$.

A solution to the inference problem can be easily turned into a solution to the checking problem. Given $\mathbf{d}_1 \vdash c \to \mathbf{d}'_2$ to be checked for validity, we first compute the type declaration $\mathbf{d}_2$ returned by the type inference procedure, and then we check whether $\mathbf{d}_2 \geq_t \mathbf{d}'_2$ holds. This meta-procedure is summarized in Figure 2.

LEMMA 2.17. *Let χ be a type inference procedure that is sound, and complete for $\mathcal{B} \subseteq \mathcal{BT}$.* CHECK($\chi$) *is a decision procedure for the type checking problem w.r.t. type assertions $\mathbf{d}_1 \vdash c \to \mathbf{d}_2$ such that all types in $\mathbf{d}_2$ belong to $\mathcal{B}$.*

3. INFERENCE PROCEDURES FOR TYPE ASSERTIONS

3.1 First Intuitions

Our approach switches from the *logical* view of constraints-as-formulas to a *geometric* view of constraints-as-polyhedra. Consider a linear constraint c and a type declaration $\mathbf{d}$. We observe that c can be equivalently represented as a linear system of inequalities $\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c$ where $\mathbf{v} = vars(c) \cup vars(\mathbf{d})$. The set of solutions of c coincides then with the polyhedron represented by $\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c$, which we call the *geometric* representation of c. Analogously, the linear constraint $\phi(\mathbf{d})$ can be represented as $\mathbf{A}_\mathbf{d}\mathbf{v} \leq \mathbf{B}_\mathbf{d}\mathbf{a}_\mathbf{d}$, where $\mathbf{a}_\mathbf{d}$ is the symbolic vector of parameters in $\upsilon(\mathbf{d})$. The resulting system $\phi(\mathbf{d}) \wedge c$ is a parameterized system of linear inequalities $\mathcal{P}$, where variables in $\upsilon(\mathbf{d})$ play the role of parameters:

$$\begin{pmatrix} \mathbf{A}_c \\ \mathbf{A}_\mathbf{d} \end{pmatrix} \mathbf{v} \leq \begin{pmatrix} \mathbf{b}_c \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{B}_\mathbf{d} \end{pmatrix} \mathbf{a}_\mathbf{d} \qquad (2)$$

The notion of parameterized polyhedron models the solutions of a parameterized linear system.

DEFINITION 3.1 (PARAMETERIZED POLYHEDRON). *A* parameterized polyhedron *is a collection of polyhedra defined by fixing the value for the parameters in a parameterized system of linear inequalities:* $Sol(\mathbf{Ax} \leq \mathbf{b} + \mathbf{Ba}, \mathbf{u}) = \{\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b} + \mathbf{Bu}\}$.

$Sol()$ is now a binary function. In addition to a system of parameterized linear inequalities, an assignment to parameters is required.

Example 3.2. Let $\mathbf{d}$ be z :! and c be $y - x \leq z, y + x \leq z, -y - 2x \leq 5 - z$. We have that $\phi(\mathbf{d})$ is $z = a$, and then the parameterized system for $\phi(\mathbf{d}) \wedge c$ is:

$$\begin{pmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \\ -2 & -1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ 5 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{pmatrix} a$$

Under this interpretation, validity of $\mathbf{d} \vdash c \rightarrow x : \tau$ has an intuitive geometric interpretation. Assume that $x = \mathbf{v}_i$. $\mathbf{d} \vdash c \rightarrow x : \tau$ is valid iff for every $\mathbf{u} \in \mathcal{R}^{|v(\mathbf{d})|}$, the set of solution points $\mathcal{S}_\mathbf{u} = Sol(\mathcal{P}, \mathbf{u})$ is empty or, with $M_\mathbf{u} = max\{\mathbf{v}_i \mid \mathbf{v} \in \mathcal{S}_\mathbf{u}\}$ and $m_\mathbf{u} = min\{\mathbf{v}_i \mid \mathbf{v} \in \mathcal{S}_\mathbf{u}\}$, the following statements hold:

—if $\tau = \Box_r$ then $M_\mathbf{u} \in \mathcal{R}$, $m_\mathbf{u} \in \mathcal{R}$, and $M_\mathbf{u} - m_\mathbf{u} \leq r$, namely x assumes a range of values of width at most r; an alternative formulation is to require that for every $x, y \in \mathcal{S}_\mathbf{u}, abs(x - y) \leq r$;

—as a special case of the previous one, if $\tau = !$ then x assumes a single value;

—if $\tau = \Box$ then $M_\mathbf{u} \in \mathcal{R}$ and $m_\mathbf{u} \in \mathcal{R}$, namely both an upper and a lower bound exist for x;

—if $\tau = \sqcup$ then $m_\mathbf{u} \in \mathcal{R}$, namely a lower bound exists for x;

—if $\tau = \sqcap$ then $M_\mathbf{u} \in \mathcal{R}$, namely an upper bound exists for x;

—if $\tau = \star$ then we have nothing to show.

Unfortunately, this procedure is not effective, since there are infinitely many $\mathcal{S}_\mathbf{u}$ to be checked. In the next two subsections, we will develop approaches for turning the intuitions above into effective procedures.

Example 3.3. Let us consider an example explaining the difference between the $\Box_r$ and the $\Box$ types. The type assertion x :! $\vdash 0 \leq z \leq x \rightarrow z : \Box$ is valid, since for every $a \in \mathcal{R}$, z is lower bounded (by 0) and upper bounded (by $x = a$) in $\mathcal{S}_a = Sol((x = a, 0 \leq z \leq x), (a))$. However, the width of the variability range of z is a. Thus, for any $r \geq 0$ by choosing $a = r + 1$ the type assertion x :! $\vdash 0 \leq z \leq x \rightarrow z : \Box_r$ is invalidated.

3.2 A Linear Programming Approach for $\mathcal{BT}_\Box$

In this section we develop an inference algorithm which does not explicitly take into account parameters. We will be able to reason on type assertions over $\mathcal{BT}_\Box$. First of all, let us consider the case of unsatisfiable constraints.

$$\begin{pmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \\ -2 & -1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

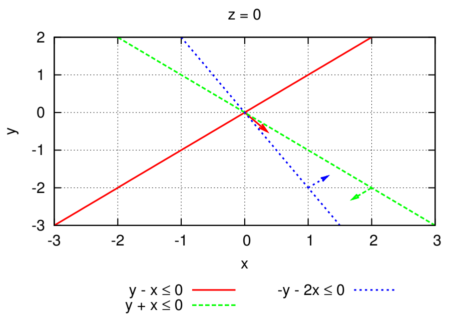

Fig. 3. The homogeneous version of the parameterized linear system in Example 3.2.

Lemma 3.4. *Consider the parameterized linear system $\mathcal{P} = \phi(\mathbf{d}) \wedge c$ in (2). There exists a parameter instance $\mathbf{u}$ such that $Sol(\mathcal{P}, \mathbf{u}) \neq \emptyset$ iff $Sol(\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c) \neq \emptyset$.*

As a consequence, if $Sol(\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c) = \emptyset$ (i.e., c is unsatisfiable) then there is no chance to obtain a non-empty polyhedron by some instantiation of the parameters in $\phi(\mathbf{d})$. In this case, we can infer assertions of the form $\mathbf{d} \vdash c \rightarrow x :!$, for every variable x. Hence, from now on, we will concentrate on satisfiable constraints.

As it will be recalled later on, a non-empty polyhedron $Sol(\mathbf{Ax} \leq \mathbf{b})$ can be decomposed into the vectorial sum of its characteristic cone $Sol(\mathbf{Ax} \leq \mathbf{0})$ with a polytope, a polyhedra bounded along every dimension. Therefore, the existence of an upper/lower bound for a linear function over a polyhedron depends only on its characteristic cone. It is immediate from Definition 3.1 that for every parameter instance $\mathbf{u}$, the polyhedra $Sol(\mathcal{P}, \mathbf{u})$ share the same characteristic cone. As a consequence, proving the existence of an upper bound relies only on the homogeneous version of $\mathcal{P}$, which is not anymore parameterized.

Lemma 3.5. *Consider the parameterized polyhedron $\mathcal{P}$ in (2). Let $\mathcal{H}$ be its homogeneous version: $\mathbf{A}_c\mathbf{v} \leq \mathbf{0}$, $\mathbf{A_d}\mathbf{v} \leq \mathbf{0}$, and assume that $Sol(\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c) \neq \emptyset$.*

We have that $max\{\mathbf{c}^T\mathbf{v} \mid \mathbf{v} \in Sol(\mathcal{H})\} = 0$ iff for every parameter instance $\mathbf{u}$, $Sol(\mathcal{P}, \mathbf{u}) = \emptyset$ or $max\{\mathbf{c}^T\mathbf{v} \mid \mathbf{v} \in Sol(\mathcal{P}, \mathbf{u})\} \in \mathcal{R}$.

When $\mathbf{c}$ is always 0 except for the i^{th} position where it is 1, we have $\mathbf{c}^T\mathbf{v} = \mathbf{v}_i$. Lemma 3.5 solves then the problem of deciding whether $\mathbf{d} \vdash c \rightarrow \mathbf{v}_i : \sqcap$, without having to take into account parameters. By reasoning similarly for types $\sqcup$ and $\square$, we can state an effective procedure, called LPInfer and summarized in Figure 4.

Example 3.6. The homogeneous version of the parameterized linear system in Example 3.2 and its graphical representation are reported in Figure 3. It is readily checked that x has a lower bound and y has an upper bound.

Notice that, in general, the homogeneous version of $\phi(x :!)$, $\phi(x : \square_r)$ and $\phi(x : \square)$ collapse to $x = 0$. Also, the homogeneous version of $\phi(x : \sqcup)$ is $x \geq 0$, and the one of $\phi(x : \sqcap)$ is $x \leq 0$.

Soundness and a relative form of completeness of LPInfer follow. As a consequence of Lemma 2.17, Check(LPInfer) is a decision procedure for $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ when $\mathbf{d}_2$ is defined in the $\mathcal{BT}_\square$ type system.

```
Input: a type declaration $\mathbf{d}_1$, a linear constraint $c$, and a set of variables $\mathbf{v}$

Step 0. Define $\mathbf{v}_c = vars(c)$, $\mathbf{n} = nf(\mathbf{d}_1)$, $\mathbf{d} = \mathbf{n}|_{\mathbf{v}_c}$.
Step 1. Let $\mathbf{A}_c\mathbf{v}_c \leq \mathbf{b}_c$ be the geometric representation of $c$, and $\mathbf{A_d}\mathbf{v}_c \leq \mathbf{B_d}\mathbf{a_d}$ the geometric
        representation of $\phi(\mathbf{d})$.
Step 2. If $Sol(\mathbf{A}_c\mathbf{v}_c \leq \mathbf{b}_c) = \emptyset$ Then For every $x$ in $\mathbf{v}$, output "$x$ :!"
        Else
Step 3.   For every $x$ in $\mathbf{v} \setminus \mathbf{v}_c$:
            (a). output "$x : \tau$" if $x : \tau$ is in $\mathbf{n}$;
            (b). output "$x : \star$" otherwise.
Step 4.   For every $x$ in $\mathbf{v} \cap \mathbf{v}_c$,
          let $M = max\{x \mid \mathbf{A}_c\mathbf{v}_c \leq \mathbf{0}, \mathbf{A_d}\mathbf{v}_c \leq \mathbf{0}\}$, $m = max\{-x \mid \mathbf{A}_c\mathbf{v}_c \leq \mathbf{0}, \mathbf{A_d}\mathbf{v}_c \leq \mathbf{0}\}$:
            (a). output "$x : \Box$" if $M = 0$ and $m = 0$;
            (b). output "$x : \sqcup$" if $M = \infty$ and $m = 0$;
            (c). output "$x : \sqcap$" if $M = 0$ and $m = \infty$;
            (d). output "$x : \star$" if $M = \infty$ and $m = \infty$.
```

Fig. 4. LPInfer procedure.

Theorem 3.7 (LPInfer - soundness and completeness). LPInfer *is sound for the type inference problem, and it is complete for* $\mathcal{BT}_\Box$.

The LPInfer procedure is not tied to any underlying linear programming solver. One straight choice is to adopt the Simplex-based approach [Murty 1983; Schrijver 1986], with at most $2|\mathbf{v} \cap \mathbf{v}_c| + 1$ calls to the Simplex algorithm, namely one call to test satisfiability at *Step 2* and two calls per variable to test upper and lower bounds at *Step 4*. Due to the approach that we will follow later on for dealing with parameters, we present here an instantiation of LPInfer relying on the generating matrix and the vertex matrix of polyhedra. This is an alternative representation of polyhedra, known as the explicit representation or the Minkowski's form [Schrijver 1986, Section 8.9]. Later on in Sect. 3.7, we will discuss the computational complexity of this instantiation compared to a linear programming based one.

Theorem 3.8 (Minkowski's decomposition theorem for polyhedra). *There exists an effective procedure that given* $\mathbf{Ax} \leq \mathbf{b}$ *decides whether or not the polyhedron* $Sol(\mathbf{Ax} \leq \mathbf{b})$ *is empty and, if not, it yields a generating matrix* $\mathbf{R}$ *and a vertex matrix* $\mathbf{V}$ *such that:*

$$Sol(\mathbf{Ax} \leq \mathbf{b}) = \{\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq 0\ \} + \{\mathbf{x} \mid \mathbf{x} = \mathbf{V}\boldsymbol{\gamma}, \boldsymbol{\gamma} \geq 0, \Sigma\boldsymbol{\gamma} = 1\ \},$$

and $Sol(\mathbf{Ax} \leq \mathbf{0}) = \{\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq 0\ \}$.

A column of $\mathbf{R}$ is called a *ray*: for any $\mathbf{x}_0 \in Sol(\mathbf{Ax} \leq \mathbf{b})$ and ray $\mathbf{r}$, it turns out that $\mathbf{r}\lambda + \mathbf{x}_0 \in Sol(\mathbf{Ax} \leq \mathbf{b})$ for every $\lambda \geq 0$. A column of $\mathbf{V}$ is called a *vertex*. The set $ConvexHull(\mathbf{V}) = \{\mathbf{x} \mid \mathbf{x} = \mathbf{V}\boldsymbol{\gamma}, \boldsymbol{\gamma} \geq 0, \Sigma\boldsymbol{\gamma} = 1\ \}$, where $\mathbf{V}$ is a matrix or a finite set of vectors, is the convex hull of the vertices, namely the smallest convex set which contains all vertices. A procedure to extract minimal $\mathbf{R}$ and $\mathbf{V}$ is the double description method, also known as the Motzkin-Chernikova-Le Verge algorithm [Motzkin et al. 1953; Chernikova 1965; Le Verge 1992].

Turning back to the LPInfer procedure, the satisfiability test at *Step 2* is performed as part of the construction of the explicit representation of the polyhedron. The maximization problems at *Step 4* can easily be solved given the explicit representation as follows.

LEMMA 3.9. *Consider a characteristic cone $Sol(\mathbf{Ax} \leq \mathbf{0})$, and let $\mathbf{R}$ be its generating matrix. We have that $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{0}\} = 0$ iff $\mathbf{c}^T\mathbf{R} \leq \mathbf{0}$.*

Since in our context $\mathbf{c}$ is always zero except for the i^{th} element, which is 1 or -1, we can conclude that a variable $\mathbf{v}_i$ (the i^{th} variable in $\mathbf{v}$) is bounded from above by 0 (resp., bounded from below by 0) iff all values in $row(\mathbf{R}, i)$ are non-positive (resp., non-negative).

Example 3.10. The Minkowski's form of the homogeneous system in Figure 3 is the following:

$$\left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mid \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ -2 & -1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix}, \lambda_1 \geq 0, \lambda_2 \geq 0 \right\}$$

Intuitively, the two columns in the generating matrix $\mathbf{R}$ correspond to vectors lying on the border of the cone in the graph of Figure 3. Using Lemma 3.9, it is readily checked that when $\mathbf{c}^T$ is one of $(-1\ 0\ 0)$, $(0\ 1\ 0)$, $(0\ 0\ 1)$ or $(0\ 0\ -1)$ then $\mathbf{c}^T\mathbf{R} \leq \mathbf{0}$, i.e. x is bounded from below, y from above, and z from both.

3.3 An Implicit Equality Approach for $\mathcal{BT}_!$

The LPINFER procedure is sound, and it is complete except for the $\Box_r$ types. Thus, we can build on the results of the last subsection by concentrating on inference/checking of type assertions $\mathbf{d} \vdash c \rightarrow x : \Box_r$. In this section, we restrict to consider $\Box_0$, i.e. the ! type. In such a case, a solution to the type inference problem without taking into account parameters is still possible.

Example 3.11. Consider $x : \Box \vdash z = x, z - y = 2, z + y = 0 \rightarrow x :!, y :!, z :!$. Starting from the involved constraint, by Gauss-Jordan elimination, we derive: $x = z$, $y = z - 2$, $2z = 2$ and then $z = 1, y = -1, x = 1$. Hence the type assertion is valid.

Notice that we made no use of $x : \Box$ in proving validity of the type assertion. This fact can be generalized in a result that separates type inference/checking for $\mathcal{BT}_!$ into proofs for the $\mathcal{BT}_\Box$ subset and for the ! type.

THEOREM 3.12 (DEFINITENESS I). $\mathbf{d} \vdash c \rightarrow x :!$ *is valid iff* $\mathbf{d}|_! \vdash c \rightarrow x :!$ *is valid.*

Another intuition from Example 3.11 is that we can use Gauss-Jordan elimination to infer definiteness of variables.

Example 3.13. Consider $x :! \vdash c \rightarrow y :!, z :!$, where c is $z - y = x, z + y = x, k + h = x + 1, x = 0$. We have that $\phi(x :!) \wedge c$ is $x = a, z - y = x, z + y = x, k + h = x + 1, x = 0$. By Gauss-Jordan elimination, we replace z by $x + y$ to obtain $x = a, z = x + y, y = 0, k + h = x, x = 0$. Finally, we replace back y by 0 to obtain $x = a, z = x, y = 0, h = x + 1 - k, x = 0$. Hence $z = x = a$ and $y = 0$ imply that the type assertion is valid.

Notice that Gauss-Jordan elimination provided us with a constructive proof of definiteness, i.e., we can express definite variables as linear expressions over the parameters. In general, consider a constraint c with equalities only, i.e., with geometric representation $\mathbf{A}_c\mathbf{v} = \mathbf{b}_c$. Since $\phi(\mathbf{d}|_!)$ is of the form $\mathbf{x} = \mathbf{a}$, the overall system $\phi(\mathbf{d}|_!) \wedge c$ is a linear system of equalities $\mathbf{x} = \mathbf{a}, \mathbf{A}_c\mathbf{v} = \mathbf{b}_c$. By Gauss-Jordan elimination of variables in $\mathbf{v} \setminus \mathbf{x}$, the system can be transformed into the following form:

$$\mathbf{x} = \mathbf{a}, \mathbf{I}\mathbf{w} = \mathbf{b}' + \mathbf{A}'\mathbf{z} + \mathbf{B}'\mathbf{x}, \mathbf{0} = \mathbf{b}'' + \mathbf{B}''\mathbf{x},$$

where $\mathbf{I}$ is a diagonal matrix, $\mathbf{w}$ and $\mathbf{z}$ are a partition of variables in $\mathbf{v} \setminus \mathbf{x}$ with $\mathbf{z}$ free to assume any value, and $\mathbf{0} = \mathbf{b}'' + \mathbf{B}''\mathbf{x}$ is the condition of satisfiability of the system. It is immediate to observe that, given this form, a variable x is definite in the original system iff it is in $\mathbf{x}$ or it is in $\mathbf{w}$ and it is defined in terms of $\mathbf{b}'$ and $\mathbf{x}$ only. Moreover, notice that we can rule out $\mathbf{x} = \mathbf{a}$ from the system, since we implicitly assume that variables in $\mathbf{x}$ are definite by eliminating only variables in $\mathbf{v} \setminus \mathbf{x}$. Also, we can rule out $\mathbf{b}_c$ since $\mathbf{b}'$ and $\mathbf{b}''$ play no role in definiteness.

Lemma 3.14. *Let c be a satisfiable linear constraint whose geometric representation is $\mathbf{A}_c\mathbf{v}_c = \mathbf{b}_c$, where $\mathbf{v}_c = vars(c)$, and $x \in \mathbf{v}_c$.*

$\mathbf{d}|_! \vdash c \rightarrow x :!$ *is valid iff x is in $\mathbf{x} = vars(\mathbf{d}|_!)$ or, called $\mathbf{I}\mathbf{w} = \mathbf{A}'\mathbf{z} + \mathbf{B}'\mathbf{x}, \mathbf{0} = \mathbf{B}''\mathbf{x}$ the system obtained by Gauss-Jordan elimination from $\mathbf{A}_c\mathbf{v}_c = \mathbf{0}$ of variables in $\mathbf{v}_c \setminus \mathbf{x}$, there exists i such that $x = \mathbf{w}_i$ and $row(\mathbf{A}', i) = \mathbf{0}$.*

Example 3.15. Continuing Example 3.13, Gauss-Jordan elimination of z, y, k, h yields the system:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} z \\ y \\ h \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \end{pmatrix} \begin{pmatrix} k \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} x \end{pmatrix}.$$

The satisfiability condition of the system is $x = 0$. z and y are defined in terms of x only, hence they are defined. k is free to assume any value, hence it cannot be definite. Finally, h is defined in terms of x and k, hence it can assume any value.

So far we considered equality constraints only. In the presence of inequalities, as in $x :! \vdash x \leq y, y \leq x \rightarrow y :!$, Gaussian elimination alone is not enough. We recall the following well-known result (see [Stuckey 1991] or the survey [Greenberg 1996]).

Theorem 3.16 (Implicit equalities). *Assume that $Sol(\mathbf{A}\mathbf{x} \leq \mathbf{b}) \neq \emptyset$. There exists an effective procedure that given $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ yields an equivalent system $\mathbf{A}^=\mathbf{x} = \mathbf{b}^=, \mathbf{A}^+\mathbf{x} \leq \mathbf{b}^+$ such that for every $\mathbf{c}^T\mathbf{x} \leq b$ from $\mathbf{A}^+\mathbf{x} \leq \mathbf{b}^+$ there exists $\mathbf{x}_0 \in Sol(\mathbf{A}\mathbf{x} \leq \mathbf{b})$ such that $\mathbf{c}^T\mathbf{x}_0 < b$.*

As an example, the system $x \leq y, y \leq x$ is equivalent to $x = y$, hence $x :! \vdash x \leq y, y \leq x \rightarrow y :!$ is valid iff $x :! \vdash x = y \rightarrow y :!$ is valid, which can be shown by Gaussian elimination. An inequality $\mathbf{c}^T\mathbf{x} \leq b$ such that $\mathbf{c}^T\mathbf{x}_0 = b$ for every $\mathbf{x}_0 \in Sol(\mathbf{A}\mathbf{x} \leq \mathbf{b})$ is called an implicit equality.

Definition 3.17. *We denote by $ie(\mathbf{A}\mathbf{x} \leq \mathbf{b})$ the linear system $\mathbf{A}^=\mathbf{x} = \mathbf{b}^=$ obtained by any fixed procedure implementing Theorem 3.16.*

Intuitively, $ie(c)$ contains all the information about (implicit) equalities in c. If x is constant in every solution of c, then the same holds for $ie(c)$. Stated otherwise, we can reason about definiteness by restricting to $ie(c)$ only.

```
Input: a type declaration $\mathbf{d}_1$, a linear constraint $c$, and a set of variables $\mathbf{v}$

Step 0. Define $\mathbf{v}_c = vars(c)$, $\mathbf{d} = \mathbf{d}_1|_!$, $\mathbf{x} = vars(\mathbf{d})$.
Step 1. Let $\mathbf{A}_c\mathbf{v}_c \leq \mathbf{b}_c$ be the geometric representation of $c$.
Step 2. Let $\mathbf{A}^=\mathbf{v}_c = \mathbf{b}^=$ be $ie(\mathbf{A}_c\mathbf{v}_c \leq \mathbf{b}_c)$.
Step 3. Rewrite $\mathbf{A}^=\mathbf{v}_c = \mathbf{0}$ into $\mathbf{I}\mathbf{w} = \mathbf{A}'\mathbf{z} + \mathbf{B}'\mathbf{x}, \mathbf{0} = \mathbf{B}''\mathbf{x}$ by Gauss-Jordan elimination of $\mathbf{v}_c \setminus \mathbf{x}$.
Step 4. For every $x : \tau$ in output from LPInfer($\mathbf{d}_1, c, \mathbf{v}$):
Step 5.   If $\tau \neq \square$ or $x \notin \mathbf{v}_c$ Then output "$x : \tau$"
Step 6.   Else
            (a). output "$x$ :!" if $x$ is in $\mathbf{x}$ or there exists $i$ such that $x = \mathbf{w}_i$, and $row(\mathbf{A}', i) = \mathbf{0}$;
            (b). output "$x : \square$" otherwise.
```

Fig. 5. IEInfer procedure

Lemma 3.18. *Let c be a satisfiable linear constraint. $\mathbf{d}|_! \vdash c \rightarrow x$:! is valid iff $\mathbf{d}|_! \vdash ie(c) \rightarrow x$:! is valid.*

Summarizing, Figure 5 reports a procedure, called IEInfer, which first computes $ie(c)$ and then transform it by Gauss-Jordan elimination. Soundness and completeness of IEInfer is stated next.

Theorem 3.19 (IEInfer - soundness and completeness). *IEInfer is sound for the type inference problem, and it is complete for $\mathcal{BT}_!$.*

The IEInfer procedure is parametric to a specific implementation of the $ie()$ function for deriving the set of (implicit) equalities of c. Again, we have two alternatives. One is a Simplex-based algorithm for detecting implicit equalities, such as the one proposed in [Stuckey 1991] and refined in [Refalo 1998]. Another choice is to rely, again, on the Minkowski's form of polyhedra.

Lemma 3.20. *Assume $Sol(\mathbf{A}\mathbf{x} \leq \mathbf{b}) \neq \emptyset$, and let $\mathbf{R}$ and $\mathbf{V}$ be the generating and vertex matrices of $\mathbf{A}\mathbf{x} \leq \mathbf{b}$. The system $\mathbf{A}^=\mathbf{x} = \mathbf{b}^=$ can be calculated as the system of equalities $\mathbf{c}^T\mathbf{x} = b$, where $\mathbf{c}^T\mathbf{x} \leq b$ is from $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ and such that $\mathbf{c}^T\mathbf{R} = \mathbf{0}$ and $\mathbf{c}^T\mathbf{V} = b\mathbf{1}^T$.*

The condition $\mathbf{c}^T\mathbf{R} = \mathbf{0}$, known as the saturation of the inequality by the rays, is typically checked by double description method implementations in order to remove redundant inequalities and to simplify implicit inequalities into equalities. In practice, those implementations maintain and transform both the Minkwoski's form and the explicit form (this is the "double" description) of Theorem 3.16. In addition, they reduce to consider homogeneous systems by transforming forth and back $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ into the form $\mathbf{A}'\mathbf{x}' \leq \mathbf{0}$ with $\mathbf{x} \subseteq \mathbf{x}'$ (see [Goldman 1956; Wilde 1993] for details on the transformation). For homogeneous systems, the additional condition $\mathbf{c}^T\mathbf{V} = b\mathbf{1}^T$ of Lemma 3.20 is trivially satisfied, since $b = 0$ and $\mathbf{V} = \mathbf{0}$.

The implementation that later on we will adopt, namely the `polylib` library [Loechner 2010], performs both the homogeneous transformation and the saturation condition check. In addition to the Minkowski's form, it returns a system of linear inequalities that is equivalent to the original one and such that all implicit equalities are detected. As a practical consequence, there is no need to apply Lemma 3.20 on the output of the `polylib` library.

3.4 A Parameterized Polyhedra Approach for $\mathcal{BT}$

In this section, we reason on the full type system $\mathcal{BT}$ by explicitly dealing with the parameterized system in (2) through an extension of the Minkowski's decomposition theorem. As done in the last subsection, we would like to build on the existing LPInfer procedure. A direct extension of Theorem 3.12, however, does not hold for the $\Box_r$ type when $r > 0$.

Example 3.21. The type assertion $x : \Box_2 \vdash y = x/2 \rightarrow y : \Box_1$ is valid. Intuitively, if $x \in [a, a+2]$ then $x/2 \in [b, b+1]$ for $b = a/2$. However, $\vdash y = x/2 \rightarrow y : \Box_1$ is not valid.

Nevertheless, a similar result can be stated which allows for separating type inference/checking for $\mathcal{BT}$ into proofs for $\mathcal{BT}_\Box$ and for its complement.

Theorem 3.22 (Definiteness II). *Let $\mathcal{B} = \mathcal{BT} \setminus \mathcal{BT}_\Box = \{\Box_s \mid s \in \mathcal{R}, s \geq 0\}$. We have that: $\mathbf{d} \vdash c \rightarrow x : \Box_r$ is valid iff $\mathbf{d}|_\mathcal{B} \vdash c \rightarrow x : \Box_r$ is valid.*

Let us consider now an example which illustrates the Fourier-Motzkin elimination method for linear inequalities applied in the presence of parameters [Keerthi and Sridharan 1990].

Example 3.23. Consider the constraint c defined as $y + x \leq z, y - x \leq z, z \leq y, 0 \leq z, w \leq z$, and the type declaration $z :!$. We start by isolating variable y in $\phi(z :!) \wedge c$, as shown at **(a)** in the figure below.

$$\begin{array}{rcl} & y & \leq z - x \\ & y & \leq z + x \\ z \leq & y & \\ 0 \leq & z & \\ & w & \leq z \\ & z = a & \end{array} \qquad \begin{array}{rcl} z \leq & y & \leq min\{z - x, z + x\} \\ & x = 0 & \\ 0 \leq & z & \\ & w & \leq z \\ & z = a & \end{array} \qquad \begin{array}{rcl} & y = a & \\ & x = 0 & \\ & w & \leq a \\ & z = a & \\ 0 \leq & a & \end{array}$$

(a) **(b)** **(c)**

The bounds for y can then be summarized as: $(*)$ $z \leq y \leq min\{z - x, z + x\}$. Moreover, the inequality $e_1 \leq e_2$ is implied for any e_1, e_2 such that $e_1 \leq y \leq e_2$ is in $(*)$. Actually, the original set of linear inequalities over y is equivalent to $z \leq y \leq min\{z - x, z + x\}$ plus such bounds. The new inequality set is reported at **(b)** in the figure above. By replacing backward $x = 0$ and $z = a$, we end up with the final system at **(c)** in the figure above, where no further elimination is possible. The final system is feasible when the condition $0 \leq a$ holds. In this system, we have $x = 0, y = a, z = a$. Moreover, $w \leq a$ can be rewritten as: $w = -\lambda_1 + a$ for some $\lambda_1 \geq 0$. Put in a geometrical form, the solution set of $\phi(z :!) \wedge c$ is:

$$\left\{ \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \;\middle|\; \begin{array}{c} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \end{pmatrix} \left(\lambda_1 \right) + \begin{pmatrix} 0 \\ a \\ a \\ a \end{pmatrix} \\ \text{for every } \lambda_1 \geq 0, \text{ when } a \geq 0 \end{array} \right\}$$

```
Input: a type declaration $\mathbf{d}_1$, a linear constraint $c$, and a set of variables $\mathbf{v}$
Step 0. Define $\mathbf{v}_c = vars(c)$, $\mathbf{n} = nf(\mathbf{d}_1)$, $\mathbf{d} = \mathbf{n}|_{\mathcal{B}}$, where $\mathcal{B} = \{\Box_r \mid r \in \mathcal{R}, r \geq 0\}$.
Step 1. Let $\mathbf{A}_c\mathbf{v}_c \leq \mathbf{b}$ be the geometric representation of $c$, and $\mathbf{A_d}\mathbf{v}_c \leq \mathbf{B_d}\mathbf{a_d}$ the geometric
        representation of $\phi(\mathbf{d})$.
Step 2. For the parameterized linear system $\begin{pmatrix}\mathbf{A}_c\\ \mathbf{A_d}\end{pmatrix}\mathbf{v}_c \leq \begin{pmatrix}\mathbf{b}_c\\ \mathbf{0}\end{pmatrix} + \begin{pmatrix}\mathbf{0}\\ \mathbf{B_d}\end{pmatrix}\mathbf{a}$, compute
        its generating matrix $\mathbf{R}$ and pairs $(\mathbf{v^a}(1), \mathbf{C}_1\mathbf{a} \leq \mathbf{c}_1), \ldots, (\mathbf{v^a}(k), \mathbf{C}_k\mathbf{a} \leq \mathbf{c}_k)$.
Step 3. For every $x : \tau$ in output from LPINFER$(\mathbf{d}_1, c, \mathbf{v})$:
Step 4.     If $\tau \neq \Box$ or $x \notin \mathbf{v}_c$ Then output "$x : \tau$"
Step 5.     Else let $i$ such that $x = \mathbf{v}_{ci}$:
                (a). output "$x : \Box_r$" if $row(\mathbf{R}, i) = \mathbf{0}$ and for $1 \leq m < n \leq k$, either $P_{m,n} = \emptyset$ or
                     there exists $s$ such that $\mathbf{v^a}(m)_i \simeq_s \mathbf{v^a}(n)_i$ over $P_{m,n}$, and:
                         $r = max(\{0\} \cup \{s \mid 1 \leq m < n \leq k, Sol(P_{m,n}) \neq \emptyset,$
                                 $\mathbf{v^a}(m)_i \simeq_s \mathbf{v^a}(n)_i$ over $P_{m,n}\})$,
                     where $P_{m,n} = \mathbf{C}_m\mathbf{a} \leq \mathbf{c}_m, \mathbf{C}_n\mathbf{a} \leq \mathbf{c}_n$;
                (b). output "$x : \Box$" otherwise.
```

Fig. 6. POLYINFER procedure

Summarizing, the values of x, y and z are univocally determined once the parameter a has been fixed and the system is feasible. Under the same hypotheses, the value of w is bounded from above (by a), but it is not definite. With our notation, $z :! \vdash c \rightarrow x :!, y :!, z :!, w : \sqcap$ is valid.

The final form reached in the example resembles the Minkowski's form for polyhedra, but with a parameterized vector appearing in the vertex matrix. The generalization of the Minkowski's theorem to parameterized polyhedra is provided in [Loechner and Wilde 1997] and implemented in the `polylib` library [Loechner 2010].

THEOREM 3.24 (MINKOWSKI'S THEOREM FOR PARAMETERIZED POLYHEDRA). *For a parameterized linear system* $\mathbf{Ax} \leq \mathbf{b} + \mathbf{Ba}$, *there exist a generating matrix* $\mathbf{R}$ *and finitely many pairs* $(\mathbf{v^a}(1), \mathbf{C}_1\mathbf{a} \leq \mathbf{c}_1), \ldots, (\mathbf{v^a}(k), \mathbf{C}_k\mathbf{a} \leq \mathbf{c}_k)$ *where, for* $i = 1..k$, $\mathbf{v^a}(i)$ *is a vector parametric in* $\mathbf{a}$ *and* $Sol(\mathbf{C}_i\mathbf{a} \leq \mathbf{c}_i) \neq \emptyset$, *such that:*

$$\begin{aligned} Sol(\mathbf{Ax} \leq \mathbf{b} + \mathbf{Ba}, \mathbf{u}) \;=\; & \{\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq 0\ \} + \\ & ConvexHull(\{\mathbf{v^u}(i) \mid i = 1..k, \mathbf{C}_i\mathbf{u} \leq \mathbf{c}_i\ \}), \end{aligned}$$

and $Sol(\mathbf{Ax} \leq \mathbf{0}) = \{\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq 0\ \}$.

The vertex matrix is now replaced by a set of pairs where the first element is a parameterized vertex and the second one is its *validity domain.* For a parameter instance $\mathbf{u}$, the vertex matrix is built from the (instantiated) vertices whose validity domain includes $\mathbf{u}$. The special case $k = 0$ models *empty parameterized polyhedra*, which are empty for any instance of the parameters.

Provided with an explicit form for parameterized polyhedra, we proceed by designing a procedure to check whether a variable has a bounded variability range for every parameter instance. First, we introduce a notion to model the variability range r of two expressions over the solutions of a polyhedron.

DEFINITION 3.25. *We say that* $\mathbf{c}_1^T\mathbf{x} + \alpha_1 \simeq_r \mathbf{c}_2^T\mathbf{x} + \alpha_2$ *over* $\mathbf{Ax} \leq \mathbf{b}$ *if for every* $\mathbf{x}_0 \in Sol(\mathbf{Ax} \leq \mathbf{b})$, $abs(\mathbf{c}_1^T\mathbf{x}_0 + \alpha_1 - \mathbf{c}_2^T\mathbf{x}_0 - \alpha_2) \leq r$, *and for some* $\mathbf{x}_0 \in Sol(\mathbf{Ax} \leq$

$\mathbf{b}$), $abs(\mathbf{c}_1^T\mathbf{x}_0 + \alpha_1 - \mathbf{c}_2^T\mathbf{x}_0 - \alpha_2) = r$.

A direct implementation of the $\simeq_r$ operator relies on standard linear programming problems. Called $M = max\{(\mathbf{c}_1 - \mathbf{c}_2)^T\mathbf{x} + (\alpha_1 - \alpha_2) \mid \mathbf{Ax} \leq \mathbf{b}\}$ and $m = min\{(\mathbf{c}_1 - \mathbf{c}_2)^T\mathbf{x} + (\alpha_1 - \alpha_2) \mid \mathbf{Ax} \leq \mathbf{b}\}$, we have that $\mathbf{c}_1^T\mathbf{x} + \alpha_1 \simeq_r \mathbf{c}_2^T\mathbf{x} + \alpha_2$ over $\mathbf{Ax} \leq \mathbf{b}$ iff $M \in \mathcal{R}, m \in \mathcal{R}$ and $max\{M, -m\} = r$.

Under the hypothesis that we are provided with the Minkowski's form of $\mathbf{Ax} \leq \mathbf{b}$, an alternative implementation is stated by the next result.

Definition 3.26. *The maximum norm of a vector $\mathbf{v} = (v_1, \ldots, v_n)$ is defined as: $\|\mathbf{v}\|_\infty = max\{abs(v_i) \mid i = 1..n\}$.*

Lemma 3.27. *Assume $Sol(\mathbf{Ax} \leq \mathbf{b}) \neq \emptyset$. We have that $\mathbf{c}_1^T\mathbf{x} + \alpha_1 \simeq_r \mathbf{c}_2^T\mathbf{x} + \alpha_2$ over $\mathbf{Ax} \leq \mathbf{b}$ iff called $\mathbf{R}$ and $\mathbf{V}$ the generating and vertex matrices of $\mathbf{Ax} \leq \mathbf{b}$, $(\mathbf{c}_1 - \mathbf{c}_2)^T\mathbf{R} = \mathbf{0}$ and $\|(\mathbf{c}_1 - \mathbf{c}_2)^T\mathbf{V} + (\alpha_1 - \alpha_2)\mathbf{1}^T\|_\infty = r$.*

We point out that Lemma 3.20 is a special case of this result, obtained for $r = 0$.

Also, notice that checking $\mathbf{c}_1 = \mathbf{c}_2$ and $abs(\alpha_1 - \alpha_2) = r$ is a computationally fast calculation, yet being a sufficient condition. In the actual implementation of the $\simeq_r$ operator, we first check such a sufficient condition and, if not satisfied, will check the more elaborated condition of Lemma 3.27.

Consider now the problem of computing the range width of a variable x over a parameterized polyhedron. The next result states that it can be computed by considering the pairs of parameterized vertices of the polyhedron. For each pair, we calculate the range width of x over the intersection of the domains of the two vertices, if not empty. The maximum width over all the pairs of vertices is then the maximum range width of x over any instance of the parameterized polyhedron.

Lemma 3.28. *Consider the Minkowski's form of a non-empty parameterized polyhedron as in Theorem 3.24, and let us denote $P_{m,n} = \mathbf{C}_m\mathbf{a} \leq \mathbf{c}_m, \mathbf{C}_n\mathbf{a} \leq \mathbf{c}_n$. Let $\mathcal{S}_\mathbf{u} = \{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in Sol(\mathbf{Ax} \leq \mathbf{b} + \mathbf{Ba}, \mathbf{u})\}$. For $r \in \mathcal{R}$, the following are equivalent:*

—for every $\mathbf{u}$, for every $x, y \in \mathcal{S}_\mathbf{u}$, $abs(x - y) \leq r$; and there exist $\mathbf{u}$, and $x, y \in \mathcal{S}_\mathbf{u}$, such that $abs(x - y) = r$

—$\mathbf{c}^T\mathbf{R} = \mathbf{0}$; and, for $1 \leq m < n \leq k$, $Sol(P_{m,n}) = \emptyset$ or $\mathbf{c}^T\mathbf{v}^\mathbf{a}(m) \simeq_s \mathbf{c}^T\mathbf{v}^\mathbf{a}(n)$ over $P_{m,n}$ for some $s \leq r$; and,

$$r = max(\{0\} \cup \{s \mid \; 1 \leq m < n \leq k, Sol(P_{m,n}) \neq \emptyset, \mathbf{c}^T\mathbf{v}^\mathbf{a}(m) \simeq_s \mathbf{c}^T\mathbf{v}^\mathbf{a}(n) \text{ over } P_{m,n}\}).$$

Summarizing, Lemma 3.28 provides us with a necessary and sufficient condition for inferring r in a type assertion $\mathbf{d} \vdash c \rightarrow x : \Box_r$, by directly reasoning on the Minkowski's form of the parameterized system in (2). We point out that the `polylib` library provides the validity domains $\mathbf{C}_i\mathbf{a} \leq \mathbf{c}_i$ in their Minkowski's form as well as basic operators on polyhedra, including intersection. Therefore, checking $\mathbf{c}^T\mathbf{v}^\mathbf{a}(m) \simeq_s \mathbf{c}^T\mathbf{v}^\mathbf{a}(n)$ can be implemented using Lemma 3.27.

Example 3.29. The type declaration $z :!, w :!$ and the linear constraint $z + w \geq y, y \geq z, y \geq w, x = z + 1$ give rise to a parameterized polyhedron over parameters $(a \;\; b)$ and variables $(x \;\; y)$ (we omit z and w for space) with generating matrix $\mathbf{0}$, and with pairs of vertices and domains:

$$(\begin{pmatrix} a+1 \\ b \end{pmatrix}, b \geq a \geq 0) \quad (\begin{pmatrix} a+1 \\ a \end{pmatrix}, a \geq b \geq 0) \quad (\begin{pmatrix} a+1 \\ a+b \end{pmatrix}, a \geq 0 \wedge b \geq 0).$$

Let us reason about definiteness of variables x and y by using Lemma 3.28. x is definite, since $a+1 \simeq_0 a+1$ over any polyhedron. In fact, $a+1$ expressed as a linear function of the parameters is $(1\ \ 0)\begin{pmatrix} a \\ b \end{pmatrix} + 1$. The conclusion then holds by Lemma 3.27 using the sufficient condition $\mathbf{c}_1 = (1\ \ 0) = \mathbf{c}_2$ and $abs(\alpha_1 - \alpha_2) = 0$.

Consider now y. For the first two vertices, we have $b \neq a$, thus we cannot use the sufficient condition as in the last case. Since the intersection of the domains of the vertices, namely $a = b \geq 0$, is not empty, by Lemma 3.27 we proceed by computing its generating and vertex matrices. The vertex matrix is $\mathbf{0}$, so we simply have:

$$Sol(a = b \geq 0) = \{(a\ b) \mid \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \begin{pmatrix} \lambda_1 \end{pmatrix}, \lambda_1 \geq 0\},$$

and we calculate:

$$\|(\begin{pmatrix} 0 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 \end{pmatrix})\begin{pmatrix} 1 \\ 1 \end{pmatrix} + 0\ \mathbf{1}^T\|_\infty = 0.$$

Therefore, $a \simeq_0 b$ over $a = b \geq 0$. Consider now the first and the third vertex. Since $b \neq a + b$, we compute, as before, the Minkowski's form of the intersection of their domains:

$$Sol(b \geq a \geq 0) = \{(a\ b) \mid \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \lambda_1, \lambda_2 \geq 0\}.$$

The vertex matrix $\mathbf{V}$ clearly satisfies $\|(\mathbf{c}_1 - \mathbf{c}_2)^T\mathbf{V} + (\alpha_1 - \alpha_2)\mathbf{1}^T\|_\infty = 0$. However, for the generating matrix $\mathbf{R}$, the condition $(\mathbf{c}_1 - \mathbf{c}_2)^T\mathbf{R} = \mathbf{0}$ does not hold:

$$(\begin{pmatrix} 0 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 1 \end{pmatrix})\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 \end{pmatrix}.$$

Summarizing, by Lemma 3.27, $b \not\simeq_r a + b$ for any r, and then, by Lemma 3.28, y cannot be typed as $\Box_r$ for any r.

The overall procedure, called POLYInfer, is shown in Figure 6. The procedure is sound and complete for inferring validity of type assertions.

Theorem 3.30 (POLYInfer - soundness and completeness). POLYInfer *is sound for the type inference problem, and it is complete for* $\mathcal{BT}$.

3.5 Extension to Strict Inequalities and to Disequalities

So far, we considered equality and non-strict inequality primitive constraints. A generalized linear constraint admits also primitive constraint over the operators $<$, $>$ (strict inequalities) and $\neq$ (disequalities). Without any loss of generality, we write a generalized constraint as $c \wedge \bigwedge_{i=1}^m e_i \neq \alpha_i$, where c is a linear constraint and for $i = 1..m$, $e_i \neq \alpha_i$ is a disequality. We now extend type assertions to admit generalized constraints. The next result shows that validity of type assertions for a satisfiable generalized constraint can be reduced to validity of the type assertions over the linear constraint obtained by removing the disequalities in it.

THEOREM 3.31. *Let $g = c \wedge \bigwedge_{i=1}^{m} e_i \neq \alpha_i$ be a satisfiable generalized linear constraint. $\mathbf{d}_1 \vdash g \rightarrow \mathbf{d}_2$ is valid iff $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is valid.*

The conclusion does not hold for unsatisfiable constraints.

Example 3.32. The type assertion $\vdash x < 3, x \geq 3, y \leq 0 \rightarrow y :!$ is valid, since the generalized constraint is unsatisfiable. When removing the disequalities (namely, turning $x < 3$ into $x \leq 3$), only validity of $\vdash x \leq 3, x \geq 3, y \leq 0 \rightarrow y : \sqcap$ follows.

Checking satisfiability of g can easily be accomplished. In fact, by independence of negative constraints [Lassez and McAllon 1992], it reduces to show that $Sol(\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c) \neq \emptyset$, and that for every $e \neq \alpha$ in g, $e \simeq_0 \alpha$ over $\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c$ does not hold. Lemma 3.27 provides us with a procedure to show that by using the explicit form of polyhedra. Alternatively, the same result can be obtained by linear programming. Called $M = max\{e - \alpha \mid \mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c\}$ and $m = min\{e - \alpha \mid \mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c\}$, we have already noted that $e \simeq_0 \alpha$ holds iff $M, m \in \mathcal{R}$ and $max\{M, -m\} = 0$, i.e., iff $M = m = 0$. When the generalized constraint contains strict inequalities but no disequality, a single linear programming problem can be devised to cover all strict inequalities at once (see [Greenberg 1996, Theorem 4]).

3.6 Extension to Parametric Types $\mathcal{PT}$

The next example introduces type variables in type assertions.

Example 3.33. When writing the following (valid) type assertion:

$$x : \square_1 \vdash y = x + 2, x \leq z \leq x + 2 \rightarrow y : \square_1, z : \square_3$$

it is natural to observe that it holds in more general terms: the range of variability for y is the same as x, and the range of variability for z is the one for x plus 2. By introducing a "type variable" s, and assuming $s \geq 0$, we can write the general statement as: $x : \square_s \vdash y = x + 2, x \leq z \leq x + 2 \rightarrow y : \square_s, z : \square_{s+2}$.

Notice that the kind of involved expressions can be linear combinations of type variables, as in $x : \square_s, y : \square_1 \vdash z = 2x + y \rightarrow z : \square_{2s+1}$.

Let us formally introduce type variables in the syntax and semantics of type assertions.

DEFINITION 3.34. *Let $\mathcal{V}$ be a set of variables distinct from constraint variables, called* type variables. *A parametric type is $\square_e$, where $e = \mathbf{c}^T\mathbf{a} + r$ is a linear expression over variables $\mathbf{a}$ from $\mathcal{V}$. We define the set of types $\mathcal{PT}$ as $\mathcal{BT}$ augmented with parametric types.*

The syntax of type declarations and type assertions readily extend to $\mathcal{PT}$.

The semantics of type assertions over $\mathcal{PT}$ is defined by extending the ϕ and v functions in Definition 2.4 as follows:

$$\phi(x : \square_e) = a \leq x \wedge x \leq a + e \wedge 0 \leq e \qquad v(x : \square_e) = \{a\} \cup vars(e).$$

For a type assertion $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$, a type variable that appears (with non-zero coefficient) in $\mathbf{d}_2$ but that does not appear in $\mathbf{d}_1$ is called a local type variable.

Intuitively, type variables in parametric[3] types are *compiled* into parameters of the parameterized linear system underlying the type assertion. Notice that the conjunct $0 \leq e$ is syntactically satisfied for $\Box_r$ in the $\mathcal{BT}$ type system, since we assumed $r \in \mathcal{R}$ and $r \geq 0$. Therefore, the above $\phi()$ and $\upsilon()$ definitions are conservative extensions of the ones for $\Box_r$. Also, notice that, for a type assertion $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$, the restriction $\upsilon(\mathbf{d}_2) \setminus \upsilon(\mathbf{d}_1)$ in Definition 2.4 (1) is now fundamental, since a same type variable can appear both in $\mathbf{d}_1$ and in $\mathbf{d}_2$.

Example 3.35. The type assertion $x : \Box_s \vdash y = x + 2 \rightarrow y : \Box_p$ states that if x has a range of variability of s then y has some range of variability p, i.e., there exists a and $p \geq 0$ such that $a \leq y \leq a + p$. But this is equivalent to require a and b such that $a \leq y \leq b$, i.e., to state that $x : \Box_s \vdash y = x + 2 \rightarrow y : \Box$ is valid.

Local type variables in a type assertion are existentially quantified. Therefore, there is no loss of generality in assuming no local type variable in a type assertion $\mathbf{d}_1 \vdash c \rightarrow x : \Box_e$.

LEMMA 3.36. *A type assertion $\mathbf{d}_1 \vdash c \rightarrow x : \Box_e$ with local type variables is valid iff $\mathbf{d}_1 \vdash c \rightarrow x : \Box$ is valid.*

In general, admitting local type variables makes type assertions non-compositional, in the sense highlighted by the following example.

Example 3.37. The type assertion $\vdash 0 \leq x \leq 2, 0 \leq y \leq 1 \rightarrow x : \Box_s, y : \Box_{1-s}$ is invalid. In fact, x has a variability range of 2 in the solutions of the linear constraint, whilst the existentially quantified s is required $s \geq 0$ and $1 \geq s$. On the contrary, by Lemma 3.36, both $\vdash 0 \leq x \leq 2, 0 \leq y \leq 1 \rightarrow x : \Box_s$ and $\vdash 0 \leq x \leq 2, 0 \leq y \leq 1 \rightarrow y : \Box_{1-s}$ are valid.

The property of compositionality holds for type assertions without local type variables. For the rest of this section we restrict to type assertions of the form $\mathbf{d}_1 \vdash c \rightarrow x : \Box_e$.

LEMMA 3.38. *A type assertion $\mathbf{d}_1 \vdash c \rightarrow (\mathbf{d}_2, \mathbf{d}_3)$ with no local type variable is valid iff $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ and $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_3$ are valid.*

The introduction of parametric types breaks also the basic Lemma 3.4, i.e., satisfiability of the parameterized linear system $\mathcal{P} = \phi(\mathbf{d}) \wedge c$ in (2) does not reduce to satisfiability of c.

Example 3.39. For the type assertion $x : \Box_s, y : \Box_{-s-1} \vdash \texttt{true} \rightarrow x :!$, we have that $\phi(x : \Box_s, y : \Box_{-s-1}) \wedge \texttt{true}$ is $a \leq x \leq a+s, b \leq y \leq b-s-1, 0 \leq s, 0 \leq -s-1$. In particular, the last two primitive constraints can be rewritten as: $s+1 \leq 0 \leq s$, which is unsatisfiable.

Lemma 3.4 conservatively generalizes to parametric types by adding the condition that both c and $\phi(\mathbf{d})$ are satisfiable. For a type declaration $\mathbf{d}$ in $\mathcal{BT}$, $\phi(\mathbf{d})$ is always satisfiable.

[3]Notice that we will use the word 'parametric' for types and 'parameterized' for linear systems in order to resolve possible linguistic ambiguities in the text.

LEMMA 3.40. *Consider the parameterized linear system $\mathcal{P} = \phi(\mathbf{d}) \wedge c$ in (2), with $\mathbf{d}$ type declaration in $\mathcal{PT}$. There exists a parameter instance $\mathbf{u}$ such that $Sol(\mathcal{P}, \mathbf{u}) \neq \emptyset$ iff $Sol(\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c) \neq \emptyset$ and $Sol(\mathbf{A_d}\mathbf{v} \leq \mathbf{B_d}\mathbf{a}) \neq \emptyset$.*

Concerning type inference, the following example shows that complete procedures have to lift to piecewise linear expressions, which is beyond our syntax of type declarations.

Example 3.41. What parametric type τ can be inferred for the type assertion $x : \Box_s \vdash 0 \leq x \leq 2 \rightarrow x : \tau$?

When $0 \leq s \leq 2$, the range width s for x is stricter than the range width 2 imposed by the constraint $0 \leq x \leq 2$. Hence, $\tau = \Box_s$ should be inferred. Nevertheless, inferring $\tau = \Box_2$ is sound, i.e., it yields a valid type assertion.

When $s \geq 2$, the range width 2 imposed by $0 \leq x \leq 2$ is stricter than s . Hence, $\tau = \Box_2$ should be inferred. Nevertheless, inferring $\tau = \Box_s$ is sound.

Summarizing, sound procedures for type inference could return either $\tau = \Box_s$ or $\tau = \Box_2$. A complete procedure, however, should return:

$$\tau = \begin{cases} \Box_2 & \text{if } s \geq 2 \\ \Box_s & \text{otherwise} \end{cases}$$

i.e., a piecewise linear expression, which is beyond our syntax.

The notions of $lub()$ and normal form of type declarations are also affected by the same problem.

Example 3.42. By reasoning as in the last example, $lub(\{\Box_s, \Box_2\})$ cannot be expressed as $\Box_e$ for some linear expression e. Rather, we should generalize the syntax of parametric types to allow $\Box_{min\{s,2\}}$. Similarly, one concludes that $nf(x : \Box_s, x : \Box_2)$ cannot be expressed with the syntax of linear expressions.

Concerning type checking, however, we are in the position to design a decision procedure. First, an extension of the Theorem 3.22 (Definiteness II) holds.

THEOREM 3.43 (DEFINITENESS III). *Let $\mathcal{B} = \mathcal{PT} \setminus \mathcal{BT}_\Box$. A type assertion $\mathbf{d} \vdash c \rightarrow x : \Box_e$ with no local type variable is valid iff $\mathbf{d}|_\mathcal{B} \vdash c \rightarrow x : \Box_e$ is valid.*

On the contrary, Theorem 3.12 (Definiteness I) does not extend to parametric types, i.e., definiteness cannot be dealt with separately.

Example 3.44. The type assertion $x : \Box_s, x : \Box_{-s} \vdash \texttt{true} \rightarrow x :!$ is valid. In fact, the corresponding formula to be shown is:

$$\forall a, b, s\ \exists c\ \forall x\ (a \leq x \leq a + s \wedge b \leq x \leq b - s \wedge s \geq 0 \wedge\ -s \geq 0 \rightarrow x = c).$$

From the left hand side of the implication, we obtain: $s = 0 \wedge a + b \leq 2x \leq a + b$, and then $x = (a + b)/2$ is the only value that x can assume.

However, the type assertion $\vdash \texttt{true} \rightarrow x :!$ is not valid. This shows that Theorem 3.12 does not extend to parametric types. As a consequence, completeness of the CHECK(IEINFER) procedure is lost for type assertions $\mathbf{d} \vdash c \rightarrow x :!$ such that parametric types appear in $\mathbf{d}$.

Next, we revisit Lemma 3.27 and Lemma 3.28.

```
Input: a type assertion $\mathbf{d}_1 \vdash c \rightarrow x : \Box_e$ with no local type variable
Step 0. Define $\mathbf{d} = \mathbf{d}_1|_{\mathcal{B}}$, with $\mathcal{B} = \mathcal{PT} \setminus \mathcal{BT}_\Box$, $\mathbf{v} = vars(c) \cup vars(\mathbf{d}) \cup \{x\}$, and let $i$ s.t. $x = \mathbf{v}_i$.
Step 1. Let $\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c$ be the geometric representation of $c$, and $\mathbf{A_d}\mathbf{v} \leq \mathbf{B_d}\mathbf{a_d}$ the geometric
        representation of $\phi(\mathbf{d})$.
Step 2. If $Sol(\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c) = \emptyset$ or $Sol(\mathbf{A_d}\mathbf{v} \leq \mathbf{B_d}\mathbf{a_d}) = \emptyset$ Then output "Valid"
        Else
Step 3.     For the parameterized linear system $\begin{pmatrix} \mathbf{A}_c \\ \mathbf{A_d} \end{pmatrix} \mathbf{v} \leq \begin{pmatrix} \mathbf{b}_c \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{B_d} \end{pmatrix} \mathbf{a}$, compute
            its generating matrix $\mathbf{R}$ and pairs $(\mathbf{v^a}(1), \mathbf{C}_1\mathbf{a} \leq \mathbf{c}_1), \ldots, (\mathbf{v^a}(k), \mathbf{C}_k\mathbf{a} \leq \mathbf{c}_k)$:
Step 4.     (a). output "Valid" if
                (i) $row(\mathbf{R}, i) = \mathbf{0}$;
                (ii) and for every $1 \leq m < n \leq k$, $\mathbf{v^a}(m)_i \preceq_e \mathbf{v^a}(n)_i$ over $\mathbf{C}_m\mathbf{a} \leq \mathbf{c}_m, \mathbf{C}_n\mathbf{a} \leq \mathbf{c}_n$;
                (iii) and for $i = 1..k$, $min\{e \mid \mathbf{C}_i\mathbf{a} \leq \mathbf{c}_i\} \geq 0$;
            (b). output "Not valid" otherwise.
```

Fig. 7. ParCheck procedure

Definition 3.45. *Let $e = \mathbf{c}_e^T\mathbf{x} + r$. We say that $\mathbf{c}_1^T\mathbf{x} + \alpha_1 \preceq_e \mathbf{c}_2^T\mathbf{x} + \alpha_2$ over $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ if for every $\mathbf{x}_0 \in Sol(\mathbf{A}\mathbf{x} \leq \mathbf{b})$, $abs(\mathbf{c}_1^T\mathbf{x}_0 + \alpha_1 - \mathbf{c}_2^T\mathbf{x}_0 - \alpha_2) \leq \mathbf{c}_e^T\mathbf{x}_0 + r$.*

$\preceq_e$ is a weakening of the relation $\simeq_r$ from Definition 3.25 in two ways. First, any upper bound is now required, whilst $\simeq_r$ holds for the least upper bound. Second, the upper bound may be expressed using variables of the system, as in $x \preceq_z x + z$. As in the case of $\simeq_r$, the $\preceq_e$ relation can be checked by means of linear programming problems or, if we are provided with the Minkowski's form of $\mathbf{A}\mathbf{x} \leq \mathbf{b}$, by means of the following result.

Lemma 3.46. *Let $e = \mathbf{c}_e^T\mathbf{x} + r$, and assume $Sol(\mathbf{A}\mathbf{x} \leq \mathbf{b}) \neq \emptyset$.*
We have that $\mathbf{c}_1^T\mathbf{x} + \alpha_1 \preceq_e \mathbf{c}_2^T\mathbf{x} + \alpha_2$ over $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ iff called $\mathbf{R}$ and $\mathbf{V}$ the generating and vertex matrices of $\mathbf{A}\mathbf{x} \leq \mathbf{b}$, $(\mathbf{c}_1 - \mathbf{c}_2 - \mathbf{c}_e)^T\mathbf{R} \leq \mathbf{0}$ and $(\mathbf{c}_2 - \mathbf{c}_1 - \mathbf{c}_e)^T\mathbf{R} \leq \mathbf{0}$ and $(\mathbf{c}_1 - \mathbf{c}_2 - \mathbf{c}_e)^T\mathbf{V} + (\alpha_1 - \alpha_2 - r)\mathbf{1}^T \leq \mathbf{0}$ and $(\mathbf{c}_2 - \mathbf{c}_1 - \mathbf{c}_e)^T\mathbf{V} + (\alpha_2 - \alpha_1 - r)\mathbf{1}^T \leq \mathbf{0}$.

Notice that, when $e = r$, then the necessary and sufficient condition can be stated in a simpler form as: $(\mathbf{c}_1 - \mathbf{c}_2)^T\mathbf{R} = \mathbf{0}$ and $\|(\mathbf{c}_1 - \mathbf{c}_2)^T\mathbf{V} + (\alpha_1 - \alpha_2)\mathbf{1}^T\|_\infty \leq r$, which closely resembles the one in Lemma 3.27. Lemma 3.28 is extended to the checking problem in the presence of parametric types as follows.

Lemma 3.47. *Consider the Minkowski's form of a non-empty parameterized polyhedron as in Theorem 3.24, and let us denote $P_{m,n} = \mathbf{C}_m\mathbf{a} \leq \mathbf{c}_m, \mathbf{C}_n\mathbf{a} \leq \mathbf{c}_n$.*
Let $\mathcal{S}_\mathbf{u} = \{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in Sol(\mathbf{A}\mathbf{x} \leq \mathbf{b} + \mathbf{B}\mathbf{a}, \mathbf{u})\}$, and $e = \mathbf{c}_e^T\mathbf{a} + r$.
We have that for every $\mathbf{u}$, for every $x, y \in \mathcal{S}_\mathbf{u}$, $abs(x - y) \leq \mathbf{c}_e^T\mathbf{u} + r$ iff the following hold: (i) $\mathbf{c}^T\mathbf{R} = \mathbf{0}$; (ii) and, for $1 \leq m < n \leq k$, $\mathbf{c}^T\mathbf{v^a}(m) \preceq_e \mathbf{c}^T\mathbf{v^a}(n)$ over $P_{m,n}$; (iii) and, for $i = 1..k$, $min\{e \mid \mathbf{C}_i\mathbf{a} \leq \mathbf{c}_i\} \geq 0$.

Example 3.48. Given the Fahrenheit-Celsius constraint c from the introduction $fr = 9/5cl + 32$, let us show that $cl : \Box_s \vdash c \rightarrow fr : \Box_{1.8s}$ is valid by exploiting Lemma 3.47. The parameterized linear system $\phi(cl : \Box_s) \wedge c$ is $a \leq cl, cl \leq a + s, 0 \leq s, fr = 9/5cl + 32$, and the corresponding parameterized polyhedron is non-empty (e.g., for the parameter values $a = s = 0$ there is a solution: $cl = 0, fr = 32$).

Fixed $\mathbf{x} = \begin{pmatrix} fr \\ cl \end{pmatrix}$ and $\mathbf{a} = \begin{pmatrix} a \\ s \end{pmatrix}$, its Minkowski's form consists of:

$$\mathbf{R} = \begin{pmatrix} 9 & -9 \\ 5 & -5 \end{pmatrix},$$

and of the following two parameterized vertices:

$$\mathbf{v^a}(1) = \begin{pmatrix} 9/5a + 32 \\ a \end{pmatrix}, \text{if } s \geq 0 \qquad \mathbf{v^a}(2) = \begin{pmatrix} 9/5s + 9/5a + 32 \\ s + a \end{pmatrix}, \text{if } s \geq 0.$$

For $\mathbf{c} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $\mathbf{c}_e = \begin{pmatrix} 0 \\ 1.8 \end{pmatrix}$ and $r = 0$, we have $\mathbf{c}^T\mathbf{x} = fr$ and $e = \mathbf{c}_e^T\mathbf{a} + r = 1.8s$. Therefore, showing that $cl : \Box_s \vdash c \rightarrow fr : \Box_{1.8s}$ is valid consists of showing conditions *(i)-(iii)* from Lemma 3.47.

(i) is immediate. *(iii)* trivially holds, since $min\{1.8s \mid s \geq 0\} \geq 0$. Finally, let us show *(ii)*, namely $\mathbf{c}^T\mathbf{v^a}(1) = 9/5a + 32 \preceq_e 9/5s + 9/5a + 32 = \mathbf{c}^T\mathbf{v^a}(2)$ over $P_{1,2} = Sol(s \geq 0)$. By Definition 3.45, this amounts at showing that for every $s \geq 0$, $abs(9/5s) \leq 1.8s$. This is trivially true since $9/5 = 1.8$.

As in the case of Lemma 3.28, we point out that the `polylib` library provides the validity domains $\mathbf{C}_i\mathbf{a} \leq \mathbf{c}_i$ in their Minkowski's form as well as basic operators on polyhedra, including intersection. Therefore, condition *(ii)* can be implemented using Lemma 3.46. Analogously, condition *(iii)* can be checked by solving linear programming problems or as follows.

LEMMA 3.49. *Let $e = \mathbf{c}_e^T\mathbf{x} + r$, and assume $Sol(\mathbf{Ax} \leq \mathbf{b}) \neq \emptyset$. We have that $min\{e \mid \mathbf{Ax} \leq \mathbf{b}\} \geq 0$ iff called $\mathbf{R}$ and $\mathbf{V}$ the generating and vertex matrices of $\mathbf{Ax} \leq \mathbf{b}$, $\mathbf{c}_e^T\mathbf{R} \geq \mathbf{0}$ and $\mathbf{c}_e^T\mathbf{V} + r\mathbf{1}^T \geq \mathbf{0}$.*

Condition *(iii)* covers contrived instances of types variables, as shown in the next example. We refer the reader to Appendix A.7 for additional details.

Example 3.50. Consider the type assertion $x : \Box_s, y : \Box_r \vdash z = 0 \rightarrow z : \Box_{s-r}$. By instantiating $s = 0, r = 1$, we obtain $x :!, y : \Box_1 \vdash z = 0 \rightarrow z : \Box_{-1}$, which is syntactically malformed as a type assertion in $\mathcal{BT}$ (since the syntactic requirement $-1 \geq 0$ does not hold). As a type assertion in $\mathcal{PT}$, however, it is well-formed, yet invalid. In fact, $\phi(x :!, y : \Box_1) \wedge z = 0$ is $a \leq x \leq a + 0 \wedge 0 \geq 0 \wedge b \leq y \leq b + 1 \wedge 1 \geq 0 \wedge z = 0$, which admits solutions. However, $\phi(z : \Box_{-1})$ is $c \leq z \leq c - 1 \wedge -1 \geq 0$, which is unsatisfiable.

Figure 7 summarizes the checking procedure PARCHECK, in the case of input type assertion $\mathbf{d}_1 \vdash c \rightarrow x : \Box_e$. We will implement (see Sect. 5) *Step 4 (a)(ii)* by means of Lemma 3.46, and *Step 4 (a)(iii)* by means of Lemma 3.49.

THEOREM 3.51 (PARCHECK - SOUNDNESS AND COMPLETENESS). PARCHECK *is a decision procedure for the type checking problem of type assertions $\mathbf{d}_1 \vdash c \rightarrow x : \Box_e$ with no local type variable.*

When local type variables exists, $\mathbf{d}_1 \vdash c \rightarrow x : \Box_e$ is equivalent to $\mathbf{d}_1 \vdash c \rightarrow x : \Box$, as shown in Lemma 3.36. In general, for a type assertion $\mathbf{d}_1 \vdash c \rightarrow x : \tau$ with $\tau \in \mathcal{BT}_\Box$, the checking procedure can resort to CHECK(LPINFER) by reducing the problem to type assertions for which CHECK(LPINFER) is sound and complete.

Lemma 3.52. *Let $\mathbf{d}_1$ be a type declaration (in $\mathcal{PT}$) such that $\phi(\mathbf{d}_1)$ is satisfiable. Let $\mathbf{d}'_1$ be the type declaration (in $\mathcal{BT}_\Box$) obtained from $\mathbf{d}_1$ by replacing every $y : \Box_e$ with $y : \Box$. For $\tau \in \mathcal{BT}_\Box$, $\mathbf{d}_1 \vdash c \to x : \tau$ is valid iff $\mathbf{d}'_1 \vdash c \to x : \tau$ is valid.*

3.7 Computational Complexity Issues

Let us consider the issue of computational complexity of the type checking problem.

First, we observe that the LPInfer procedure has a polynomial time complexity when a polynomial time algorithm (such as the one in [Khachiyan 1979]) is adopted for solving the required linear programming problems. By Theorem 3.7, Check(LPInfer) is then a polynomial time decision procedure for type assertions $\mathbf{d}_1 \vdash c \to \mathbf{d}_2$ such that types in $\mathbf{d}_1$ belong to $\mathcal{BT}$, and types in $\mathbf{d}_2$ belong to $\mathcal{BT}_\Box$. The conclusion can be strengthened to $\mathcal{BT}_!$ since the additional computations of IEInfer consist of extracting implicit equalities and of performing Gaussian elimination, both of which have polynomial time complexity.

Theorem 3.53. *The problem of checking whether $\mathbf{d}_1 \vdash c \to \mathbf{d}_2$ is valid is polynomial time decidable, if all types in $\mathbf{d}_2$ belong to $\mathcal{BT}_!$.*

Notice that the instantiation of LPInfer using the Minkowski's form of polyhedra has an exponential time complexity, since the number of vertices of a (not even parameterized) polyhedron can be exponential in the worst case. A fortiori, neither Check(LPInfer) nor Check(IEInfer), can have polynomial time complexity when they rely on computing the Minkowski's form of polyhedra.

Example 3.54. Consider linear constraints of the form: $cw_n = 0 \leq x_1 \leq 1, \ldots, 0 \leq x_n \leq 1$, for $n \geq 0$. The polyhedra $Sol(cw_n)$ is called an hypercube, since it consists of 2^n vertices of the form $(a_1\ a_2\ \ldots\ a_n)$ such that for $i = 1..n$, $a_i \in \{0, 1\}$. Computing the vertex matrix of $Sol(cw_n)$ has then an exponential time computational complexity.

When restricting to homogeneous systems, as required in LPInfer, the worst case is still exponential.

Example 3.55. The transformation of [Goldman 1956; Wilde 1993] over cw_n yields the homogeneous constraint $hw_n = 0 \leq x_1, 0 \leq \xi - x_1, \ldots, 0 \leq x_n, 0 \leq \xi - x_n, 0 \leq \xi$. For each vertex $\mathbf{v}$ of $Sol(cw_n)$, $Sol(hw_n)$ has a ray of the form $(\lambda\mathbf{v}, \lambda)$, for some $\lambda > 0$. Therefore, complexity of computing the vertex matrix of $Sol(cw_n)$ has been shifted to the computation of the generating matrix of $Sol(hw_n)$.

We refer the reader to [Borgwardt 2007] for a discussion of the average-case complexity of the double description method, the core algorithm for the extraction of the Minkowski's form of (parameterized) polyhedra. Under a reasonable stochastic model over the input linear system of inequalities, the average computational complexity of the method is shown to be polynomial in the number of inequalities (but not in the number of variables, which is the case of the last two examples).

Let us consider now Check(POLYInfer) and ParCheck. They require to compute the pairs $(\mathbf{v}^{\mathbf{a}}(1), \mathbf{C}_1\mathbf{a} \leq \mathbf{c}_1), \ldots, (\mathbf{v}^{\mathbf{a}}(k), \mathbf{C}_k\mathbf{a} \leq \mathbf{c}_k)$ as from Theorem 3.24. The algorithm proposed in [Loechner and Wilde 1997] for the purpose consists of first computing the vertices of $Sol(\mathcal{P})$ where $\mathcal{P}$ is the linear system (2) in the space of variables plus parameters; then to project each vertex over the parameter space.

The first step leads to conclude non-polynomial time complexity, as for LPInfer. The second step leads to conclude that the number of parameterized vertices k is bounded by the number of vertices in $Sol(\mathcal{P})$. Since the projection at the second step requires polynomial time [Loechner and Wilde 1997, Section 5.3], under the assumptions of [Borgwardt 2007], the computation of the parameterized vertices has a polynomial time average complexity in the number of inequalities.

Later on in Section 5 we report an experimental evaluation of the execution times of the various decision procedures over several testbeds.

4. MODING CLP($\mathcal{R}$) PROGRAMS

Constraint logic programming provides an elegant scheme for dynamically building complex constraints by exploiting recursion, non-determinism and intertwined constraint generation & solving. It is then a natural candidate as an application area for the type system theory developed in the previous sections. Here, we propose an extension of the notion of well-moding from pure logic programming.

4.1 Well-moding

Modes for pure logic programs assign to every predicate argument an input-output behavior. Input means that the predicate argument is ground on calls. Output means that it is ground on answers. As discussed in the introduction, groundness (i.e., definiteness) is restrictive in the CLP context. Based on types, we can extend the notion of moding to upper and/or lower bounds as well.

Definition 4.1 (Moding). *A* mode *for an n-ary predicate p is a function d_p from $\{1, \ldots, n\}$ to $\mathcal{BT} \times \mathcal{BT}$. We write d_p as $p(\tau_1 \times \mu_1, \ldots, \tau_n \times \mu_n)$, where $d_p(i) = (\tau_i, \mu_i)$ for $i = 1..n$.*

A mode *for a CLP($\mathcal{R}$) program P is a set of modes, one for each predicate in P. For an atom $p(\mathbf{x})$, we write $p(\mathbf{x} : \boldsymbol{\tau} \times \boldsymbol{\mu})$ to denote that $\mathbf{x}$ is the collection of variables occurring in the atom, and $p(\boldsymbol{\tau} \times \boldsymbol{\mu})$ is the mode of p. Types in $\boldsymbol{\tau}$ are called* input *modes while types in $\boldsymbol{\mu}$ are called* output *modes.*

By fixing a predicate argument mode to $! \times !$ or to $\star \times !$ we get back to the logic programming input-output behavior, respectively denoted by $+$ and $-$. Also the mode $\star \times \star$ means that the predicate argument is of no relevance in the analysis, a case denoted by ? in logic programming. We recall that programs are assumed in flat form. Several notions of moding have been proposed (see [Apt 1997] for a review) for logic programs. We extend here the notion of well-moding to CLP($\mathcal{R}$).

Definition 4.2 (Well-moding). *A CLP($\mathcal{R}$) clause:*

$$p_0(\mathbf{x}_0 : \boldsymbol{\tau}_0 \times \boldsymbol{\mu}_0) \leftarrow c, p_1(\mathbf{x}_1 : \boldsymbol{\tau}_1 \times \boldsymbol{\mu}_1), \ldots, p_n(\mathbf{x}_n : \boldsymbol{\tau}_n \times \boldsymbol{\mu}_n)$$

is well-moded if the following type assertions are valid:

$$\text{for } i = 1..n, \quad \mathbf{x}_0 : \boldsymbol{\tau}_0, \mathbf{x}_1 : \boldsymbol{\mu}_1, \ldots, \mathbf{x}_{i-1} : \boldsymbol{\mu}_{i-1} \vdash c \rightarrow \mathbf{x}_i : \boldsymbol{\tau}_i,$$
$$\mathbf{x}_0 : \boldsymbol{\tau}_0, \mathbf{x}_1 : \boldsymbol{\mu}_1, \ldots, \mathbf{x}_n : \boldsymbol{\mu}_n \vdash c \rightarrow \mathbf{x}_0 : \boldsymbol{\mu}_0.$$

A CLP($\mathcal{R}$) program P is well-moded if every clause in it is well-moded.

The definition of well-moding constrains the "flow of data" through the atoms in a clause to follow a left-to-right sequence, starting with input position in the head of the clauses and ending with output positions in the head.

Example 4.3. The `MORTGAGE` program is well-moded with moding `mortgage(!×!, □×!, !×!, ⋆×!)`, which is intended for covering the first two queries in the introduction. For clause `(m1)` we have to show that:

$$\texttt{P} :!, \texttt{T} : \square, \texttt{R} :!, \texttt{B} : \star \vdash \texttt{T = 0, B = P} \rightarrow \texttt{P} :!, \texttt{T} :!, \texttt{R} :!, \texttt{B} :!$$

is valid, which is immediate. For clause `(m2)`, called c the constraint `T >= 1, NP = P + P * 0.05 - R, NT = T - 1`, we have to show validity of:

$$\texttt{P} :!, \texttt{T} : \square, \texttt{R} :!, \texttt{B} : \star \vdash c \rightarrow \texttt{NP} :!, \texttt{NT} : \square, \texttt{R} :!, \texttt{B} : \star$$
$$\texttt{P} :!, \texttt{T} : \square, \texttt{R} :!, , \texttt{B} : \star, \texttt{NP} :!, \texttt{NT} :!, \texttt{B} :! \vdash c \rightarrow \texttt{P} :!, \texttt{T} :!, \texttt{R} :!, \texttt{B} :!$$

which are both readily checked. Analogously, `MORTGAGE` is well-moded with the moding `mortgage(⋆ × □, □×!, □ × □, □ × □)`, which is intended for covering the third query in the introduction.

We recall that the operational semantics of CLP consists of a transition system from states to states (see Appendix A.9 for details). A state is a pair $\langle Q \| c\rangle$ where Q is a query and c is a constraint, called the constraint store. Initial states are of the form $\langle Q \| \texttt{true}\rangle$. An answer constraint is the constraint store of a final state (if any) with an empty query. The definition of well-moding extends to states.

Definition 4.4. *A state* $\langle \leftarrow c, p_1(\mathbf{x}_1 : \boldsymbol{\tau}_1 \times \boldsymbol{\mu}_1), \ldots, p_n(\mathbf{x}_n : \boldsymbol{\tau}_n \times \boldsymbol{\mu}_n) \|\ c'\rangle$*, with* $n \geq 0$*, is well-moded if for* $i = 1..n$ *the type assertion:*

$$\mathbf{x}_1 : \boldsymbol{\mu}_1, \ldots, \mathbf{x}_{i-1} : \boldsymbol{\mu}_{i-1} \vdash (c \wedge c') \rightarrow \mathbf{x}_i : \boldsymbol{\tau}_i$$

is valid. A query Q *is well-moded if the state* $\langle Q \|\ \texttt{true}\rangle$ *is well-moded.*

4.2 Well-moding for Static Analysis

Widely studied properties of well-moding in logic programming include persistency along derivations, call pattern characterization and computed answer characterization. They are at the basis of several methods for program analysis, including termination [Etalle et al. 1999], transformation and optimization [Somogyi et al. 1996] techniques. The next result shows that the mentioned properties hold for the proposed extension of well-moding to CLP($\mathcal{R}$). By a left-derivation we mean a derivation via the leftmost selection rule.

Theorem 4.5. *Let* P *be a well-moded CLP($\mathcal{R}$) program and* $Q = \ \leftarrow c, p_1(\mathbf{x}_1 : \boldsymbol{\tau}_1 \times \boldsymbol{\mu}_1), \ldots, p_n(\mathbf{x}_n : \boldsymbol{\tau}_n \times \boldsymbol{\mu}_n)$ *a well-moded query. We have that:*

—*Every state selected in a left-derivation of* P *and* Q *is well-moded* (persistency).

—*For every state of the form* $\langle \leftarrow q(\mathbf{x} : \boldsymbol{\tau} \times \boldsymbol{\mu}), R \|\ c'\rangle$ *selected in a left-derivation of* P *and* Q, $\vdash c' \rightarrow \mathbf{x} : \boldsymbol{\tau}$ *is valid* (call patterns).

—*For every* c' *answer constraint of* P *and* Q, $\vdash c' \rightarrow \mathbf{x}_1 : \boldsymbol{\mu}_1, \ldots, \mathbf{x}_n : \boldsymbol{\mu}_n$ *is valid* (answers).

The next two examples provide hints on the kind of analyses that well-moding allows for by exploiting the properties above.

Example 4.6. Consider the two queries from the introduction ← mortgage(100, 5, 20, B) and ← 3 <= T, T <= 5, mortgage(100, T, 20, B). They are well-moded with the moding mortgage($!\times!$, $\square\times!$, $!\times!$, $\star\times!$). Since the MORTGAGE program has been shown to be well-moded in Example 4.3, by Theorem 4.5 we conclude definiteness of balance in every answer constraint store.

The third query from the introduction ← 0 <= B, B <= 10, 15 <= R, R <= 20, mortgage(P, 5, R, B) is well-moded with the moding mortgage($\star\times\square$, $\square\times!$, $\square\times\square$, $\square\times\square$). Again, by Theorem 4.5 we conclude boundedness of principal in every answer constraint store.

Example 4.7. The full version of the MORTGAGE program takes the interest rate as a further predicate argument.

```
(n1)  mortgage(P,T,I,R,B) ←         (n2)  mortgage(P,T,I,R,B) ←
          T = 0,                                T >= 1,
          B = P.                                NP = P + P * I - R,
                                                NT = T - 1,
                                                mortgage(NP,NT,I,R,B).
```

However, this leads to a non-linear constraint appearing in clause (n2). How can we reason on it? Our framework is heavily based on linearity. Also, many constraint solvers are incomplete with respect to non-linear constraints and delay their evaluation until they become linear: even an approach explicitly dealing with non-linear constraints fails with such constraint solvers. We exploit the call pattern characterization property of well-moding by factoring out the P * I term.

```
(n2′) mortgage(P,T,I,R,B) ←          (mu)  mult(P,I,M) ←
          T >= 1,                                P * I = M.
          NP = P + M - R,
          NT = T - 1,
          mult(P, I, M),
          mortgage(NP,NT,I,R,B).
```

Consider now as if the predicate mult is a built-in of the system, and the input-output properties of Theorem 4.5 are guaranteed for the mode mult($!\times!$, $!\times!$, $\star\times!$). The rest of the program, namely clauses (n1) and (n2′), is readily checked to be well-moded with moding mortgage($!\times!$, $\square\times!$, $!\times!$, $!\times!$, $\star\times!$). Therefore, for every call to mult the first and the second arguments are definite, and then the non-linear constraint P * I = M becomes linear at run-time. Finally, notice that we can fold back the mult predicate in (n2′) to conclude that the non-linear constraint in (n2) becomes linear at run-time.

Notice, however, that the approach requires an appropriate choice of the mode of mult and of the position of the call to mult. For instance, consider the mode mortgage($\star\times\square$, $\square\times!$, $!\times!$, $\square\times\square$, $\square\times\square$), where we intend to calculate the principal given the other arguments. The non-linear constraint NP = P + P*I - R in (n2) can be rewritten as P*(I+1) = NP + R. This suggests that mult(P, I+1, NP+R) can be called to compute P from I+1 and NP+R. There are two issues, however. First, the mode of mult must be mult($\star\times\square$, $!\times!$, $\square\times\square$), since the input arguments are now the second and the third ones. Second, the argument NP+R has type $\square$ only *after* the recursive call. As a consequence, the call to mult must occur after the recursive call to mortgage. Summarizing, we have the following version of the (n2) clause:

```
(n2″) mortgage(P,T,I,R,B) ←
        T >= 1,
        I1 = I + 1,
        M = NP + R,
        NT = T - 1,
        mortgage(NP,NT,I,R,B),
        mult(P, I1, M).
```

With the provided modes, the non-linear constraint in mult becomes linear at run-time. It is worth noting that the position of the call to mult in (n2″) can be interpreted as the point in the left-to-right flow of the calls where the non-linear constraint in (n2) becomes linear. Finally, notice that the approach of rearranging body atoms to fit the declared modes is a well-studied technique, adopted for instance in the Mercury programming language [Somogyi et al. 1996].

4.3 Extension to CLP($\mathcal{R}$) with Terms

Several existing CLP($\mathcal{R}$) languages and systems actually adopt a multisorted constraint domain, usually including at least the domain *Term* of pure logic programming terms. In those languages, terms are trees whose leaves are either symbolic constants from *Term* or linear expressions over $\mathcal{R}$. Definitions and results for multisorted domain extends naturally from the single-sorted ones[4]. More specifically, an atom is now of the form $p(t_1, \dots, t_n)$ where p is a predicate of arity n and $t_1, \dots, t_n$ are terms.

Example 4.8. The following LISTSUM program calculates the sum of the elements in a list.

```
(s1)  listsum([], S) ←         (s2)  listsum([X|Xs], S) ←
        S = 0.                          S = X+S1 ,
                                        listsum(Xs, S1).
```

Intuitive modes for listsum include listsum($\square \times \square$, $\star \times \square$) and listsum($! \times !$, $\star \times !$). The former states that the sum is upper and lower bounded if every element of the list is upper and lower bounded. The latter states that the sum is definite if every element in the list is definite.

The definition of well-moding readily extends to programs with terms, provided that the notion of type declaration is lifted to type terms. Let us denote by $vars(t)$ the set of variables of a term t. The following definition extends atd's (and a fortiori, type assertions) to terms.

Definition 4.9. *Let t be a term and $vars(t) = \{x_1, \dots, x_n\}$. We write $t : \tau$ as a shorthand for $x_1 : \tau, \dots, x_n : \tau$.*

[4]Predicates occurring in two or more sorts need to be renamed apart or to be syntactically distinguished. As an example, the unification predicate = (defined by the clause X = X. which is implicitly part of any program with terms) is used also as the linear equality predicate. Other examples are concerned with Prolog arithmetic built-in's. In order to distinguish the two sorts, concrete programming languages, such as SWI-Prolog, write constraints between curly brackets.

The definition is conservative: when t is a variable, the shorthand reduces to an atd. The semantics underlying the definition is that a term is typed as τ if every variable over the reals in it is typed as τ, and every variable over terms in it is typed as τ.

Example 4.10. The `LISTSUM` program is well-moded w.r.t. `listsum(!×!, ⋆×!)`. For instance, for clause `(s2)`, the type assertions:

$$\mathtt{X} :!, \mathtt{Xs} :!, \mathtt{S1} : \star \vdash \mathtt{S = X + S1} \rightarrow \mathtt{Xs} :! \qquad \mathtt{X} :!, \mathtt{Xs} :!, \mathtt{S1} :! \vdash \mathtt{S = X + S1} \rightarrow \mathtt{S} :!$$

are readily checked to be valid.

Using this notation, it is readily checked that the proposed notion is a conservative extension of well-moding for pure logic programs [Apt 1997]. In fact, consider a clause with constraint `true`:

$$p_0(\mathbf{t}_0 : \boldsymbol{\tau}_0 \times \boldsymbol{\mu}_0) \leftarrow p_1(\mathbf{t}_1 : \boldsymbol{\tau}_1 \times \boldsymbol{\mu}_1), \ldots, p_n(\mathbf{t}_n : \boldsymbol{\tau}_n \times \boldsymbol{\mu}_n).$$

By Lemma 2.11, the type assertions to be checked by Definition 4.2 are valid iff:

$$\text{for } i = 1..n, \quad \mathit{nf}(\mathbf{t}_0 : \boldsymbol{\tau}_0, \mathbf{t}_1 : \boldsymbol{\mu}_1, \ldots, \mathbf{t}_{i-1} : \boldsymbol{\mu}_{i-1}) \geq_t \mathit{nf}(\mathbf{t}_i : \boldsymbol{\tau}_i),$$
$$\mathit{nf}(\mathbf{t}_0 : \boldsymbol{\tau}_0, \mathbf{t}_1 : \boldsymbol{\mu}_1, \ldots, \mathbf{t}_n : \boldsymbol{\mu}_n) \geq_t \mathit{nf}(\mathbf{t}_0 : \boldsymbol{\mu}_0).$$

Assume now that only pairs $!\times!$ (input) and $\star\times!$ (output) are used. By expanding the definition of typed terms, the conditions can be rewritten as follows:

$$\text{for } i = 1..n, \quad \cup_{j=1}^{i-1} \mathit{vars}(\mathit{outp}(\mathbf{t}_j)) \cup \mathit{vars}(\mathit{inp}(\mathbf{t}_0)) \supseteq \mathit{vars}(\mathit{inp}(\mathbf{t}_i))),$$
$$\cup_{j=1}^{i-1} \mathit{vars}(\mathit{outp}(\mathbf{t}_j)) \cup \mathit{vars}(\mathit{inp}(\mathbf{t}_0)) \supseteq \mathit{vars}(\mathit{outp}(\mathbf{t}_0)).$$

where, for $i = 0..n$, $\mathit{inp}(\mathbf{t}_i)$ is the subset of $\mathbf{t}_i$ typed as $!\times!$ in $p_i(\mathbf{t}_i : \boldsymbol{\tau}_i \times \boldsymbol{\mu}_i)$, and $\mathit{outp}(\mathbf{t}_i)$ is the subset typed as $\star\times!$. This is exactly the formulation of well-moding for pure logic programs.

Theorem 4.5 readily extends to programs and queries with terms by the following fact, which states that validity of type assertions is monotonic w.r.t. term instantiation. For a term substitution θ and an atd $d = t : \tau$, we write $d\theta$ to denote $t\theta : \tau$. This naturally extends to type declarations.

Lemma 4.11. *Assume that* $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ *is valid, and let* θ *be a term substitution such that* $c\theta = c$. *Then* $\mathbf{d}_1\theta \vdash c \rightarrow \mathbf{d}_2\theta$ *is valid.*

Intuitively, this lemma allows for concluding that an instance of a well-moded clause is well-moded, provided that variables (over the reals) in the clause constraint are not instantiated. A run-time error is typically raised if a variable over the reals is instantiated to a term.

4.4 Extension to Parametric Types

In this section, the definition of well-moding is extended to admit parametric types. The resulting moding system allows for more expressive analyses.

Definition 4.12 (Parametric moding). *A* parametric mode *for an n-ary predicate p is a function d_p from $\{1, \ldots, n\}$ to $\mathcal{PT} \times \mathcal{PT}$. We extend the notation used for modes to parametric modes.*

We assume that, for a parametric mode $p(\boldsymbol{\tau} \times \boldsymbol{\mu})$, if s is a type variable in $\boldsymbol{\mu}$ then s is a type variable in $\boldsymbol{\tau}$, i.e., no "local to output" type variable exist.

Let us first discuss the case of local to output type variables.

Example 4.13. Consider the simple `DOUBLE` program:

```
double(X, Y) ← Y = 2 * X.
```

The mode `double(`$\Box_b \times \Box_b$`,` $\star \times \Box_{2b}$`)` is intended to model the first argument as input with a variability range of b, and the second argument as output with a variability range of $2b$. Dually, the mode `double(`$\star \times \Box_{b/2}$`,` $\Box_b \times \Box_b$`)` swaps the input-output roles of the two arguments. What about the mode `double(`$\Box_b \times \Box_s$`,` $\star \times \Box_{2s}$`)` containing local to output type variables?

First, this mode is somehow weaker than `double(`$\Box_b \times \Box_b$`,` $\star \times \Box_{2b}$`)`, which explicitly states the relations between input-output variability ranges. Thus, it would be more informative to use the stronger mode.

Second, `double(`$\Box_b \times \Box_s$`,` $\star \times \Box_{2s}$`)` states that given the first argument as input, the second argument is output and, after resolution, its variability range is twice the one of the first argument. Using type assertions, this amounts at stating that $x : \Box_b \vdash y = 2x \rightarrow x : \Box_s, y : \Box_{2s}$ is valid. This highlights how the local to output type variable s turns out into a local type variable in a parametric type assertion. Since we aim at a compositional notion of well-moding, and recalling the compositionality limitations highlighted in Section 3.6, we prevent local to output type variables in Definition 4.12.

In the following, we omit the adjective *parametric* when it is clear from the context. Let us consider now an example adopting a direct extension of the notion of well-moding in presence of parametric types.

Example 4.14. Consider the program `ACKERMANN` for computing the Ackermann function.

```
(a1)  ack(M, N, R) ←                (a3)  ack(M, N, R) ←
          M = 0, R = N+1.                     M >= 1, N >= 1,
(a2)  ack(M, N, R) ←                          N1 = N-1, M1 = M - 1,
          M >= 1, N = 0,                      ack(M, N1, T),
          N1 = 1, M1 = M-1,                   ack(M1, T, R).
          ack(M1, N1, R).
```

The mode `ack(`$\star\times!$`,` $\Box_b \times \Box_b$`,` $\star \times \Box_b$`)` states that if `ack(M, N, R)` is called with a variability range of b for `N` then, after it is completely resolved, `M` is definite and `R` has a variability range of b. Consider clause `(a2)`, and let c be `M >= 1, N >= 1, N1 = N-1, M1 = M - 1`. In order to show that the program is well-moded using Definition 4.2, the following parametric type assertions must be valid: $\texttt{M} : \star, \texttt{N} : \Box_b, \texttt{R} : \star \vdash c \rightarrow \texttt{M1} : \star, \texttt{N1} : \Box_b, \texttt{R} : \star$; and $\texttt{M} : \star, \texttt{N} : \Box_b, \texttt{R} : \star, \texttt{M1} :!, \texttt{N1} : \Box_b, \texttt{R} : \Box_b \vdash c \rightarrow \texttt{M} :!, \texttt{N} : \Box_b, \texttt{R} : \Box_b$. They are both readily checked.

The query ← `0 <= M, M <= 5, 1 <= N, N <= 2, ack(M, N, R)` is well-moded, with $b = 1$. In fact, $\vdash$ `0 <= M, M <= 5, 1 <= N, N <= 2` $\rightarrow \texttt{N} : \Box_1$ holds. For every constraint store c in a final state, the type assertion $\vdash c \rightarrow \texttt{R} : \Box_b$, with $b = 1$, is then valid. Stated in other words, if we start with a value for `N` known with an approximation of at most ± 0.5, in each computed answer the result `R` has the same approximation. Notice that this does not mean that two results `R` from two different computed answers differ by at most ± 0.5.

As outlined at the end of the example, however, when calling a predicate with a parametric mode it may be necessary to instantiate type variables, e.g., $b = 1$ in the example query. In the general case, this requires an extended formulation of Definition 4.2.

Example 4.15. Consider the program ACKERMANN with the additional clause:

```
(a4)  ack2(N, R) ←  ack(N, N, R).
```

and mode $\texttt{ack2(}\Box_a \times \Box_a\texttt{, } \star \times \Box_a\texttt{)}$. Is (a4) well-moded? Using Definition 4.2, we should show validity of the type assertions $\texttt{N} : \Box_a, \texttt{R} : \star \vdash \texttt{true} \rightarrow \texttt{N} : \Box_b, \texttt{R} : \star$ and $\texttt{N} : \Box_a, \texttt{R} : \star, \texttt{N} : \Box_b, \texttt{R} : \Box_b \vdash \texttt{true} \rightarrow \texttt{N} : \Box_a, \texttt{R} : \Box_a$, which is not the case since b and a are distinct type variables. Intuitively, however, b has a meta-level usage, in the sense that when calling the predicate ack the type variable b can be instantiated to satisfy the call context. By instantiating $b = a$, the type assertions above become $\texttt{N} : \Box_a, \texttt{R} : \star \vdash \texttt{true} \rightarrow \texttt{N} : \Box_a, \texttt{R} : \star$ and $\texttt{N} : \Box_a, \texttt{R} : \star, \texttt{N} : \Box_a, \texttt{R} : \Box_a \vdash \texttt{true} \rightarrow \texttt{N} : \Box_a, \texttt{R} : \Box_a$, which are trivially valid. Similarly , consider:

```
(a5)  ack3(M, R) ←  N=1, ack(M, N, R).
```

with mode $\texttt{ack3(}\star\times !\texttt{, } \star\times !\texttt{)}$. The type assertions to be shown are $\texttt{M} : \star, \texttt{R} : \star \vdash \texttt{N=1} \rightarrow \texttt{M} : \star, \texttt{N} : \Box_b, \texttt{R} : \star$ and $\texttt{N} : \Box_b, \texttt{R} : \Box_b, \texttt{M} : \Box_b \vdash \texttt{N=1} \rightarrow \texttt{R} :!, \texttt{M} :!$, where the former is valid and the latter is not. By instantiating $b = 0$, the type assertions become $\texttt{M} : \star, \texttt{R} : \star \vdash \texttt{N=1} \rightarrow \texttt{M} : \star, \texttt{N} :!, \texttt{R} : \star$ and $\texttt{N} :!, \texttt{R} :!, \texttt{M} :! \vdash \texttt{N=1} \rightarrow \texttt{R} :!, \texttt{M} :!$, which are both valid.

Let us formalize type variable instantiation by means of type substitutions.

Definition 4.16. *A type substitution ϑ is a function mapping type variables into linear expressions over type variables.*

Type substitutions readily lift to parametric types, by setting $\vartheta(\Box_e) = \Box_{\vartheta(e)}$ and $\vartheta(\tau) = \tau$ for $\tau \in \mathcal{BT}_\Box$. We are now in the position to (conservatively) extend the notion of well-moding.

Definition 4.17 (Well-moding with parametric types). *A CLP($\mathcal{R}$) clause with parametric modes:*

$$p_0(\mathbf{x}_0 : \boldsymbol{\tau}_0 \times \boldsymbol{\mu}_0) \leftarrow c, p_1(\mathbf{x}_1 : \boldsymbol{\tau}_1 \times \boldsymbol{\mu}_1), \ldots, p_n(\mathbf{x}_n : \boldsymbol{\tau}_n \times \boldsymbol{\mu}_n)$$

is well-moded if there exist type substitutions $\vartheta_1, \ldots, \vartheta_n$ such that the following type assertions have no local type variable and are valid:

$$\text{for } i = 1..n, \quad \mathbf{x}_0 : \boldsymbol{\tau}_0, \mathbf{x}_1 : \vartheta_1(\boldsymbol{\mu}_1), \ldots, \mathbf{x}_{i-1} : \vartheta_{i-1}(\boldsymbol{\mu}_{i-1}) \vdash c \rightarrow \mathbf{x}_i : \vartheta_i(\boldsymbol{\tau}_i),$$
$$\mathbf{x}_0 : \boldsymbol{\tau}_0, \mathbf{x}_1 : \vartheta_1(\boldsymbol{\mu}_1), \ldots, \mathbf{x}_n : \vartheta_n(\boldsymbol{\mu}_n) \vdash c \rightarrow \mathbf{x}_0 : \boldsymbol{\mu}_0.$$

A CLP($\mathcal{R}$) program P is well-moded if every clause in it is well-moded.

The requirement that there is no local type variable formalizes the intuitions of Example 4.15 that type variables in the modes of body atoms must be instances of type variables occurring in the mode of the head of the clause.

Similarly, well-moding for a state (and then for a query) $\langle \leftarrow c, p_1(\mathbf{x}_1 : \boldsymbol{\tau}_1 \times \boldsymbol{\mu}_1), \ldots, p_n(\mathbf{x}_n : \boldsymbol{\tau}_n \times \boldsymbol{\mu}_n) \| \, c' \rangle$ is extended by assuming type substitutions $\vartheta_1, \ldots, \vartheta_n$ such that the following type assertions have no local type variable and are valid:

$$\text{for } i = 1..n, \quad \mathbf{x}_1 : \vartheta_1(\boldsymbol{\mu}_1), \ldots, \mathbf{x}_{i-1} : \vartheta_{i-1}(\boldsymbol{\mu}_{i-1}) \vdash c \wedge c' \rightarrow \mathbf{x}_i : \vartheta_i(\boldsymbol{\tau}_i).$$

The basic property of persistency holds for the revised notions of well-moded programs and queries. Call pattern and computed answer characterizations readily follow from persistency.

THEOREM 4.18 (PERSISTENCY). *Let P be a CLP($\mathcal{R}$) program and Q a query, both well-moded with parametric modes. Every state selected in a left-derivation of P and Q is well-moded.*

The natural question with the revised definition of well-moding is: how to derive the type substitutions ϑ? In some sense, a form of type inference is needed for checking well-moding.

For an atomic query $\leftarrow c, p_1(\mathbf{x}_1 : \boldsymbol{\tau} \times \boldsymbol{\mu})$ to be well-moded, we have that $\vartheta_1(\boldsymbol{\tau})$ must have no type variable, namely all types are in $\mathcal{BT}$. Thus, the POLYINFER procedure can be adopted to infer ϑ_1. Since we assume no local to output type variables, $\vartheta_1(\boldsymbol{\mu})$ is also in $\mathcal{BT}$. As a consequence, the approach can be iterated for non-atomic queries $\leftarrow c, p_1(\mathbf{x}_1 : \boldsymbol{\tau} \times \boldsymbol{\mu}), \ldots, p(\mathbf{x}_n : \boldsymbol{\tau} \times \boldsymbol{\mu})$ for $n = 2 \ldots n$.

For a single program clause , however, we cannot resort to an inference procedure for $\mathcal{BT}$, since type assertions are in the $\mathcal{PT}$ type system. As discussed in Section 3.6, a complete inference procedure does not exist if we restrict to (non-piecewise) linear expressions. Therefore, either the type substitutions ϑ are provided by the user (as a further input in addition to modes) in a computer-assisted proof, or an inference procedure in an extended syntax must be devised.

Finally, for a program, we have simply to show the proof obligations of each program clause in isolation, since the type substitutions ϑ are local to each clause.

5. EXPERIMENTAL RESULTS

5.1 The `clpt` system

Proof obligations of well-moding consist of repeatedly calling a decision procedure for checking validity of type assertions syntactically built from the clauses of the program under consideration. Checking well-moding is then a representative test-bed for testing the efficiency in practice of the procedures designed in the paper.

We have implemented, in standard C++, the checking procedure for well-moding, including the extension to CLP($\mathcal{R}$) with terms, and the following decision and inference procedures: CHECK(χ), LPINFER, IEINFER, POLYINFER, PARCHECK. Validity of a type assertion $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is checked on the basis of the type system $\mathbf{d}_1$ and $\mathbf{d}_2$ are defined in: by CHECK(IEINFER) for $\mathcal{BT}_!$; by CHECK(POLYINFER) for $\mathcal{BT}$; and by the PARCHECK procedure for $\mathcal{PT}$. For non-valid type assertions, the POLYINFER procedure is called to provide the most general type that can be inferred for variables in $\mathbf{d}_2$. The implementation relies on the `polylib` library [Loechner 2010] for the calculation of the Minkowski's form of (parameterized) polyhedra. The overall system, called `clpt`, is available as open source from `http://www.di.unipi.it/~ruggieri/software`.

5.2 Testbeds and Experiments

Tables I-IV report the execution times over several testbeds of programs. Tests were run on a PC Xeon 2.8GHz with Linux 2.6.17. In brief, those results provide us with confidence on the efficiency of the proposed approach in practice. More in depth, let us consider the various testbeds.

program	C	A	mode of the main predicates	time
ack	3	6	ack($\star \times \Box$, $\Box \times \Box$, $\star \times \Box$)	0.0008
ack	3	6	ack($\star \times !$, $! \times !$, $\star \times !$)	0.0010
ack	3	6	ack($\star \times !$, $\Box_2 \times \Box_2$, $\star \times \Box_2$)	0.0025
ack	3	6	ack($\star \times !$, $\Box_b \times \Box_b$, $\star \times \Box_b$)	0.0030
fib	2	4	fib($! \times !, \star \times !$)	0.0007
fib	2	4	fib($\star \times \Box_1, \star \times !$)	0.0019
mc91	2	4	mc($\sqcap \times \Box, \star \times \Box$)	0.0005
mc91	2	4	mc($! \times !, \star \times !$)	0.0005
mortgage	2	3	mortgage($\star \times \Box, \sqcap \times !, \Box \times \Box, \Box \times \Box$)	0.0007
mortgage	2	3	mortgage($! \times !, \sqcap \times !, ! \times !, \star \times !$)	0.0007
mortgage	2	4	mortgage($! \times !, \sqcap \times !, ! \times !, ! \times !, \star \times !$) mult($! \times !, ! \times !, \star \times !$)	0.0009
schedule	10	21	schedule($\Box \times \Box, \star \times \sqcap, \star \times \sqcup$) into($\Box \times \Box, \Box \times \Box, \star \times \sqcap, \star \times \sqcup$)	0.0020
schedule	10	21	schedule($\Box_b \times \Box_b, \star \times \sqcap, \star \times \sqcup$) into($\Box_b \times \Box_b, \Box_b \times \Box_b, \star \times \sqcap, \star \times \sqcup$)	0.0080
send+more =money	4	9	solve($\star \times \Box, \star \times \Box, \star \times \Box, \star \times \Box, \star \times \Box, \star \times \Box,$ $\star \times \Box, \star \times \Box$)	0.0633
send+more =money	4	9	solve($\star \times \Box_1, \star \times \Box_9, \star \times \Box_9, \star \times \Box_9, \star \times !, \star \times \Box_9,$ $\star \times \Box_9, \star \times \Box_9$)	0.4846
send+more =money	4	9	solve($\Box_b \times \Box_b, \star \times \Box_9, \star \times \Box_9, \star \times \Box_9, \star \times !, \star \times \Box_9,$ $\star \times \Box_9, \star \times \Box_9$)	1.1526
tak	3	8	tak($! \times !, ! \times !, ! \times !, \star \times !$)	0.0010

Table I. The **folk** testbed. Elapsed time in seconds, C = no. of clauses, A = no. of atoms.

program	C	A	mode of the main predicates	time
listsum	2	3	listsum($\Box \times \Box$, $\star \times \Box$)	0.0005
listsumpos	2	3	listsumpos($\star \times \Box$, $\sqcap \times \Box$)	0.0005
nqueens	13	25	nqueens($! \times !$, $\star \times !$)	0.0016
plate	6	11	gp192($\star \times \sqcup$)	0.0024
quicksort	7	14	quicksort($! \times !$, $\star \times !$)	0.0010
quicksort	7	14	quicksort($\Box_1 \times \Box_1$, $\star \times \Box_1$)	0.0015
quicksort	7	14	quicksort($\Box_b \times \Box_b$, $\star \times \Box_b$)	0.0017
sequence	15	32	sequence_problem($\star \times !$)	0.0017
tree_layout	35	82	gp207($\star \times !$, $\star \times \sqcup$, $\star \times \star$)	0.0053

Table II. The **term** testbed. Elapsed time in seconds, C = no. of clauses, A = no. of atoms.

The **folk** *testbed.* This testbed consists of small-size programs with linear constraints only, from the CLP($\mathcal{R}$) folklore. For a same program, we considered different modes of usage, as done for the MORTGAGE example throughout the paper. Also, we consider different type systems for specifying modes. For instance, the ack program is tested for modes whose types belong to $\mathcal{BT}_\Box$, $\mathcal{BT}_!$, $\mathcal{BT}$, and $\mathcal{PT}$. Notice how the execution times in Table I increase with the expressiveness of the type system and, a fortiori, with the complexity of the checking procedure adopted. Running times are very low on average. The send+more=money program is the most demanding one, due to the presence of a worst-case constraint in it (see later on the **worst** testbed).

The **term** *testbed.* It includes programs mixing linear constraints and terms, mainly from CLP textbooks [Marriott and Stuckey 1998]. As a special case, when

program	**C**	**A**	**mode of the main predicates**	**time**
assoc	13	23	assoc_to_list(!×!, ⋆×⊔)	0.0039
credit	33	63	credit(!×!, ⋆×!)	0.0094
sequence	5	18	q(⋆×!)	0.0041
ttt	36	75	play(⋆×!)	0.0136
Ackermann	7	14	all arguments moded as !×!	0.0015
CaffeineMark	4944	9888	all arguments moded as !×!	3.1226
JLex	850	1700	all arguments moded as !×!	0.5400
Kitten	2320	4640	all arguments moded as !×!	0.6115
NQueens	290	580	all arguments moded as !×!	0.0970
RayTracer	75	150	all arguments moded as !×!	0.0300

Table III. The **mlsize** testbed. Elapsed time in seconds, C = no. of clauses, A = no. of atoms.

program	**C**	**A**	**mode of the main predicates**	**time**
worst_10	1	1	worst(⋆×□)	0.07
worst_11	1	1	worst(⋆×□)	0.32
worst_12	1	1	worst(⋆×□)	1.23
worst_13	1	1	worst(⋆×□)	7.00
worst_14	1	1	worst(⋆×□)	33.50
worst_15	1	1	worst(⋆×□)	135.52

Table IV. The **worst** testbed. Elapsed time in seconds, C = no. of clauses, A = no. of atoms.

clauses contain no constraint, pure Prolog programs belong to this testbed. Table II shows very low execution times, better than in the **folk** testbed. In fact, we observe that a variable x representing a term (e.g., a list) does not typically occur in a clause constraint, hence proving validity of a type assertion $\mathbf{d} \vdash c \rightarrow x : \tau$ reduces to checking $\mathbf{d} \vdash \texttt{true} \rightarrow x : \tau$ which, by Lemma 2.11, reduces to checking whether $\tau = \star$, or $x : \tau$ belongs to $nf(\mathbf{d})$.

The **mlsize** *testbed.* This set of medium-to-large size programs is automatically generated from Prolog and Java programs. Prolog programs are transformed into CLP($\mathcal{R}$) programs by applying the term-size or the list-size norm to predicate arguments, a basic abstract interpretation technique used by some termination analysers [Lagoon et al. 2003; Mesnard and Ruggieri 2003]. Java bytecode programs are transformed into CLP($\mathcal{R}$) programs by applying the Julia+BinTerm system [Spoto et al. 2010]. This analyzer combines information from sharing, cyclicity, and path-length analysis to generate binary CLP programs, the termination of which ensures termination of the original Java programs.

The execution times reported in Table III show that the clpt system scales up to large-size programs. This is theoretically justified by noting that proof obligations of well-moding consider each clause in isolation.

The **worst** *testbed.* This testbed consists of programs worst_n with only one rule of the form $worst(y) \leftarrow cw_n, y = 1$, where cw_n is the worst-case constraint from Example 3.54 yielding an exponential number of vertices in n, even for non-parameterized polyhedra. Such an exponential grow is reflected by the execution times shown in Table IV. Notice that, due to the mode worst(⋆×□), the clpt system actually adopts the Check(LPInfer) procedure for programs in this testbed.

				time	
program	**C**	**A**	**modes**	**T2**	**ratio**
`ack`	3	6	`ack(`$\star \times \square$, $\square \times \square$, $\star \times \square$`)`	0.0049	6.33
`listsum`	2	3	`listsum(`$\square \times \square$, $\star \times \square$`)`	0.0027	6.13
`mc91`	2	4	`mc(`$\sqcap \times \square, \star \times \square$`)`	0.0028	5.18
`nqueens`	13	25	`nqueens(` $\square \times \square$, $\star \times \square$`)`	0.0089	5.94
`schedule`	10	21	`schedule(`$\square \times \square, \star \times \sqcap, \star \times \sqcup$`)` `into(`$\square \times \square, \square \times \square, \star \times \sqcap, \star \times \sqcup$`)`	0.0139	6.95
`send+more =money`	4	9	`solve(`$\star \times \square, \star \times \square, \star \times \square, \star \times \square,$ $\star \times \square, \star \times \square, \star \times \square, \star \times \square$`)`	0.0102	0.16
`tree_layout`	35	82	`gp207(` $\star \times \square$, $\star \times \sqcup$, $\star \times \star$`)`	0.0305	5.75
`worst_13`	1	1	`worst(`$\star \times \square$`)`	0.0020	0.00028
`worst_14`	1	1	`worst(`$\star \times \square$`)`	0.0021	0.00007
`Ackermann`	7	14	all arguments moded as $\square \times \square$	0.0115	7.67
`CaffeineMark`	4944	9888	all arguments moded as $\square \times \square$	19.013	6.45
`JLex`	850	1700	all arguments moded as $\square \times \square$	3.088	6.01
`Kitten`	2320	4640	all arguments moded as $\square \times \square$	4.241	7.30
`NQueens`	290	580	all arguments moded as $\square \times \square$	0.632	6.87
`RayTracer`	75	150	all arguments moded as $\square \times \square$	0.172	6.02

Table V. Elapsed times for the CHECK(LP2INFER) procedure, and ratio over CHECK(LPINFER). Elapsed time in seconds, C = no. of clauses, A = no. of atoms. T2 = time of CHECK(LP2INFER), ratio = T2 /(time of CHECK(LPINFER)).

5.3 Polyhedra-based vs Simplex-based Implementations

We are interested in comparing performances of the polyhedra-based implementation of the checking procedures with a Simplex-based one. This can be done for the $\mathcal{BT}_\square$ type system by comparing the CHECK(LPINFER) procedure (using the Minkowski's form of polyhedra) with an implementation using a Simplex-based linear programming solver, which we call CHECK(LP2INFER).

We have implemented the CHECK(LP2INFER) procedure as part of the `clpt` system by relying on the `lpsolve` library [Berkelaar et al. 2010]. Table V reports the execution times for some programs from the previously introduced testbeds moded with types in $\mathcal{BT}_\square$. Apart from the worst-case scenario (we recall that `send+more=money` contains a worst case constraint), on average the polyhedra-based approach is 5 to 7 times faster than a Simplex-based one.

Although the results may be biased by (in)efficiencies of the adopted libraries, we observe that, on average, the size of the constraints appearing in a program clause is quite small, as per number of variables and number of inequalities. Moreover, a same constraint is involved in $n+1$ type assertions, where n is the number of atoms in a clause body. Therefore, an approach computing the set of (parameterized) vertices once and for all the $n+1$ type assertions can reasonably outperform a Simplex-based one requiring a relatively large setup time for each of the $n+1$ type assertions. Finally, a polyhedra-based implementation shares some computations[5] with the CHECK(POLYINFER) and PARCHECK procedures. A polyhedra-based

[5]Since the double description method, adopted to compute the Minkowski's form of (parameterized) polyhedra, is incremental, the CHECK(POLYINFER) and PARCHECK procedures benefit from the availability of the Minkowski's form of $\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c$ computed by CHECK(LPINFER).

implementation of Check(LPInfer) can then speed up those procedures, which do not have a linear programming equivalent.

6. RELATED WORK

A class of formulas, called *parametric queries*, is investigated in [Huynh et al. 1991]. It includes formulas $\exists a \forall \mathbf{v}\ c \rightarrow x \sim a$, where $\sim \in \{\leq, =, \geq\}$, or, with our notation, type assertions of the form $\vdash c \rightarrow x : \tau$ with $\tau \in \mathcal{BT}_\Box$. The approach switches from the problem of checking $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c\} \leq a$ to its dual form $max\{0 \mid \mathbf{y}^T\mathbf{A}_c = \mathbf{c}, a = \mathbf{y}^T\mathbf{b}_c + q, \mathbf{y} \geq \mathbf{0}, q \geq 0\} = 0$, namely on checking feasibility of $\mathbf{y}^T\mathbf{A}_c = \mathbf{c}, a = \mathbf{y}^T\mathbf{b}_c + q, \mathbf{y} \geq \mathbf{0}, q \geq 0$. This requires to solve a distinct linear programming problem for each variable to be typed. By switching to the homogeneous problem, our approach allows for concentrating on maximizing several linear functions on a single polyhedron, for which we need to extract its Minkowski's form only once. More importantly, as soon as general type assertions $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ are considered, switching to the dual form yields a non-linear problem.

The problem of maximizing a linear function over a parameterized system of linear inequalities is addressed by *(multi)parameterized linear programming.* The solution of the problem can be expressed as a piecewise linear function of the parameters [Gal and Nedoma 1972; Gal 1995], or as the maximum of a finite set of linear functions of the parameters [Borrelli et al. 2003; Keerthi and Sridharan 1990; Schechter 1987]. Therefore, an approach alternative to the Check(POLYInfer) procedure would consist of computing (for each variable to be typed) the *max* and *min* functions of a parameterized linear programming problem and then of comparing the resulting piecewise linear functions on each pair of breaks they are defined on. It is worth mentioning that, even for a single parameter, the number of breaks can be exponential [Murty 1983].

Definiteness analysis for CLP($\mathcal{R}$) has been investigated in several works [Baker and Søndegaard 1993; Codish et al. 2001; Garcia de la Banda et al. 1996; Howe and King 2000], and it is used as a basic tool in CLP($\mathcal{R}$) compiler optimizations [Kelly et al. 1998]. Here we have extended the concept from definite values, namely the ! type, to ranges, namely the $\Box_r$ type. The cited papers adopt abstract interpretation techniques to *infer* boolean expressions relating definiteness of predicate arguments. E.g., an inferred $p(x, y) = x \rightarrow y$ states that if x is definite when $p(x, y)$ is called then y is definite when it is completely resolved. Compared to the notion of well-moding, inference does not need modes to be specified. However, the mentioned approaches restrict to consider equality constraints only. Also, it is worth noting that groundness inference for logic programs is shown to be exponential in the worst case [Genaim et al. 2001]. Finally, we include in this stream of research also the work [Hanus 1995] which adopts abstract interpretation to detect non-linear constraints that become linear at run-time.

A survey of *applications of polyhedra* and their Minkowski's form to the analysis and verification of hardware and software systems is reported in [Bagnara et al. 2009]. The definition of the Minkowski's form has been extended in [Bagnara et al. 2005] to explicitly take into account strict inequalities. Finally, we refer the reader to [Bagnara et al. 2008] for an experimental comparison of several libraries, including `polylib`, for reasoning about polyhedra.

7. CONCLUSIONS

We have introduced a type system for linear constraints over the reals that is able to reason about definiteness, upper bounds, lower bounds, range width (or approximatively known values) of variables. The problems of inferring and checking validity of type assertions have been investigated and solved by proposing specialized procedures of increasing complexity and expressiveness. Extensions to generalized constraints and parametric types are also presented.

As an application area, types are used for annotating (moding) CLP($\mathcal{R}$) programs, while type assertions represent the basic tool for extending the notion of well-moding from logic programming to CLP($\mathcal{R}$). The extension is conservative, since it can reason on programs mixing linear constraints and logic terms.

We have implemented in standard C++ all the type assertion checking and inference procedures, and the well-moding checking procedure. The system developed, called `clpt`, is released as open source. We have conducted an experimental evaluation of `clpt` over several testbeds of programs. The results show the efficiency of the proposed approach in practice.

Acknowledgements. We thank Maurice Bruynooghe and Sven Verdoolaege for suggesting the use of implicit equalities in checking $\mathbf{d}|_{!} \vdash c \rightarrow x :!$. We are grateful to Vincent Loechner for his precious support with the `polylib` library.

REFERENCES

Apt, K. R. 1997. *From Logic Programming to Prolog.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Bagnara, R., Hill, P. M., Ricci, E., and Zaffanella, E. 2005. Not necessarily closed convex polyhedra and the double description method. *Formal Aspects Comput. 17,* 2, 222–257.

Bagnara, R., Hill, P. M., and Zaffanella, E. 2008. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Sci. Comput. Program. 72,* 1-2, 3–21.

Bagnara, R., Hill, P. M., and Zaffanella, E. 2009. Applications of polyhedral computations to the analysis and verification of hardware and software systems. *Theor. Comput. Sci. 410,* 46, 4672–4691.

Baker, N. and Søndegaard, H. 1993. Definiteness analysis for CLP($\mathcal{R}$). In *Proceedings of the 16th Australian Computer Science Conference*, G. Gupta, G. Mohay, and R. Topor, Eds. Australian Computer Science Communications, vol. 15. 321–332.

Basu, S., Pollack, R., and Roy, M.-F. 1996. On the combinatorial and algebraic complexity of quantifier elimination. *J. ACM 43,* 6, 1002–1045.

Becket, R., Garcia de la Banda, M., Marriott, K., Somogyi, Z., Stuckey, P. J., and Wallace, M. 2006. Adding constraint solving to mercury. In *Proceedings of the 8th International Symposium on Practical Aspects of Declarative Languages*, P. Van Hentenryck, Ed. Lecture Notes in Computer Science, vol. 3819. Springer, 118–133.

Berkelaar, M., Eikland, K., and Notebaert, P. 2010. `lp_solve`: Open source (mixed-integer) linear programming system. Available at `http://lpsolve.sourceforge.net`, Version 5.5.0.15.

Borgwardt, K. H. 2007. Average-case analysis of the double description method and the beneath-beyond algorithm. *Discrete Comput. Geom. 37,* 2, 175–204.

Borrelli, F., Bemporad, A., and Morari, M. 2003. Geometric algorithm for multiparametric linear programming. *J. Optim. Theory Appl. 118,* 3, 515–540.

Brown, C. W. 2003. QEPCAD B: A program for computing with semi-algebraic sets using CADs. *ACM SIGSAM Bullettin 37,* 4, 97–108.

Chernikova, N. V. 1965. An algorithm for finding the general formula for non-negative solutions of systems of linear inequalities. *U.S.S.R. Comput. Math. Math. Phys. 5*, 228–233.

CODISH, M., GENAIM, S., SØNDEGAARD, H., AND STUCKEY, P. 2001. Higher-precision groundness analysis. In *Proceedings of the 17th International Conference on Logic Programming*, P. Codognet, Ed. Lecture Notes in Computer Science, vol. 2237. Springer, 135–149.

COLLINS, G. E. AND HONG, H. 1991. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symbolic Comput. 12,* 3, 299–328.

DAVENPORT, J. H. AND HEINTZ, J. 1988. Real quantifier elimination is doubly exponential. *J. Symbolic Comput. 5,* 1-2, 29–35.

DOLZMANN, A. AND STURM, T. 1997. REDLOG: Computer algebra meets computer logic. *ACM SIGSAM Bullettin 31,* 2, 2–9.

DOLZMANN, A., STURM, T., AND WEISPFENNING, V. 1998a. A new approach for automatic theorem proving in real geometry. *J. Automat. Reason. 21,* 3, 357–380.

DOLZMANN, A., STURM, T., AND WEISPFENNING, V. 1998b. Real quantifier elimination in practice. In *Algorithmic Algebra and Number Theory*, B. H. Matzat, G.-M. Greuel, and G. Hiss, Eds. Springer, 221–248.

ETALLE, S., BOSSI, A., AND COCCO, N. 1999. Termination of well-moded programs. *J. Logic Programming 38,* 2, 243–257.

GAL, T. 1995. *Postoptimal Analyses, Parametric Programming, and Related Topics*, 2nd ed. de Gruyter and Co., Berlin, Germany.

GAL, T. AND NEDOMA, J. 1972. Multiparametric linear programming. *Management Sci. 18*, 406–422.

GARCIA DE LA BANDA, M., HERMENEGILDO, M., BRUYNOOGHE, M., DUMORTIER, V., JANSSENS, G., AND SIMOENS, W. 1996. Global analysis of constraint logic programs. *ACM Trans. Program. Lang. Syst. 18,* 5, 564–614.

GENAIM, S., CODISH, M., AND HOWE, J. M. 2001. Worst-case groundness analysis using definite boolean functions. *Theory Pract. Logic Program. 1,* 5, 611–615.

GOLDMAN, A. J. 1956. Resolution and separation theorems for polyhedral convex sets. In *Linear Inequalities and Related Systems*, H. W. Kuhn and A. W. Tucker, Eds. Princeton University Press, Princeton, NJ, USA.

GREENBERG, H. J. 1996. Consistency, redundancy, and implied equalities in linear systems. *Ann. Math. Artificial Intelligence 17,* 1-2, 37–83.

HANUS, M. 1995. Compile-time analysis of nonlinear constraints in CLP($\mathcal{R}$). *New Gen. Comput. 13,* 2, 155–186.

HOLZBAUR, C. 1995. OFAI clp(q,r) manual, edition 1.3.3. Tech. Rep. TR-95-09, Austrian Research Institute.

HOWE, J. M. AND KING, A. 2000. Abstracting numeric constraints with boolean functions. *Inf. Process. Lett. 75,* 1-2, 17–23.

HUYNH, T., JOSKOWICZ, L., LASSEZ, C., AND LASSEZ, J.-L. 1991. Practical tools for reasoning about linear constraints. *Fund. Inform. 15,* 3-4, 357–380.

JAFFAR, J. AND MAHER, M. 1994. Constraint logic programming: A survey. *J. Logic Programming 19,20*, 503–581.

JAFFAR, J., MAHER, M., MARRIOTT, K., AND STUCKEY, P. J. 1998. The semantics of constraint logic programs. *J. Logic Programming 37,* 1-3, 1–46.

JAFFAR, J., MICHAYLOV, S., STUCKEY, P., AND YAP, R. 1992. The CLP($\mathcal{R}$) language and system. *ACM Trans. Program. Lang. Syst. 14,* 3, 339–395.

KEERTHI, S. S. AND SRIDHARAN, K. 1990. Solution of parameterized linear inequalities by Fourier elimination and its applications. *J. Optim. Theory Appl. 65,* 1, 161–169.

KELLY, A. D., MARRIOTT, K., MACDONALD, A., STUCKEY, P. J., AND YAP, R. 1998. Optimizing compilation of CLP($\mathcal{R}$). *ACM Trans. Program. Lang. Syst. 20,* 6, 1223–1250.

KHACHIYAN, L. 1979. A polynomial algorithm in linear programming. *Soviet Math. 20*, 191–194.

LAGOON, V., MESNARD, F., AND STUCKEY, P. J. 2003. Termination analysis with types is more accurate. In *Proceedings of the 19th International Conference on Logic Programming.* Lecture Notes in Computer Science. Springer, 254–268.

Lassez, J.-L. and McAllon, K. 1992. A canonical form for generalized linear constraints. *J. Symbolic Comput. 13,* 1, 1–24.

Le Verge, H. 1992. A note on Chernikova's algorithm. Tech. Rep. 635, IRISA, Campus Universitaire de Beaulieu, Rennes, France.

Loechner, V. 2010. Polylib: a library for manipulating parameterized polyhedra. Available at `http://icps.u-strasbg.fr/polylib`, Version 5.22.5.

Loechner, V. and Wilde, D. K. 1997. Parameterized polyhedra and their vertices. *Int. J. Parallel Prog. 25*, 525–549.

Marriott, K. and Stuckey, P. J. 1998. *Programming with Constraints: An Introduction.* The MIT Press, Cambridge, MA, USA.

Mesnard, F. and Ruggieri, S. 2003. On proving left-termination of constraint logic programs. *ACM Trans. Comput. Logic 4,* 2, 207–259.

Motzkin, T., Raiffa, H., Thompson, G., and Thrall, R. 1953. The double description method. In *Contributions to the theory of games – Volume II*, H. Kuhn and A. Tucker, Eds. Annals of Mathematics Studies, vol. 28. Princeton University Press, Princeton, NJ, USA, 51–73.

Murty, K. G. 1983. *Linear Programming.* John Wiley & Sons, Hoboken, NJ, USA.

Refalo, P. 1998. Approaches to the incremental detection of implicit equalities with the revised simplex method. In *Principles of Declarative Programming*, C. Palamidessi, H. Glaser, and K. Meinke, Eds. Lecture Notes in Computer Science, vol. 1490. Springer, 481–497.

Renegar, J. 1992. On the computational complexity and geometry of the first-order theory of the reals. *J. Symbolic Comput. 13,* 3, 255–352.

Ruggieri, S. and Mesnard, F. 2008. Typing linear constraints for moding CLP($\mathcal{R}$) programs. In *Proceedings of the 15th Static Analysis Symposium*, M. Alpuente and G. Vidal, Eds. Lecture Notes in Computer Science, vol. 5079. Springer, 128–143.

Schechter, M. 1987. Polyhedral functions and multiparametric linear programming. *J. Optim. Theory Appl. 53,* 2, 269–280.

Schrijver, A. 1986. *Theory of Linear and Integer Programming.* John Wiley & Sons, Hoboken, NJ, USA.

Shoenfield, J. 1967. *Mathematical Logic.* Addison-Wesley, Reading, MA, USA.

Somogyi, Z., Henderson, F., and Conway, T. 1996. The execution algorithm of Mercury, an efficient purely declarative logic programming language. *J. Logic Programming 29,* 1–3, 17–64.

Spoto, F., Mesnard, F., and Payet, E. 2010. A termination analyser for Java bytecode based on path-length. *ACM Trans. Program. Lang. Syst. 32,* 3.

Strzebonski, A. 2000. Solving systems of strict polynomial inequalities. *J. Symbolic Comput. 29,* 3, 471–480.

Stuckey, P. J. 1991. Incremental linear constraint solving and implicit equalities. *ORSA J. Comput. 3*, 269–274.

Van Den Vries, L. 1988. Alfred Tarski's elimination theory for closed fields. *J. Symbolic Logic 53,* 1, 7–19.

Wilde, D. K. 1993. A library for doing polyhedral operations. Tech. Rep. PI 785, IRISA, Campus Universitaire de Beaulieu, Rennes, France.

Received November 2008; August 2009; accepted December 2009

THIS DOCUMENT IS THE ONLINE-ONLY APPENDIX TO:

Typing Linear Constraints

SALVATORE RUGGIERI
Dipartimento di Informatica, Università di Pisa, Italy
FRED MESNARD
IREMIA, LIM, Université de la Réunion, France

ACM Transactions on Programming Languages and Systems, Vol. V, No. N, M 20YY, Pages 1–40.

A. PROOFS

A.1 Section 2.1

The proofs in this section follow a proof-theoretic approach based on first order predicate calculus. We denote by $\phi[t/x]$ the substitution of free occurrences of the variable x with the term t within the formula ϕ.

To keep the notation simple, we will exploit the fact that, since the $\upsilon()$ function assumes fresh variables, the quantification $\exists \upsilon(\mathbf{d}_2) \setminus \upsilon(\mathbf{d}_1)$ in (1) from Definition 2.4 is equivalent to $\exists \upsilon(\mathbf{d}_2)$ as far as the type system $\mathcal{BT}$ is considered. When writing instances of (1), $\mathbf{v}$ will denote $vars(c) \cup vars(\mathbf{d}_1) \cup vars(\mathbf{d}_2)$ where c is the constraint and $\mathbf{d}_1, \mathbf{d}_2$ are the type declarations involved.

LEMMA 2.8.

PROOF. Assume that $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is valid, i.e. $\forall \upsilon(\mathbf{d}_1)\ \exists \upsilon(\mathbf{d}_2)\ \forall \mathbf{v}.\ \phi(\mathbf{d}_1) \wedge c \rightarrow \phi(\mathbf{d}_2)$ holds. The result follows from the parts *(a), (b)* and *(c)* below.

(a). Since $\mathbf{d}'_1 \geq_t \mathbf{d}_1$ implies $\phi(\mathbf{d}'_1) \rightarrow \phi(\mathbf{d}_1)$ (modulo renaming of parameters), we have that $\phi(\mathbf{d}'_1) \wedge c \rightarrow \phi(\mathbf{d}_1) \wedge c$. By validity of $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$, we conclude that $\forall \upsilon(\mathbf{d}'_1)\ \exists \upsilon(\mathbf{d}_2)\ \forall \mathbf{v}.\ \phi(\mathbf{d}'_1) \wedge c \rightarrow \phi(\mathbf{d}_2)$ is true, i.e. $\mathbf{d}'_1 \vdash c \rightarrow \mathbf{d}_2$ is valid.

(b). If $\mathcal{R} \models c' \rightarrow c$ holds, then $\forall \upsilon(\mathbf{d}'_1)\ \exists \upsilon(\mathbf{d}_2)\ \forall \mathbf{v}.\ \phi(\mathbf{d}'_1) \wedge c \rightarrow \phi(\mathbf{d}_2)$ implies $\forall \upsilon(\mathbf{d}'_1)\ \exists \upsilon(\mathbf{d}_2)\ \forall \mathbf{v}.\ \phi(\mathbf{d}'_1) \wedge c' \rightarrow \phi(\mathbf{d}_2)$, i.e. $\mathbf{d}'_1 \vdash c' \rightarrow \mathbf{d}_2$ is valid.

(c). Since $\mathbf{d}_2 \geq_t \mathbf{d}'_2$ implies $\phi(\mathbf{d}_2) \rightarrow \phi(\mathbf{d}'_2)$ (modulo renaming of parameters), then $\forall \upsilon(\mathbf{d}'_1)\ \exists \upsilon(\mathbf{d}_2)\ \forall \mathbf{v}.\ \phi(\mathbf{d}'_1) \wedge c' \rightarrow \phi(\mathbf{d}_2)$ implies $\forall \upsilon(\mathbf{d}'_1)\ \exists \upsilon(\mathbf{d}'_2)\ \forall \mathbf{v}.\ \phi(\mathbf{d}'_1) \wedge c' \rightarrow \phi(\mathbf{d}'_2)$, i.e. $\mathbf{d}'_1 \vdash c' \rightarrow \mathbf{d}'_2$ is valid. □

We state a few intuitive properties of the $\vdash$ relation.

LEMMA A.1. *Let $\mathbf{d}_2$ be $x_1 : \tau_1, \ldots, x_n : \tau_n$. The following are equivalent:*

(i). $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ *is valid;*

(ii). $\mathbf{d}_1 \vdash c \rightarrow nf(\mathbf{d}_2)$ *is valid;*

(iii). $nf(\mathbf{d}_1) \vdash c \rightarrow \mathbf{d}_2$ is valid;

(iv). for $i = 1..n$, $\mathbf{d}_1 \vdash c \rightarrow x_i : \tau_i$ is valid.

Proof. *(i)* is equivalent to *(ii)* and to *(iii)* since in order to obtain the normal form, we have to go through a sequence of deletions (remove duplicate atd's or that assign lower types than existing ones) and/or mergings (merge $x : \sqcap, x : \sqcup$ into $x : \Box$). Merging has no change on $\phi()$. Removing duplicates or lower types yields an equivalent formula. For instance, $\forall a, b \exists c, d \forall x_i . x_i \leq a \wedge x_i = b \rightarrow x_i \leq c \wedge x_i = d$ holds iff $\forall b \exists d \forall x_i . x_i = b \rightarrow x_i = d$. The if part follows by setting $c = d$. The only-if part follows by setting $a = b$.

(i) is equivalent to *(iv)* since, by distributivity laws, we can rewrite $\forall \upsilon(\mathbf{d}_1)\ \exists \upsilon(\mathbf{d}_2)\ \forall \mathbf{v}.\phi(\mathbf{d}_1) \wedge c \rightarrow \phi(\mathbf{d}_2)$ into $\bigwedge_{i=1..n} \forall \upsilon(\mathbf{d}_1) \exists \upsilon(x_i : \tau_i) \forall \mathbf{v}.\phi(\mathbf{d}_1) \wedge c \rightarrow \phi(x_i : \tau_i)$, which is our conclusion. $\Box$

When showing validity of $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$, atd's in $\mathbf{d}_2$ typing variables not in $vars(c)$ can be dealt with by means of the $\geq_t$ relation.

Lemma A.2. *Let $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ be a type assertion with c satisfiable. Called $\mathbf{v}_c = vars(c)$, we have that: $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is valid iff (i) $\mathbf{d}_1|_{\mathbf{v}_c} \vdash c \rightarrow \mathbf{d}_2|_{\mathbf{v}_c}$ is valid; and (ii) $nf(\mathbf{d}_1) \geq_t nf(\mathbf{d}_2|_{vars(\mathbf{d}_2)\setminus\mathbf{v}_c})$.*

Proof. By Lemma A.1*(i-iii)*, we can assume that $\mathbf{d}_1$ and $\mathbf{d}_2$ are in normal form. Also, by Lemma A.1 *(iv)*, we have to show that for every $x : \tau$ in $\mathbf{d}_2$: $\mathbf{d}_1 \vdash c \rightarrow x : \tau$ is valid iff *(i)* $\mathbf{d}_1|_{\mathbf{v}_c} \vdash c \rightarrow x : \tau$ is valid if $x \in vars(c)$; and *(ii)* $nf(\mathbf{d}_1) \geq_t nf(x : \tau)$ if $x \notin vars(c)$. The result is immediate for $\tau = \star$, so we can assume $\tau \neq \star$ in the following. Let us distinguish two cases.

($x \in vars(c)$).

Let $\mathbf{v} = vars(\mathbf{d}_1) \cup vars(c)$. By definition, $\mathbf{d}_1 \vdash c \rightarrow x : \tau$ is valid iff:

$$\forall \upsilon(\mathbf{d}_1) \exists \upsilon(x : \tau) \forall \mathbf{v}.(\phi(\mathbf{d}_1) \wedge c) \rightarrow \phi(x : \tau). \tag{3}$$

Called $\mathbf{d}_1|_{-\mathbf{v}_c} = \mathbf{d}_1|_{vars(\mathbf{d}_1)\setminus\mathbf{v}_c}$, it can be rewritten as:

$$\forall \upsilon(\mathbf{d}_1)(\exists \upsilon(x : \tau) \forall \mathbf{v}_c.(\phi(\mathbf{d}_1|_{\mathbf{v}_c}) \wedge c) \rightarrow \phi(x : \tau)) \vee (\forall \mathbf{v} \setminus \mathbf{v}_c.\neg\phi(\mathbf{d}_1|_{-\mathbf{v}_c})).$$

The formula $\forall \mathbf{v} \setminus \mathbf{v}_c.\neg\phi(\mathbf{d}_1|_{-\mathbf{v}_c})$ is false for any instance of $\upsilon(\mathbf{d}_1)$. A solution for $\phi(\mathbf{d}_1|_{-\mathbf{v}_c})$ can be found by setting variables to the values of the parameters they are constrained by. As an example, for $a \leq x \leq a + 2$, set $x = a$. Since $\mathbf{d}_1$ is in normal form, this definition is well-formed. Therefore, (3) reduces to $\forall \upsilon(\mathbf{d}_1) \exists \upsilon(x : \tau) \forall \mathbf{v}_c.(\phi(\mathbf{d}_1|_{\mathbf{v}_c}) \wedge c) \rightarrow \phi(x : \tau)$. Since universal quantification over $\upsilon(\mathbf{d}_1|_{-\mathbf{v}_c})$ has no effect, this is equivalent to the desired conclusion: $\mathbf{d}_1|_{\mathbf{v}_c} \vdash c \rightarrow x : \tau$ is valid.

($x \notin vars(c)$).

Let $\mathbf{v} = vars(\mathbf{d}_1) \cup vars(c) \cup \{x\}$. By definition, $\mathbf{d}_1 \vdash c \rightarrow x : \tau$ is valid iff (3) holds. Called $\mathbf{d}^1_x = \mathbf{d}_1|_{\{x\}}$ and $\mathbf{d}^1_{-x} = \mathbf{d}_1|_{\mathbf{v}\setminus\{x\}}$, (3) can be rewritten as:

$$\forall \upsilon(\mathbf{d})(\exists \upsilon(x : \tau) \forall x.\phi(\mathbf{d}^1_x) \rightarrow \phi(x : \tau)) \vee (\forall \mathbf{v} \setminus \{x\}.\neg(\phi(\mathbf{d}^1_{-x}) \wedge c)).$$

since $vars(\mathbf{d}^1_x) \cup vars(x : \tau) \subseteq \{x\}$ and $vars(\mathbf{d}^1_{-x}) \cup vars(c) \subseteq \mathbf{v} \setminus \{x\}$. Moreover, by factoring out universal quantification over $\upsilon(\mathbf{d})$, it can be further rewritten as:

$$(\forall \upsilon(\mathbf{d}^1_x) \exists \upsilon(x : \tau) \forall x.\phi(\mathbf{d}^1_x) \rightarrow \phi(x : \tau)) \vee (\forall \upsilon(\mathbf{d}^1_{-x}) \forall \mathbf{v} \setminus \{x\}.\neg(\phi(\mathbf{d}^1_{-x}) \wedge c)).$$

The formula $\forall \upsilon(\mathbf{d}^1_{-x})\forall \mathbf{v} \setminus \{x\}.\neg(\phi(\mathbf{d}^1_{-x}) \wedge c)$ is false, since c is satisfiable. In fact, a solution of c can be extended to a solution of $\phi(\mathbf{d}^1_{-x}) \wedge c$ by setting the value of a parameter in $\upsilon(\mathbf{d}^1_{-x})$ to the value of the variable it constraints, e.g., by setting a to the value of y if $a \leq y \leq a + 2$ is in $\phi(\mathbf{d}^1_{-x})$. Since $\mathbf{d}_1$ is in normal form, this definition is well-formed.

Therefore, (3) reduces $\forall \upsilon(\mathbf{d}^1_x)\exists \upsilon(x : \tau)\forall x.\phi(\mathbf{d}^1_x) \rightarrow \phi(x : \tau)$, namely to validity of $\mathbf{d}^1_x \vdash \texttt{true} \rightarrow x : \tau$. We have then to show that $\mathbf{d}^1_x \vdash \texttt{true} \rightarrow x : \tau$ is valid iff $\mathbf{d}^1_x \geq_t x : \tau$ holds, where, since $\mathbf{d}_1$ is in normal form, $\mathbf{d}^1_x$ is either empty or $x : \mu$, with $\mu \neq \star$. If $\mathbf{d}^1_x$ is empty, then neither $\mathbf{d}^1_x \vdash \texttt{true} \rightarrow x : \tau$ can be valid nor $\mathbf{d}^1_x \geq_t x : \tau$ holds, since $\tau \neq \star$. If $\mathbf{d}^1_x$ is $x : \mu$, then $\mathbf{d}^1_x \geq_t x : \tau$ iff $\mu \geq_t \tau$ which, by definition of $\geq_t$, holds iff $x : \mu \vdash \texttt{true} \rightarrow x : \tau$. □

We are now in the position to show the other results of Section 2.1.

Lemma 2.11.

Proof. An immediate instance of Lemma A.2, by noting that $vars(c) = \emptyset$. □

Lemma 2.12.

Proof. Let $\mathbf{v} = vars(c) \cup vars(\mathbf{d}_1) \cup vars(\mathbf{d}_2) \cup vars(\mathbf{d}_3)$. Consider an instance $\mathbf{u}_1$ of parameters in $\upsilon(\mathbf{d}_1)$. By validity of $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$, there exists $\mathbf{u}_2$ instance of parameters in $\upsilon(\mathbf{d}_2)$ such that $\forall \mathbf{v}.\phi(\mathbf{d}_1)[\upsilon(\mathbf{d}_1)/\mathbf{u}_1] \wedge c \rightarrow \phi(\mathbf{d}_2)[\upsilon(\mathbf{d}_2)/\mathbf{u}_2]$ is true. This implies that:

$$\forall \mathbf{v}.\phi(\mathbf{d}_1)[\upsilon(\mathbf{d}_1)/\mathbf{u}_1] \wedge c \rightarrow \phi(\mathbf{d}_1)[\upsilon(\mathbf{d}_1)/\mathbf{u}_1] \wedge c \wedge \phi(\mathbf{d}_2)[\upsilon(\mathbf{d}_2)/\mathbf{u}_2]$$

is true. By validity of $\mathbf{d}_1, \mathbf{d}_2 \vdash c \rightarrow \mathbf{d}_3$, there exists $\mathbf{u}_3$ instance of parameters in $\upsilon(\mathbf{d}_3)$ such that:

$$\forall \mathbf{v}.\phi(\mathbf{d}_1)[\upsilon(\mathbf{d}_1)/\mathbf{u}_1] \wedge \phi(\mathbf{d}_2)[\upsilon(\mathbf{d}_2)/\mathbf{u}_2] \wedge c \rightarrow \phi(\mathbf{d}_3)[\upsilon(\mathbf{d}_3)/\mathbf{u}_3]$$

is true. By transitivity of implication, we conclude that:

$$\forall \mathbf{v}.\phi(\mathbf{d}_1)[\upsilon(\mathbf{d}_1)/\mathbf{u}_1] \wedge c \rightarrow \phi(\mathbf{d}_3)[\upsilon(\mathbf{d}_3)/\mathbf{u}_3]$$

is true. By eliminating from the quantification over $\mathbf{v}$ those variables not in $vars(c) \cup vars(\mathbf{d}_1) \cup vars(\mathbf{d}_3)$ and by reintroducing existential quantification over $\upsilon(\mathbf{d}_3)$ and universal quantification over $\upsilon(\mathbf{d}_1)$, we get the desired conclusion. □

A.2 Section 2.2

Lemma 2.14.

Proof. (One solution) By Lemma A.1 *(iv)*, we can restrict to the case $\mathbf{d}_1 \vdash c \rightarrow x : \tau$, for $x \in \mathbf{v}$. Let $\mathcal{V} = \{r \in \mathcal{R} \mid \mathbf{d}_1 \vdash c \rightarrow x : \Box_r \text{ is valid}\}$.

If $\mathcal{V} = \emptyset$, then by defining $\tau = lub\{\mu \in \mathcal{BT}_\Box \mid \mathbf{d}_1 \vdash c \rightarrow x : \mu \text{ is valid}\}$, we have that $\mathbf{d}_2 = x : \tau$ satisfies the requirements of Definition 2.13.

If $\mathcal{V} \neq \emptyset$, we distinguish two cases based on $\overline{r} = inf\ \mathcal{V}$, the infinum of $\mathcal{V}$.

Assume first that $\overline{r} \in \mathcal{V}$. Then $\mathbf{d}_1 \vdash c \rightarrow x : \tau$ is valid iff $\tau \in \mathcal{BT}_\Box$ or $\tau = \Box_r$ with $r \in \mathcal{V}$ iff $\Box_{\overline{r}} \geq_t \tau$ iff $x : \Box_{\overline{r}} \geq_t x : \tau$. Thus, $\mathbf{d}_2 = x : \Box_{\overline{r}}$ satisfies the conclusion.

Assume now that $\overline{r} \notin \mathcal{V}$. Consider an instance $\mathbf{u}$ of parameters in $\upsilon(\mathbf{d}_1)$. Since $(\phi(\mathbf{d}_1) \wedge c)[\upsilon(\mathbf{d}_1)/\mathbf{u}]$ is a linear constraint, the set of its solutions $\mathcal{S}_\mathbf{u}$ is a closed set (actually, it is a polyhedron). For some a, $\mathcal{S}_{\mathbf{u}}$ is included in the set of solutions of $\phi(x : \Box_r) = a \leq x \leq a+r$ for any $r \in \mathcal{V}$, namely for any $r > \overline{r}$. Since $\mathcal{S}_{\mathbf{u}}$ is a closed set, it is included in the set of solutions of $\phi(x : \Box_{\overline{r}}) = a \leq x \leq a+\overline{r}$ as well. Since this holds for any $\mathbf{u}$, we conclude that $\mathbf{d}_1 \vdash c \rightarrow x : \Box_{\overline{r}}$ is valid, namely that $\overline{r} \in \mathcal{V}$, which is absurd.

(Only one normal form) Assume there exist two type declarations $\mathbf{d}_2$ and $\mathbf{d}_2'$ satisfying the inference problem. Since $vars(\mathbf{d}_2) \subseteq \mathbf{v}$ and $\mathbf{d}_2 \geq_t \mathbf{d}_2$, by the assumption that $\mathbf{d}_2$ is a solution we have $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is valid. This and the assumption that $\mathbf{d}_2'$ is a solution imply $\mathbf{d}_2' \geq_t \mathbf{d}_2$, from which we have $nf(\mathbf{d}_2') \geq_t nf(\mathbf{d}_2)$. Analogously, one shows $nf(\mathbf{d}_2) \geq_t nf(\mathbf{d}_2')$. By definition of $\geq_t$, we conclude that $x : \tau$ is in $nf(\mathbf{d}_2)$ iff it is in $nf(\mathbf{d}_2')$, i.e. $nf(\mathbf{d}_2) = nf(\mathbf{d}_2')$ modulo reordering of atd's. □

Lemma 2.17.

Proof. (Output "valid") By soundness of χ, $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}$ is valid. By the monotonicity Lemma 2.8 and the test $\mathbf{d} \geq_t \mathbf{d}_2$ at *Step 1*, $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is valid as well.

(Output "not valid") By completeness of χ for $\mathcal{B}$, if $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ were valid, then the test $\mathbf{d} \geq_t \mathbf{d}_2$ would have passed. □

A.3 Section 3.2

Lemma 3.4.

Proof. The only-if part is trivial. Concerning the if part, we observe that, given $\mathbf{x} \in Sol(\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c)$, we can choose parameter values that are exactly equal to the value of variables they constraint in $\phi(\mathbf{d})$, e.g., when $a \leq x \leq a+2$ is in $\phi(\mathbf{d})$ we fix a to the value of x in $\mathbf{x}$. Since parameters in $\upsilon(x : \tau)$ are fresh variables, this definition is well-formed. □

We recall the following result, which relates the solutions of a linear programming problem over $\mathbf{Ax} \leq \mathbf{b}$ to the ones over its homogeneous version $\mathbf{Ax} \leq \mathbf{0}$.

Theorem A.3. *Let $Sol(\mathbf{Ax} \leq \mathbf{b})$ be a non-empty polyhedron. We have:*

$$max\ \{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\} \in \mathcal{R} \quad \text{iff} \quad max\ \{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{0}\} = 0$$

Proof. See [Murty 1983, Corollary 3.1]. □

Lemma 3.5.

Proof. Consider the if part. By Lemma 3.4, there exists some $\mathbf{u}$ such that $Sol(\mathcal{P}, \mathbf{u}) \neq \emptyset$. By hypothesis, we have $max\{\mathbf{c}^T\mathbf{v} \mid \mathbf{v} \in Sol(\mathcal{P}, \mathbf{u})\} \in \mathcal{R}$. By Theorem A.3, we conclude $max\{\mathbf{c}^T\mathbf{v} \mid \mathbf{v} \in Sol(\mathcal{H})\} = 0$. Consider now the only-if part. Let $\mathbf{u}$ be such that $Sol(\mathcal{P}, \mathbf{u}) \neq \emptyset$. By Theorem A.3, $max\{\mathbf{c}^T\mathbf{v} \mid \mathbf{v} \in Sol(\mathcal{H})\} = 0$ implies $max\{\mathbf{c}^T\mathbf{v} \mid \mathbf{v} \in Sol(\mathcal{P}, \mathbf{u})\} \in \mathcal{R}$. □

Theorem 3.7 (LPInfer - soundness and completeness).

Proof. *(Soundness)* Working on $\mathbf{n} = nf(\mathbf{d}_1)$ at *Step 0* is correct by Lemma A.1. By the same lemma, each variable x in $\mathbf{v}$ can be considered separately.

Step 2 is justified by Lemma 3.4.

If $x \notin \mathbf{v}_c = vars(c)$ then, by Lemma A.2, validity of $\mathbf{n} \vdash c \rightarrow x : \tau$ is equivalent to $nf(\mathbf{n}) = \mathbf{n} \geq_t x : \tau$. This holds iff either $x : \tau$ is in $\mathbf{n}$, which justifies *Step 3* (a), or $\tau = \star$, which justifies *Step 3* (b).

If $x \in \mathbf{v}_c$ then validity of $\mathbf{n} \vdash c \rightarrow x : \tau$ is equivalent to validity of $\mathbf{n}|_{\mathbf{v}_c} \vdash c \rightarrow x : \tau$, i.e., by the definition of $\mathbf{d}$ at *Step 0*, to validity of $\mathbf{d} \vdash c \rightarrow x : \tau$. The outputs at *Step 4* are then justified by Lemma 3.5.

(Completeness) By Lemma A.1 we can consider each variable x in $\mathbf{v}$ separately. Assume that $\mathbf{d}_1 \vdash c \rightarrow x : \tau$ is valid. If c is unsatisfiable, completeness follows by Lemma 3.4 and *Step 2*. If $x \notin \mathbf{v}_c$, completeness follows since *Step 3* in the soundness part hold as if and only-if by Lemma A.2. If $x \in \mathbf{v}_c$ and $\tau \in \mathcal{BT}_\Box$, completeness follows since *Step 4* holds as if and only-if by Lemma 3.5. $\Box$

Lemma 3.9.

Proof. If $\mathbf{c}^T\mathbf{R} \leq \mathbf{0}$ then $\mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} \leq 0$ for every $\boldsymbol{\lambda} \geq \mathbf{0}$. Moreover, for $\boldsymbol{\lambda} = 0$ we have $\mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} = 0$. Summarizing, $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{0}\} = max\{\mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} \mid \boldsymbol{\lambda} \geq \mathbf{0}\} = 0$.

Conversely, if $\mathbf{c}^T\mathbf{R} \not\leq \mathbf{0}$ then there exists a column $\mathbf{R}_j$ of $\mathbf{R}$ such that $\mathbf{c}^T\mathbf{R}_j > 0$. By choosing $\boldsymbol{\lambda}'_i = 0$ for $i \neq j$ and $\boldsymbol{\lambda}'_j = 1$, for $\mathbf{x} = \mathbf{R}\boldsymbol{\lambda}'$ we get $\mathbf{c}^T\mathbf{R}\boldsymbol{\lambda}' = \mathbf{c}^T\mathbf{R}_j > 0$. Summarizing, $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{0}\} = max\{\mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} \mid \boldsymbol{\lambda} \geq \mathbf{0}\} \geq \mathbf{c}^T\mathbf{R}\boldsymbol{\lambda}' > 0$. $\Box$

A.4 Section 3.3

We recall that a polyhedron is a convex set, namely for $\mathbf{x}, \mathbf{y}$ in $Sol(\mathbf{Ax} \leq \mathbf{b})$ and any $0 \leq \lambda \leq 1$ the vector $\lambda\mathbf{x} + (1-\lambda)\mathbf{y}$ belongs to $Sol(\mathbf{Ax} \leq \mathbf{b})$ as well. Geometrically, this means that the segment from $\mathbf{x}$ to $\mathbf{y}$ is included in the polyhedron.

Theorem 3.12 [(Definiteness I)].

Proof. The if-part is immediate by Lemma 2.8 since $\mathbf{d} \geq_t \mathbf{d}|_!$.

Consider now the only-if part. If c is unsatisfiable, the conclusion readily follows. Also, we can assume that $\mathbf{d}$ is in normal form. In fact, let $\mathbf{d} \vdash c \rightarrow x :!$ be valid. By Lemma A.1, $nf(\mathbf{d}) \vdash c \rightarrow x :!$ is valid as well. If we could show the conclusion for normal forms, we get that $nf(\mathbf{d})|_! \vdash c \rightarrow x :!$ is valid. Finally, since $nf(\mathbf{d})|_! = \mathbf{d}|_!$, we conclude that $\mathbf{d}|_! \vdash c \rightarrow x :!$ is valid.

Let $\mathbf{v} = vars(c) \cup vars(\mathbf{d}) \cup \{x\}$, and let i such that $\mathbf{v}_i = x$. Since the order of atd's in $\mathbf{d}$ is irrelevant for validity, we can assume that $\mathbf{d} = \mathbf{d}_1, \mathbf{d}|_!$, with $\mathbf{d}_1$ assigning no variable with the ! type. We reason by induction on the length of $\mathbf{d}_1$. When $\mathbf{d}_1$ is the empty sequence, the conclusion is immediate since $\mathbf{d} = \mathbf{d}|_!$. Assume now that $\mathbf{d}_1 = y : \tau, \mathbf{d}_2$, with $\tau \neq !$, and let j such that $\mathbf{v}_j = y$. We will show that $\mathbf{d}_2, \mathbf{d}|_! \vdash c \rightarrow x :!$ is valid, and then, by inductive hypothesis, we have that $\mathbf{d}|_! \vdash c \rightarrow x :!$ is valid. Let us consider the possible cases for τ.

$(\tau = \star)$. This cannot occur, since $\mathbf{d}$ is in normal form.

$(\tau = \sqcap)$. Assume, by absurd, that $\mathbf{d}_2, \mathbf{d}|_! \vdash c \rightarrow x :!$ is not valid. Called $\overline{\mathbf{d}} = \mathbf{d}_2, \mathbf{d}|_!$, there exists an instance $\mathbf{u}$ of parameters $\mathbf{a} = v(\overline{\mathbf{d}})$ of the parameterized system $\phi(\overline{\mathbf{d}}) \wedge c$ such that the instantiated system has at least two solution points $\mathbf{w}$ and $\mathbf{z}$ which differ on the value of x, i.e., in formulas:

$$\mathbf{A}_c\mathbf{w} \leq \mathbf{b}_c, \mathbf{A}_{\overline{\mathbf{d}}}\mathbf{w} \leq \mathbf{B}_{\overline{\mathbf{d}}}\mathbf{u} \quad \text{and} \quad \mathbf{A}_c\mathbf{z} \leq \mathbf{b}_c, \mathbf{A}_{\overline{\mathbf{d}}}\mathbf{z} \leq \mathbf{B}_{\overline{\mathbf{d}}}\mathbf{u},$$

and $\mathbf{w}_i \neq \mathbf{z}_i$. Let $m = max\{\mathbf{w}_j, \mathbf{z}_j\}$. We have: $\mathbf{A}_c\mathbf{w} \leq \mathbf{b}_c, \mathbf{A}_{\overline{\mathbf{d}}}\mathbf{w} \leq \mathbf{B}_{\overline{\mathbf{d}}}\mathbf{u}, \mathbf{w}_j \leq m$ and $\mathbf{A}_c\mathbf{z} \leq \mathbf{b}_c, \mathbf{A}_{\overline{\mathbf{d}}}\mathbf{z} \leq \mathbf{B}_{\overline{\mathbf{d}}}\mathbf{u}, \mathbf{z}_j \leq m$, where $\mathbf{w}_j \leq m$ and $\mathbf{z}_j \leq m$ correspond to the instantiation of $\phi(y : \sqcap)$. Therefore, there exists a parameter instance $(\mathbf{u}\ \ m)$ in the parameter space augmented by one dimension (needed since y is not typed in $\overline{\mathbf{d}}$ as

$\mathbf{d}$ is in normal form) such that $\phi(\mathbf{d}) \wedge c$ admits two solutions with distinct values for x. Summarizing, $\mathbf{d} \vdash c \rightarrow x :!$ is not valid, contrarily to the hypothesis.

($\tau = \sqcup$). Analogous to $\tau = \sqcap$, but considering $m = min\{\mathbf{w}_j, \mathbf{z}_j\}$.

($\tau = \Box$). Analogous to $\tau = \sqcap$, but now consider both an upper bound $m_1 = max\{\mathbf{w}_j, \mathbf{z}_j\}$ and a lower bound $m_2 = min\{\mathbf{w}_j, \mathbf{z}_j\}$.

($\tau = \Box_s$, with $s > 0$). Analogous to $\tau = \Box$, under the assumption that $abs(\mathbf{w}_j - \mathbf{z}_j) \leq s$. Let us show that such an assumption holds. Let $\mathbf{w}$ and $\overline{\mathbf{z}}$ be any two solutions such that $\mathbf{w}_i \neq \overline{\mathbf{z}}_i$. If $abs(\mathbf{w}_j - \overline{\mathbf{z}}_j) \leq s$, we are done. Otherwise, since polyhedra are convex sets, $\mathbf{z} = \lambda\mathbf{w} + (1-\lambda)\overline{\mathbf{z}}$ is also a solution for every $0 \leq \lambda \leq 1$. In particular, for $\lambda_0 = 1 - s/abs(\mathbf{w}_j - \overline{\mathbf{z}}_j)$, we have:

$$abs(\mathbf{w}_j - \mathbf{z}_j) = abs((1-\lambda_0)(\mathbf{w}_j - \overline{\mathbf{z}}_j)) = (1-\lambda_0)abs(\mathbf{w}_j - \overline{\mathbf{z}}_j) = s.$$

Moreover, $s > 0$ implies $\lambda_0 < 1$, and then $\mathbf{z}_i = \lambda_0\mathbf{w}_i + (1-\lambda_0)\overline{\mathbf{z}}_i \neq \mathbf{w}_i$, since $\mathbf{w}_i \neq \overline{\mathbf{z}}_i$. Summarizing, the assumptions that $\mathbf{w}$ and $\mathbf{z}$ are two solutions with $\mathbf{w}_i \neq \mathbf{z}_i$ and $abs(\mathbf{w}_j - \mathbf{z}_j) \leq s$ are satisfied. $\Box$

Lemma 3.14

Proof. Since $\phi(\mathbf{d}|_!)$ is of the form $\mathbf{x} = \mathbf{a}$, the overall system $\phi(\mathbf{d}|_!) \wedge c$ is a linear system of equalities $\mathbf{x} = \mathbf{a}, \mathbf{A}_c\mathbf{v} = \mathbf{b}_c$. By Gauss-Jordan elimination of variables in $\mathbf{v} \setminus \mathbf{x}$, the system can be transformed into the equivalent following form:

$$\mathbf{x} = \mathbf{a}, \mathbf{I}\mathbf{w} = \mathbf{b}' + \mathbf{A}'\mathbf{z} + \mathbf{B}'\mathbf{x}, \mathbf{0} = \mathbf{b}'' + \mathbf{B}''\mathbf{x}, \qquad (4)$$

where $\mathbf{I}$ is a diagonal matrix, $\mathbf{w}$ and $\mathbf{z}$ are a partition of variables in $\mathbf{v} \setminus \mathbf{x}$ with $\mathbf{z}$ free to assume any value, and $\mathbf{0} = \mathbf{b}'' + \mathbf{B}''\mathbf{x}$ is the condition of satisfiability of the system (4). If we show the conclusion for (4), then it holds for the system:

$$\mathbf{I}\mathbf{w} = \mathbf{A}'\mathbf{z} + \mathbf{B}'\mathbf{x}, \mathbf{0} = \mathbf{B}''\mathbf{x},$$

since the proof obligations do not involve $\mathbf{a}$, $\mathbf{b}'$ nor $\mathbf{b}''$.

(Only-if part). Assume $\mathbf{d}|_! \vdash c \rightarrow x :!$ valid. If x is in $\mathbf{x}$, we are done. x cannot be in $\mathbf{z}$ otherwise it would be a variable free to assume any value, fixed any $\mathbf{u}$ instance of $\mathbf{a}$ such that the system (4) is satisfiable. Finally, let x be $\mathbf{w}_i$, for some i. Then $x = \mathbf{b}'_i + row(\mathbf{A}', i)\mathbf{z} + row(\mathbf{B}', i)\mathbf{x}$ is in (4). If $\mathbf{r}^T = row(\mathbf{A}', i) = \mathbf{0}$, we are done. Otherwise, assuming $\mathbf{r}_j \neq 0$ for some j, we can fix $\mathbf{z} = \mathbf{0}$ except for $\mathbf{z}_j$ which is free to assume any value. This results in $x = \mathbf{b}'_i + \mathbf{r}_j\mathbf{z}_j + row(\mathbf{B}', i)\mathbf{x}$, which leads to infinitely many distinct solutions for x.

(If-part). For any $\mathbf{u}$ instance of $\mathbf{a}$ such that the system (4) is satisfiable, every variable in $\mathbf{x}$ trivially assumes a single value. This property extends to variables $\mathbf{w}_i$ for which $row(\mathbf{A}', i) = \mathbf{0}$, since $\mathbf{w}_i$ is defined as:

$$\mathbf{w}_i = \mathbf{b}'_i + row(\mathbf{A}', i)\mathbf{z} + row(\mathbf{B}', i)\mathbf{x}$$

and $\mathbf{x} = \mathbf{u}$. $\Box$

A linear system $\mathbf{A}^=\mathbf{x} = \mathbf{b}^=, \mathbf{A}^+\mathbf{x} \leq \mathbf{b}^+$ as in Theorem 3.16 has a solution $\mathbf{x}_0$ for which $\mathbf{A}^+\mathbf{x}_0 < \mathbf{b}^+$. $\mathbf{x}_0$ is called an inner point.

Lemma A.4. *Assume that $Sol(\mathbf{A}\mathbf{x} \leq \mathbf{b}) \neq \emptyset$, and let $\mathbf{A}^=\mathbf{x} = \mathbf{b}^=, \mathbf{A}^+\mathbf{x} \leq \mathbf{b}^+$ as in Theorem 3.16. There exists $\mathbf{x}_0 \in Sol(\mathbf{A}\mathbf{x} \leq \mathbf{b})$ such that $\mathbf{A}^+\mathbf{x}_0 < \mathbf{b}^+$. Moreover, for every $\mathbf{y}_0$ such that $\mathbf{A}^=\mathbf{y}_0 = \mathbf{b}^=$ there exists $0 < \lambda \leq 1$ such that for every $0 < \lambda_0 \leq \lambda$, $\mathbf{z}_0 = \lambda_0\mathbf{y}_0 + (1-\lambda_0)\mathbf{x}_0$ is in $Sol(\mathbf{A}\mathbf{x} \leq \mathbf{b})$.*

PROOF. Assume that $\mathbf{A}^+\mathbf{x} \leq \mathbf{b}^+$ consists of $\mathbf{c}_i^T\mathbf{x} \leq b_i$ for $i = 1, \ldots, n$. Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathit{Sol}(\mathbf{A}\mathbf{x} \leq \mathbf{b})$ such that $\mathbf{c}_i^T\mathbf{x}_i < b_i$ for $i = 1 \ldots n$. Let $\mathbf{x}_0 = (\mathbf{x}_1 + \ldots + \mathbf{x}_n)/n$. We calculate $\mathbf{A}^=\mathbf{x}_0 = (\mathbf{A}^=\mathbf{x}_1 + \ldots + \mathbf{A}^=\mathbf{x}_n)/n = (n\mathbf{b}^=)/n = \mathbf{b}^=$. Moreover, for $i = 1 \ldots n$ we calculate $\mathbf{c}_i^T\mathbf{x}_0 = (\mathbf{c}_i^T\mathbf{x}_1 + \ldots + \mathbf{c}_i^T\mathbf{x}_n)/n \leq (\mathbf{c}_i^T\mathbf{x}_i + (n-1)b_i)/n < b_i$ since $\mathbf{c}_i^T\mathbf{x}_i < b_i$ and $\mathbf{c}_i^T\mathbf{x}_j \leq b_i$ for $j \neq i$.

Consider now the second part of the lemma. First, for any $0 \leq \lambda \leq 1$, we have $\mathbf{A}^=\mathbf{z}_0 = \lambda\mathbf{A}^=\mathbf{y}_0 + (1-\lambda)\mathbf{A}^=\mathbf{x}_0 = \lambda\mathbf{b}^= + (1-\lambda)\mathbf{b}^= = \mathbf{b}^=$. Let now fix:

$$\lambda = \mathit{min}(\{1\} \cup \{(b_i - \mathbf{c}_i^T\mathbf{x}_0)/(\mathbf{c}_i^T\mathbf{y}_0 - \mathbf{c}_i^T\mathbf{x}_0) \mid i = 1 \ldots n, \mathbf{c}_i^T\mathbf{y}_0 > \mathbf{c}_i^T\mathbf{x}_0\}).$$

We have $\lambda > 0$ since $\mathbf{c}_i^T\mathbf{x}_0 < b_i$ for $i = 1..n$. Let us now calculate, for $i = 1..n$:

—if $\mathbf{c}_i^T\mathbf{y}_0 \leq \mathbf{c}_i^T\mathbf{x}_0$, then $\mathbf{c}_i^T\mathbf{z}_0 = \lambda\mathbf{c}_i^T\mathbf{y}_0 + (1-\lambda)\mathbf{c}_i^T\mathbf{x}_0 \leq \lambda\mathbf{c}_i^T\mathbf{x}_0 + (1-\lambda)\mathbf{c}_i^T\mathbf{x}_0 = \mathbf{c}_i^T\mathbf{x}_0 < b_i$;

—if $\mathbf{c}_i^T\mathbf{y}_0 > \mathbf{c}_i^T\mathbf{x}_0$, then $\mathbf{c}_i^T\mathbf{z}_0 = \lambda\mathbf{c}_i^T\mathbf{y}_0 + (1-\lambda)\mathbf{c}_i^T\mathbf{x}_0 = \lambda(\mathbf{c}_i^T\mathbf{y}_0 - \mathbf{c}_i^T\mathbf{x}_0) + \mathbf{c}_i^T\mathbf{x}_0 \leq (b_i - \mathbf{c}_i^T\mathbf{x}_0) + \mathbf{c}_i^T\mathbf{x}_0 = b_i$.

Summarizing, $\mathbf{z}_0 \in \mathit{Sol}(\mathbf{A}^=\mathbf{x} = \mathbf{b}^=, \mathbf{A}^+\mathbf{x} \leq \mathbf{b}^+) = \mathit{Sol}(\mathbf{A}\mathbf{x} \leq \mathbf{b})$. The same conclusion readily follows for any $0 < \lambda_0 \leq \lambda$. □

LEMMA 3.18

PROOF. The if-part follows by the monotonicity Lemma 2.8, since $\mathcal{R} \models c \rightarrow ie(c)$. We show the only-if part by contraposition. Let $\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c$ be the geometric representation of c and $\mathbf{A}^=\mathbf{v} \leq \mathbf{b}^=$ be the one of $ie(c)$, where $\mathbf{v} = vars(c) \cup vars(\mathbf{d}|_!) \cup \{x\}$. Also, let i such that $\mathbf{v}_i = x$. Called $\mathbf{x} = vars(\mathbf{d}|_!)$, $\phi(\mathbf{d}|_!)$ is of the form $\mathbf{x} = \mathbf{a}$.

Assume that $\mathbf{d}|_! \vdash ie(c) \rightarrow x :!$ is not valid. Since c is satisfiable, this implies that there exist $\mathbf{u}$ instance of $\mathbf{a}$, and $\mathbf{w}^0, \mathbf{w}^1 \in \mathit{Sol}(\mathbf{x} = \mathbf{u}, \mathbf{A}^=\mathbf{v} \leq \mathbf{b}^=)$ such that $\mathbf{w}_i^0 \neq \mathbf{w}_i^1$. Consider now the linear constraint $c' = \mathbf{x} = \mathbf{a} \wedge c$ in the space of variables plus parameters. We have that $ie(c')$ is $\mathbf{x} = \mathbf{a}, \mathbf{A}^=\mathbf{v} \leq \mathbf{b}^=$. Also, $(\mathbf{w}^0, \mathbf{u})$ and $(\mathbf{w}^1, \mathbf{u})$ belong to $\mathit{Sol}(\mathbf{x} = \mathbf{a}, \mathbf{A}^=\mathbf{v} \leq \mathbf{b}^=)$.

Let $(\mathbf{v}_0, \mathbf{u}_0)$ be the inner point as from Lemma A.4 applied to c'. Then there exists $0 < \lambda \leq 1$ such that:

$$\text{for every } \ 0 < \lambda_0 \leq \lambda, \begin{pmatrix} \lambda_0\mathbf{w}^0 + (1-\lambda_0)\mathbf{v}_0 \\ \lambda_0\mathbf{u} + (1-\lambda_0)\mathbf{u}_0 \end{pmatrix} \in \mathit{Sol}(c').$$

Analogously, there exists $0 < \lambda' \leq 1$ such that:

$$\text{for every } \ 0 < \lambda_0' \leq \lambda', \begin{pmatrix} \lambda_0'\mathbf{w}^1 + (1-\lambda_0')\mathbf{v}_0 \\ \lambda_0'\mathbf{u} + (1-\lambda_0')\mathbf{u}_0 \end{pmatrix} \in \mathit{Sol}(c').$$

For $\lambda_1 = \mathit{min}(\lambda, \lambda') > 0$, we then have:

$$\begin{pmatrix} \lambda_1\mathbf{w}^0 + (1-\lambda_1)\mathbf{v}_0 \\ \lambda_1\mathbf{u} + (1-\lambda_1)\mathbf{u}_0 \end{pmatrix}, \begin{pmatrix} \lambda_1\mathbf{w}^1 + (1-\lambda_1)\mathbf{v}_0 \\ \lambda_1\mathbf{u} + (1-\lambda_1)\mathbf{u}_0 \end{pmatrix} \in \mathit{Sol}(c').$$

Called $\mathbf{u}_1 = \lambda_1\mathbf{u} + (1-\lambda_1)\mathbf{u}_0$, $\mathbf{z}^0 = \lambda_1\mathbf{w}^0 + (1-\lambda_1)\mathbf{v}_0$ and $\mathbf{z}^1 = \lambda_1\mathbf{w}^1 + (1-\lambda_1)\mathbf{v}_0$, we have that $\mathbf{u}_1$ is an instance of parameters $\mathbf{a}$ such that $\mathbf{z}^0, \mathbf{z}^1 \in \mathit{Sol}(\mathbf{x} = \mathbf{u}_1, \mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c)$. Moreover, $\mathbf{z}_i^0 \neq \mathbf{z}_i^1$ since $\mathbf{w}_i^0 \neq \mathbf{w}_i^1$ and $\lambda_1 > 0$. Summarizing, $\mathbf{d}|_! \vdash c \rightarrow x :!$ is not valid. □

Theorem 3.19 [(IEInfer - soundness and completeness)]

Proof. Working on $\mathbf{d} = \mathbf{d}_1|_!$ at *Step 0* is sound and complete by Theorem 3.12. Also, by Lemma A.1, each variable x in $\mathbf{v}$ can be considered separately.

(Soundness) Step 2 is justified by Lemma 3.18. *Step 5* is justified by Soundness of LPInfer. Finally, *Step 6* is justified by the if-part of Lemma 3.14.

(Completeness) Assume that $\mathbf{d}|_! \vdash c \rightarrow x : \tau$ is valid.

If $x \notin \mathbf{v}_c = vars(c)$ then the output of LPInfer is complete, since validity of $\mathbf{d}_1 \vdash c \rightarrow x : \tau$ is equivalent to check $nf(\mathbf{d}_1) \geq_t x : \tau$, which is dealt with at *Step 3* of LPInfer, as shown in the proof of Theorem 3.7.

If $\tau \neq \Box$ then the output of LPInfer is complete again: for $\tau = !$ there cannot be any more general answer type; for $\Box >_t \tau$ there cannot be any more general answer, otherwise LPInfer would have answered $\tau = \Box$.

If $x \in \mathbf{v}_c$ and $\tau = \Box$ the test at *Step 6 (a,b)* is complete by the only-if part of Lemma 3.14. □

We conclude by showing how to compute the set of implicit equalities starting from the Minkowski's form of a polyhedron.

Lemma 3.20

Proof. We have to show that: $\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}\boldsymbol{\gamma}, \boldsymbol{\lambda}, \boldsymbol{\gamma} \geq \mathbf{0}, \Sigma\boldsymbol{\gamma} = \mathbf{1}^T\boldsymbol{\gamma} = 1\} = \{b\}$ iff $\mathbf{c}^T\mathbf{R} = \mathbf{0}$ and $\mathbf{c}^T\mathbf{V} = b\mathbf{1}^T$.

The if part easily follows since $\mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} + \mathbf{c}^T\mathbf{V}\boldsymbol{\gamma} = b\mathbf{1}^T\boldsymbol{\gamma} = b$. We show the only-if part by contraposition.

If $\mathbf{c}^T\mathbf{R} \neq \mathbf{0}$ then for some column $\boldsymbol{R}_i$, $\mathbf{c}^T\mathbf{R}_i \neq 0$. By fixing any $\boldsymbol{\gamma} = \mathbf{0}$ except for $\boldsymbol{\gamma}_1 = 1$, and $\boldsymbol{\lambda} = \mathbf{0}$ except for the i^{th} element, we have $\mathbf{c}^T\mathbf{x} = \mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} + \mathbf{c}^T\mathbf{V}\boldsymbol{\gamma} = \mathbf{c}^T\mathbf{R}_i\boldsymbol{\lambda}_i + \mathbf{c}^T\mathbf{V}_1$, which can assume infinitely many values, since $\boldsymbol{\lambda}_i$ can be any non-negative number. Hence, $\mathbf{c}^T\mathbf{x}$ cannot constantly be equal to b.

If $\mathbf{c}^T\mathbf{R} = \mathbf{0}$ and $\mathbf{c}^T\mathbf{V} \neq b\mathbf{1}^T$, then for some column $\mathbf{V}_i$, $\mathbf{c}^T\mathbf{V}_i \neq b$. By fixing $\boldsymbol{\gamma} = \mathbf{0}$ except for $\boldsymbol{\gamma}_i = 1$, we have $\mathbf{c}^T\mathbf{x} = \mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} + \mathbf{c}^T\mathbf{V}\boldsymbol{\gamma} = \mathbf{c}^T\mathbf{V}_i \neq b$. □

A.5 Section 3.4

Theorem 3.22 [(Definiteness II)].

Proof. The if-part is immediate by Lemma 2.8 since $\mathbf{d} \geq_t \mathbf{d}|_{\mathcal{B}}$.

Consider the only-if part. If c is unsatisfiable, the conclusion readily follows. Also, we can assume that $\mathbf{d}$ is in normal form. In fact, let $\mathbf{d} \vdash c \rightarrow x : \Box_r$ be valid. By Lemma A.1, $nf(\mathbf{d}) \vdash c \rightarrow x : \Box_r$ is valid as well. If we could show the conclusion for normal forms, we get that $nf(\mathbf{d})|_{\mathcal{B}} \vdash c \rightarrow x : \Box_r$ is valid. Finally, due to the structure of $\mathcal{B}$, we observe that $\mathbf{d}|_{\mathcal{B}} \geq_t nf(\mathbf{d})|_{\mathcal{B}}$ holds, and then, by Lemma 2.8 again, we conclude that $\mathbf{d}|_{\mathcal{B}} \vdash c \rightarrow x : \Box_r$ is valid.

Let $\mathbf{v} = vars(c) \cup vars(\mathbf{d}) \cup \{x\}$, and let j such that $\mathbf{v}_i = x$. Since the order of atd's in $\mathbf{d}$ is irrelevant for validity, we can assume that $\mathbf{d} = \mathbf{d}_1, \mathbf{d}|_{\mathcal{B}}$, with $\mathbf{d}_1$ assigning no variable to types in $\mathcal{B}$. We reason by induction on the length of $\mathbf{d}_1$. When $\mathbf{d}_1$ is the empty sequence, the conclusion is immediate since $\mathbf{d} = \mathbf{d}|_{\mathcal{B}}$. Assume now that $\mathbf{d}_1 = y : \tau, \mathbf{d}_2$, with $\tau \notin \mathcal{B}$, and let j such that $\mathbf{v}_j = y$. We will show that $\mathbf{d}_2, \mathbf{d}|_{\mathcal{B}} \vdash c \rightarrow x : \Box_r$ is valid, and then, by inductive hypothesis, we have that $\mathbf{d}|_{\mathcal{B}} \vdash c \rightarrow x : \Box_r$ is valid. Let us consider the possible cases for τ.

$(\tau = \star)$. This cannot occur, since $\mathbf{d}$ is in normal form.

$(\tau = \sqcap)$. Assume, by absurd, that $\mathbf{d}_2, \mathbf{d}|_{\mathcal{B}} \vdash c \to x : \square_r$ is not valid. Called $\overline{\mathbf{d}} = \mathbf{d}_2, \mathbf{d}|_{\mathcal{B}}$, there exists an instance $\mathbf{u}$ of parameters $\mathbf{a} = v(\overline{\mathbf{d}})$ of the parameterized system $\phi(\overline{\mathbf{d}}) \wedge c$ such that the instantiated system has at least two solution points $\mathbf{w}$ and $\mathbf{z}$ which differ on the value of x by at least r, i.e., in formulas:

$$\mathbf{A}_c\mathbf{w} \leq \mathbf{b}_c, \mathbf{A}_{\overline{\mathbf{d}}}\mathbf{w} \leq \mathbf{B}_{\overline{\mathbf{d}}}\mathbf{u} \quad \text{and} \quad \mathbf{A}_c\mathbf{z} \leq \mathbf{b}_c, \mathbf{A}_{\overline{\mathbf{d}}}\mathbf{z} \leq \mathbf{B}_{\overline{\mathbf{d}}}\mathbf{u},$$

and $abs(\mathbf{w}_i - \mathbf{z}_i) > r$. Let $m = max\{\mathbf{w}_j, \mathbf{z}_j\}$. We have: $\mathbf{A}_c\mathbf{w} \leq \mathbf{b}_c, \mathbf{A}_{\overline{\mathbf{d}}}\mathbf{w} \leq \mathbf{B}_{\overline{\mathbf{d}}}\mathbf{u}, \mathbf{w}_j \leq m$ and $\mathbf{A}_c\mathbf{z} \leq \mathbf{b}_c, \mathbf{A}_{\overline{\mathbf{d}}}\mathbf{z} \leq \mathbf{B}_{\overline{\mathbf{d}}}\mathbf{u}, \mathbf{z}_j \leq m$, where $\mathbf{w}_j \leq m$ and $\mathbf{z}_j \leq m$ correspond to the instantiation of $\phi(y : \sqcap)$. Therefore, there exist a parameter instance $(\mathbf{u}\ \ m)$ in the parameter space augmented by one dimension (needed since y is not typed in $\overline{\mathbf{d}}$ as $\mathbf{d}$ is in normal form) such that $\phi(\mathbf{d}) \wedge c$ admits two solutions $\mathbf{w}$ and $\mathbf{z}$ with $abs(\mathbf{w}_i - \mathbf{z}_i) > r$. Summarizing, $\mathbf{d} \vdash c \to x : \square_r$ is not valid, contrarily to the hypothesis.

$(\tau = \sqcup)$. Analogous to $\tau = \sqcap$, but considering $m = min\{\mathbf{w}_j, \mathbf{z}_j\}$.

$(\tau = \square)$. Analogous to $\tau = \sqcap$, but now consider both an upper bound $m_1 = max\{\mathbf{w}_j, \mathbf{z}_j\}$ and a lower bound $m_2 = min\{\mathbf{w}_j, \mathbf{z}_j\}$. $\square$

The next results characterizes the $\simeq_r$ relation in terms of the Minkowski's form of a polyhedron.

Lemma 3.27.

Proof. Notice that since $Sol(\mathbf{Ax} \leq \mathbf{b}) \neq \emptyset$, its generating and vertex matrices $\mathbf{R}$ and $\mathbf{V}$ exist. Let us set $\mathbf{c} = \mathbf{c}_1 - \mathbf{c}_2$ and $\alpha = \alpha_1 - \alpha_2$. It is immediate to observe that $\mathbf{c}_1^T\mathbf{x} + \alpha_1 \simeq_r \mathbf{c}_2^T\mathbf{x} + \alpha_2$ iff $\mathbf{c}^T\mathbf{x} + \alpha \simeq_r 0$.

Consider the if-part. For every $\mathbf{x}_0$ such that $\mathbf{Ax}_0 \leq \mathbf{b}$ holds, there exists $\boldsymbol{\lambda}, \boldsymbol{\mu} \geq \mathbf{0}$ with $\Sigma\boldsymbol{\mu} = 1$ such that $\mathbf{x}_0 = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}\boldsymbol{\mu}$. We calculate: $abs(\mathbf{c}^T\mathbf{x}_0 + \alpha) = abs(\mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} + \mathbf{c}^T\mathbf{V}\boldsymbol{\mu} + \alpha) = abs(\mathbf{c}^T\mathbf{V}\boldsymbol{\mu} + \alpha\mathbf{1}^T\boldsymbol{\mu}) = abs((\mathbf{c}^T\mathbf{V} + \alpha\mathbf{1}^T)\boldsymbol{\mu})$, since $\mathbf{c}^T\mathbf{R} = \mathbf{0}$ and $\mathbf{1}^T\boldsymbol{\mu} = \Sigma\boldsymbol{\mu} = 1$. Since $\mathbf{1} \geq \boldsymbol{\mu}$, we have $abs(\mathbf{c}^T\mathbf{x}_0 + \alpha) = abs((\mathbf{c}^T\mathbf{V} + \alpha\mathbf{1}^T)\boldsymbol{\mu}) \leq \|\mathbf{c}^T\mathbf{V} + \alpha\mathbf{1}^T\|_\infty = r$. Moreover, by fixing $\boldsymbol{\mu} = \mathbf{0}$ except for $\boldsymbol{\mu}_i = 1$, where i is such that $abs(\mathbf{c}^T\mathbf{V}_i + \alpha\mathbf{1}_i^T) = r$, we obtain an $\mathbf{x}_0 = \mathbf{V}\boldsymbol{\mu}$ such that $abs(\mathbf{c}^T\mathbf{x}_0 + \alpha) = abs((\mathbf{c}^T\mathbf{V} + \alpha\mathbf{1}^T)\boldsymbol{\mu}) = abs(\mathbf{c}^T\mathbf{V}_i + \alpha\mathbf{1}_i^T) = r$.

Consider now the only-if part. Let $\mathbf{V}_k$ be any column of $\mathbf{V}$. Hence, $\mathbf{V}_k \in Sol(\mathbf{Ax} \leq \mathbf{b})$ and then $abs(\mathbf{c}^T\mathbf{V}_k + \alpha) \leq r$.

If $\mathbf{c}^T\mathbf{R} \neq \mathbf{0}$ then there exists a column $\mathbf{R}_j$ of $\mathbf{R}$ such that $\mathbf{c}^T\mathbf{R}_j \neq 0$. By choosing $\boldsymbol{\lambda}_i = 0$ for $i \neq j$, $\boldsymbol{\mu}_i = 0$ for $i \neq k$ and $\boldsymbol{\mu}_k = 1$, for $\mathbf{x}_0 = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}\boldsymbol{\mu}$ we get $\mathbf{c}^T\mathbf{x}_0 + \alpha = \mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} + \mathbf{c}^T\mathbf{V}\boldsymbol{\mu} + \alpha = \mathbf{c}^T\mathbf{R}_j\boldsymbol{\lambda}_j + \mathbf{c}^T\mathbf{V}_k + \alpha \not\simeq_r 0$, since $\mathbf{c}^T\mathbf{R}_j \neq 0$ and $\boldsymbol{\lambda}_j$ can be any positive real number.

If $\mathbf{c}^T\mathbf{R} = \mathbf{0}$ and $\|\mathbf{c}^T\mathbf{V} + \alpha\mathbf{1}^T\|_\infty \neq r$, we distinguish two cases. If for some i, $abs(\mathbf{c}^T\mathbf{V}_i + \alpha\mathbf{1}_i^T) > r$, then $\mathbf{c}^T\mathbf{x} + \alpha \not\simeq_r 0$ over $\mathbf{Ax} \leq \mathbf{b}$ since every vertex $\mathbf{V}_i$ is in $Sol(\mathbf{Ax} \leq \mathbf{b})$. If for all i, $abs(\mathbf{c}^T\mathbf{V}_i + \alpha\mathbf{1}_i^T) < r$, then there cannot exist $\mathbf{x}_0 \in Sol(\mathbf{Ax} \leq \mathbf{b})$ such that $abs(\mathbf{c}^T\mathbf{x}_0 + \alpha) = r$. In fact, since $\mathbf{c}^T\mathbf{R} = \mathbf{0}$, such an $\mathbf{x}_0$ can be rewritten as $\mathbf{x}_0 = \mathbf{V}\boldsymbol{\mu}$ for some $\boldsymbol{\mu} \geq \mathbf{0}$ with $\Sigma\boldsymbol{\mu} = 1$. Then, $abs(\mathbf{c}^T\mathbf{x}_0 + \alpha) = abs(\mathbf{c}^T\mathbf{V}\boldsymbol{\mu} + \alpha) \leq max\{abs(\mathbf{c}^T\mathbf{V}_i + \alpha\mathbf{1}_i^T)\} < r$. $\square$

The next result states that the range width of a linear function over the solutions of a polytope (a polyhedron whose generating matrix is empty), can be computed as the maximum range width between two vertices.

Lemma A.5. *Let $\alpha \geq 0$, and $\mathbf{V}$ be a matrix of k columns such that for $1 \leq m < n \leq k$, $abs(\mathbf{c}^T\mathbf{V}_m - \mathbf{c}^T\mathbf{V}_n) \leq \alpha$. Then for every $\boldsymbol{\mu}' \geq \mathbf{0}, \Sigma\boldsymbol{\mu}' = 1, \boldsymbol{\mu}'' \geq \mathbf{0}, \Sigma\boldsymbol{\mu}'' = 1$, we have $abs(\mathbf{c}^T\mathbf{V}(\boldsymbol{\mu}' - \boldsymbol{\mu}'')) \leq \alpha$.*

Proof. Without any loss of generality, we can assume $abs(\mathbf{c}^T\mathbf{V}(\boldsymbol{\mu}' - \boldsymbol{\mu}'')) = \mathbf{c}^T\mathbf{V}(\boldsymbol{\mu}' - \boldsymbol{\mu}'')$ (otherwise, simply swap $\boldsymbol{\mu}'$ and $\boldsymbol{\mu}''$).

When $k = 1$, we have $\boldsymbol{\mu}' = \boldsymbol{\mu}'' = \mathbf{1}$ and then $abs(\mathbf{c}^T\mathbf{V}(\boldsymbol{\mu}' - \boldsymbol{\mu}'')) = 0 \leq \alpha$.

When $k > 1$, we reason by contraposition. Assume $\mathbf{c}^T\mathbf{V}(\boldsymbol{\mu}' - \boldsymbol{\mu}'') > \alpha$ for some $\boldsymbol{\mu}', \boldsymbol{\mu}''$. Let m be such that $\mathbf{c}^T\mathbf{V}_m$ is maximum, and n be such that $\mathbf{c}^T\mathbf{V}_n$ is minimum. We have $\alpha < \mathbf{c}^T\mathbf{V}(\boldsymbol{\mu}' - \boldsymbol{\mu}'') = \mathbf{c}^T\mathbf{V}\boldsymbol{\mu}' - \mathbf{c}^T\mathbf{V}\boldsymbol{\mu}'' \leq \mathbf{c}^T\mathbf{V}_m - \mathbf{c}^T\mathbf{V}_n \leq abs(\mathbf{c}^T\mathbf{V}_m - \mathbf{c}^T\mathbf{V}_n) = abs(\mathbf{c}^T\mathbf{V}_n - \mathbf{c}^T\mathbf{V}_m)$. This show our conclusion for some $m \leq n$. Moreover, we claim that $m \neq n$. Otherwise we have $\mathbf{c}^T\mathbf{V}_i = \mathbf{c}^T\mathbf{V}_j$ for every $1 \leq i, j \leq k$, and then we can use any other index pair i, j such that $i < j$. □

Lemma 3.28.

Proof. For a parameter instance $\mathbf{u}$ let us denote by $\mathcal{V}^{\mathbf{u}}$ the set of (instances of the) parameterized vertices whose domain includes $\mathbf{u}$, namely: $\mathcal{V}^{\mathbf{u}} = \{\mathbf{v}^{\mathbf{u}}(i) \mid i = 1..k, \mathbf{C}_i\mathbf{u} \leq \mathbf{c}_i\ \}$. If $\mathcal{V}^{\mathbf{u}} \neq \emptyset$, let us denote by $\mathbf{V}^{\mathbf{u}}$ a matrix whose columns are the vectors in $\mathcal{V}^{\mathbf{u}}$.

(If-part). Notice that the hypotheses imply that $r \geq 0$. We first show the for-all statement. Let $\mathbf{u}$ be fixed. If $\mathcal{S}_{\mathbf{u}} = \emptyset$, we are done. Otherwise, by Theorem 3.24, $\mathcal{V}^{\mathbf{u}}$ is not empty. Since $\mathbf{c}^T\mathbf{R} = \mathbf{0}$, we calculate:

$$\begin{aligned}\mathcal{S}_{\mathbf{u}} &= \{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}^{\mathbf{u}}\boldsymbol{\mu}, \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\}\\ &= \{z \mid z = \mathbf{c}^T\mathbf{V}^{\mathbf{u}}\boldsymbol{\mu}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\}.\end{aligned}$$

Moreover, since $\mathbf{u}$ belongs to the domains of all the vertices in $\mathbf{V}^{\mathbf{u}}$, for every pair $\mathbf{v}^{\mathbf{u}}(m), \mathbf{v}^{\mathbf{u}}(n)$ of them $Sol(P_{m,n}) \neq \emptyset$. By hypothesis, we then have:

$$abs(\mathbf{c}^T\mathbf{v}^{\mathbf{u}}(m) - \mathbf{c}^T\mathbf{v}^{\mathbf{u}}(n)) \leq r. \tag{5}$$

For $x, y \in \mathcal{S}_{\mathbf{u}}$, $abs(x - y)$ can be rewritten as $abs(\mathbf{c}^T\mathbf{V}^{\mathbf{u}}(\boldsymbol{\mu}_x - \boldsymbol{\mu}_y))$, for some $\boldsymbol{\mu}_x \geq \mathbf{0}, \Sigma\boldsymbol{\mu}_x = 1, \boldsymbol{\mu}_y \geq \mathbf{0}, \Sigma\boldsymbol{\mu}_y = 1$. By Lemma A.5, this and (5) imply $abs(x - y) \leq r$.

Next, let us show the existential statement.

If $r = 0$ the conclusion holds by considering any $\mathbf{u}$ such that $\mathcal{S}_{\mathbf{u}} \neq \emptyset$ (at least one exists, since the parameterized polyhedron is assumed not to be empty). For any $x \in \mathcal{S}_{\mathbf{u}}$, we have $abs(x - x) = 0$.

If $r > 0$, by assumption there exists m, n be such that $\mathbf{c}^T\mathbf{v}^{\mathbf{a}}(m) \simeq_r \mathbf{c}^T\mathbf{v}^{\mathbf{a}}(n)$ over $P_{m,n}$, with $Sol(P_{m,n}) \neq \emptyset$. By Definition 3.25, there exists $\mathbf{u} \in Sol(P_{m,n})$ such that $abs(\mathbf{c}^T\mathbf{v}^{\mathbf{u}}(m) - \mathbf{c}^T\mathbf{v}^{\mathbf{u}}(n)) = r$. The conclusion then follows by noting that $x = \mathbf{c}^T\mathbf{v}^{\mathbf{u}}(m)$ and $y = \mathbf{c}^T\mathbf{v}^{\mathbf{u}}(n)$ belong to $\mathcal{S}_{\mathbf{u}}$ since $\mathbf{v}^{\mathbf{u}}(m)$ and $\mathbf{v}^{\mathbf{u}}(n)$ are in $\mathbf{V}^{\mathbf{u}}$.

(Only-If part). We show the contrapositive of the three conclusions.

First, if $\mathbf{c}^T\mathbf{R} \neq \mathbf{0}$, consider any $\mathbf{u}$ in $Sol(\mathbf{C}_1\mathbf{a} \leq \mathbf{c}_1)$. Notice that $k \geq 1$ since the parameterized polyhedron is assumed not to be empty, and $Sol(\mathbf{C}_1\mathbf{a} \leq \mathbf{c}_1) \neq \emptyset$ by Theorem 3.24. We have:

$$\begin{aligned}\mathcal{S}_{\mathbf{u}} &= \{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}^{\mathbf{u}}\boldsymbol{\mu}, \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\}\\ &= \{z \mid z = \mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} + \mathbf{c}^T\mathbf{V}^{\mathbf{u}}\boldsymbol{\mu}, \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\}.\end{aligned}$$

Let $\mathbf{R}_j$ be a column of $\mathbf{R}$ such that $\mathbf{c}^T\mathbf{R}_j = \alpha \neq 0$. By fixing $\boldsymbol{\lambda}_i = 0$ for $i \neq j$, we get that $\mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} = \alpha\boldsymbol{\lambda}_j$. By fixing $\boldsymbol{\mu}$ in any way such that $\boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1$, we also have $\mathbf{c}^T\mathbf{V}^{\mathbf{u}}\boldsymbol{\mu} = \beta$, for some β. Summarizing, $\mathcal{S}_{\mathbf{u}}$ includes $\alpha\boldsymbol{\lambda}_j + \beta$ for every $\boldsymbol{\lambda}_j \geq 0$. Hence, $abs(x - y)$ cannot be bounded by any r for every $x, y \in \mathcal{S}_{\mathbf{u}}$.

Second, assume that there exist $1 \leq m < n \leq k$ such that $Sol(P_{m,n}) \neq \emptyset$ and for every $s \leq r$, $\mathbf{c}^T\mathbf{v}^{\mathbf{a}}(m) \not\simeq_s \mathbf{c}^T\mathbf{v}^{\mathbf{a}}(n)$ over $P_{m,n}$. This implies that there exists $\mathbf{u}$ in $Sol(P_{m,n})$ such that $abs(\mathbf{c}^T\mathbf{v}^{\mathbf{u}}(m) - \mathbf{c}^T\mathbf{v}^{\mathbf{u}}(n)) > r$. By fixing $\boldsymbol{\mu}_i = 0$ for $i \neq m$ and $\boldsymbol{\mu}_m = 1$, we define $\mathbf{x} = \mathbf{V}^{\mathbf{u}}\boldsymbol{\mu} = \mathbf{v}^{\mathbf{u}}(m)$. Analogously, by fixing $\boldsymbol{\mu}_i = 0$ for $i \neq n$ and $\boldsymbol{\mu}_n = 1$, we define $\mathbf{y} = \mathbf{V}^{\mathbf{u}}\boldsymbol{\mu} = \mathbf{v}^{\mathbf{u}}(n)$. It turns out $x = \mathbf{c}^T\mathbf{x}, y = \mathbf{c}^T\mathbf{y}$ belong to:

$$\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} = \mathbf{V}^{\mathbf{u}}\boldsymbol{\mu}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\} = \{z \mid z = \mathbf{c}^T\mathbf{V}^{\mathbf{u}}\boldsymbol{\mu}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\} \subseteq \mathcal{S}_{\mathbf{u}},$$

and $abs(x - y) > r$.

Third, we have to show that if $r \neq \hat{r}$, where:

$$\begin{aligned}\hat{r} = max(\{0\} \cup \{s \mid \ & 1 \leq m < n \leq k, Sol(P_{m,n}) \neq \emptyset, \\ & \mathbf{c}^T\mathbf{v}^{\mathbf{a}}(m) \simeq_s \mathbf{c}^T\mathbf{v}^{\mathbf{a}}(n) \text{ over } P_{m,n}\}),\end{aligned}$$

then: *(a)* for some $\mathbf{u}$, and some $x, y \in \mathcal{S}_{\mathbf{u}}$, $abs(x - y) > r$; or *(b)* for every $\mathbf{u}$, and every $x, y \in \mathcal{S}_{\mathbf{u}}$, $abs(x-y) < r$. Without any loss of generality, we can assume that for $1 \leq m < n \leq k$, $Sol(P_{m,n}) = \emptyset$ or $\mathbf{c}^T\mathbf{v}^{\mathbf{a}}(m) \simeq_s \mathbf{c}^T\mathbf{v}^{\mathbf{a}}(n)$ over $P_{m,n}$ for some $s \leq r$ (otherwise, we can apply the previous part of the proof to obtain *(a)*). This trivially implies $\hat{r} \in \mathcal{R}$ and $r \geq \hat{r}$. Therefore, we can assume $r > \hat{r}$. We will show *(b)* by absurd. Assume that *(b)* does not hold, namely there exist $\mathbf{u}$ and $x, y \in \mathcal{S}_{\mathbf{u}}$, $abs(x - y) \geq r > \hat{r}$. Since we can assume $\mathbf{c}^T\mathbf{R} = \mathbf{0}$ (already shown), we calculate:

$$\begin{aligned}\mathcal{S}_{\mathbf{u}} &= \{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}^{\mathbf{u}}\boldsymbol{\mu}, \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\} \\ &= \{z \mid z = \mathbf{c}^T\mathbf{V}^{\mathbf{u}}\boldsymbol{\mu}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\}.\end{aligned}$$

Thus, $abs(x - y) > \hat{r}$ can be rewritten as $abs(\mathbf{c}^T\mathbf{V}^{\mathbf{u}}(\boldsymbol{\mu}_x - \boldsymbol{\mu}_y)) > \hat{r}$, for some $\boldsymbol{\mu}_x \geq \mathbf{0}, \Sigma\boldsymbol{\mu}_x = 1, \boldsymbol{\mu}_y \geq \mathbf{0}, \Sigma\boldsymbol{\mu}_y = 1$. By Lemma A.5, there exist m', n' such that $abs(\mathbf{c}^T\mathbf{V}^{\mathbf{u}}_{m'} - \mathbf{c}^T\mathbf{V}^{\mathbf{u}}_{n'}) > \hat{r}$. For some m, the column $\mathbf{V}^{\mathbf{u}}_{m'}$ corresponds to an instance of a parameterized vertex $\mathbf{v}^{\mathbf{u}}(m)$, i.e., $\mathbf{V}^{\mathbf{u}}_{m'} = \mathbf{v}^{\mathbf{u}}(m)$. Analogously, for some n, $\mathbf{V}^{\mathbf{u}}_{n'} = \mathbf{v}^{\mathbf{u}}(n)$. Also, we can assume $m < n$ since we are considering absolute values.

Therefore, we can conclude that there exist $1 \leq m < n \leq k$ such that $Sol(P_{m,n}) \neq \emptyset$, since $\mathbf{u} \in Sol(P_{m,n})$, and such that $abs(\mathbf{c}^T\mathbf{v}^{\mathbf{u}}(m) - \mathbf{c}^T\mathbf{v}^{\mathbf{u}}(n)) > \hat{r}$. This is impossible by definition of $\hat{r}$. $\square$

THEOREM 3.30 [(POLYINFER - SOUNDNESS AND COMPLETENESS)].

PROOF. Working on $\mathbf{n} = nf(\mathbf{d}_1)$ at *Step 0* is sound and complete by Lemma A.1; while working on $\mathbf{d} = \mathbf{n}|_{\mathcal{B}}$ is sound and complete by Theorem 3.22. Also, by Lemma A.1, each variable x in $\mathbf{v}$ can be considered separately. Finally, we recall that Lemma 3.28 provides necessary and sufficient conditions for computing the minimum r such that $\mathbf{d} \vdash c \to x : \Box_r$ is valid.

(Soundness) Step 4 is justified by Soundness of LPINFER.

Step 5 is justified by the if-part of Lemma 3.28 instantiated by fixing $\mathbf{c}_j = 0$ for $j \neq i$, and $\mathbf{c}_i = 1$, so that $\mathbf{c}^T\mathbf{v}_c = x$. Notice that, since $r \geq 0$ by its definition, $\Box_r$ is a type (in $\mathcal{BT}$).

(Completeness) Assume that $\mathbf{d} \vdash c \to x : \tau$ is valid.

If $x \notin \mathbf{v}_c = vars(c)$ then the output of LPInfer is complete, since validity of $\mathbf{d}_1 \vdash c \rightarrow x : \tau$ is equivalent to check $nf(\mathbf{d}_1) \geq_t x : \tau$. This is dealt with at *Step 3* of LPInfer, as shown in the proof of Theorem 3.7.

If $\tau \neq \Box$ then the output of LPInfer is complete again: for $\tau = !$ there cannot be any more general answer type; for $\Box >_t \tau$ there cannot be any more general answer, otherwise LPInfer would have answered $\tau = \Box$.

If $x \in \mathbf{v}_c$ and $\tau = \Box$ the test at *Step 5 (a,b)* is complete by the only-if part of Lemma 3.28. $\Box$

A.6 Section 3.5

Next lemma states that if we are given a point $\mathbf{x}$ in an hyperplane and a point $\mathbf{y}$ outside it, the segment from $\mathbf{x}$ to $\mathbf{y}$ lies outside the hyperplane, except for $\mathbf{x}$.

Lemma A.6. *Let* $\mathbf{c}^T\mathbf{x} = \alpha$ *and* $\mathbf{c}^T\mathbf{y} \neq \alpha$. *For every* $0 \leq \lambda < 1$, *called* $\mathbf{z} = \lambda\mathbf{x} + (1-\lambda)\mathbf{y}$, *we have* $\mathbf{c}^T\mathbf{z} \neq \alpha$.

Proof. We have $\mathbf{c}^T\mathbf{z} > \alpha$ iff $\lambda\mathbf{c}^T\mathbf{x} + (1-\lambda)\mathbf{c}^T\mathbf{y} > \alpha$ iff $\lambda\alpha + (1-\lambda)\mathbf{c}^T\mathbf{y} > \alpha$ iff $(1-\lambda)\mathbf{c}^T\mathbf{y} > \alpha(1-\lambda)$ and finally, since $\lambda < 1$, iff $\mathbf{c}^T\mathbf{y} > \alpha$. The case of $\mathbf{c}^T\mathbf{z} < \alpha$ is analogous. $\Box$

Let us provide a set oriented characterization of generalized constraints.

Lemma A.7. *Let* $g = c \wedge \bigwedge_{i=1}^m e_i \neq \alpha_i$ *be a generalized linear constraint. We have:*

$$Sol(g) = \bigcap_{i=1}^m Sol(c) \setminus Sol(e_i = \alpha_i).$$

Proof.

$$\begin{aligned} Sol(g) &= Sol(\bigwedge_{i=1}^m (c \wedge e_i \neq \alpha_i)) = \bigcap_{i=1}^m Sol(c \wedge e_i \neq \alpha_i) \\ &= \bigcap_{i=1}^m Sol(c) \cap Sol(e_i \neq \alpha_i) = \bigcap_{i=1}^m Sol(c) \setminus Sol(e_i = \alpha_i). \end{aligned}$$

$\Box$

The next result can be interpreted geometrically as follows. Given a point $\mathbf{x}$ in the solution set of a generalized linear constraints $c \wedge \bigwedge_{i=1}^m e_i \neq \alpha_i$, and a point $\mathbf{y}$ in the polyhedron $Sol(c)$, the hyperplanes $e_i = \alpha_i$ intersect the segment from $\mathbf{x}$ to $\mathbf{y}$ in finitely many points.

Lemma A.8. *Let* $g = c \wedge \bigwedge_{i=1}^m e_i \neq \alpha_i$ *be a generalized linear constraint. For every* $\mathbf{x} \in Sol(g)$ *and* $\mathbf{y} \in Sol(c)$ *there exists* $0 < \lambda < 1$ *such that, for every* $0 < \lambda_0 \leq \lambda$, $\lambda_0\mathbf{x} + (1-\lambda_0)\mathbf{y} \in Sol(g)$.

Proof. The proof is by induction on m. For $m = 0$ the result is immediate for any $0 < \lambda < 1$ since $Sol(g) = Sol(c)$ is a convex set. Consider the inductive step.

If for every $0 < \lambda_0 < 1$, $\lambda_0\mathbf{x} + (1-\lambda_0)\mathbf{y} \in Sol(g)$ then we are done with any $0 < \lambda < 1$. Assume then some $0 < \lambda_1 < 1$ such that $\mathbf{y}_1 = \lambda_1\mathbf{x} + (1-\lambda_1)\mathbf{y} \notin Sol(g)$. Let J be the set of all $j \in \{1, \ldots, m\}$ such that $\mathbf{y}_1 \in Sol(e_j = \alpha_j)$. By Lemma A.7, $J \neq \emptyset$. We observe that $\mathbf{y} \notin Sol(e_j = \alpha_j)$ for every $j \in J$. Otherwise, since $\mathbf{x}$ is a linear combination of $\mathbf{y}$ and $\mathbf{y}_1$ (namely, $\mathbf{x} = \mathbf{y}_1/\lambda_1 - (1-\lambda_1)/\lambda_1\mathbf{y}$), it would belong to some $Sol(e_j = \alpha_j)$ with $j \in J$. But this is impossible since $\mathbf{x} \in Sol(g)$ implies, by Lemma A.7, $\mathbf{x} \notin Sol(e_j = \alpha_j)$ for $j = 1..m$.

By Lemma A.6 applied to $\mathbf{y}_1$ and $\mathbf{y}$, we have that for every $0 \leq \lambda_0 < 1$ and $j \in J$, $\mathbf{z} = \lambda_0\mathbf{y}_1 + (1-\lambda_0)\mathbf{y} \notin Sol(e_j = \alpha_j)$, i.e., the segment from $\mathbf{y}_1$ to $\mathbf{y}$ does not belong to $Sol(e_j = \alpha_j)$, except for $\mathbf{y}_1$.

By inductive hypothesis on $g_1 = c \wedge \bigwedge_{i\in\{1,\ldots,m\}\setminus J}\ e_i \neq \alpha_i$, $\mathbf{y}_1$ (which is in $Sol(g_1)$) and $\mathbf{y}$ (which is in $Sol(c)$), there exists $0 < \lambda_2 < 1$ such that, for every $0 < \lambda_0 \leq \lambda_2$, $\mathbf{z} = \lambda_0\mathbf{y}_1 + (1-\lambda_0)\mathbf{y} \in Sol(g_1)$. In addition, we have shown that $\mathbf{z} \notin Sol(e_j = \alpha_j)$, for $j \in J$. Therefore, by Lemma A.7, $\mathbf{z} \in Sol(g)$. The conclusion now follows by obvserving that $\mathbf{z}$ can be rewritten as $\lambda_0\mathbf{x}+(1-\lambda_0)\mathbf{y}$ for $0 < \lambda_0 \leq \lambda$, where $\lambda = \lambda_1\lambda_2$. □

Since $Sol(g)$ can be an open set, we write $sup\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in Sol(g)\}$ to denote the supremum, instead of $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in Sol(g)\}$ to denote the maximum.

Lemma A.9. *Let $g = c \wedge \bigwedge_{i=1}^m e_i \neq \alpha_i$ be a satisfiable generalized linear constraint. We have: $sup\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in Sol(g)\} \in \mathcal{R}$ iff $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in Sol(c)\} \in \mathcal{R}$.*

Proof. The if part follows since $\emptyset \subset Sol(g) \subseteq Sol(c)$. Consider now the only-if part. Let $\alpha = sup\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in Sol(g)\} \in \mathcal{R}$. Since g is satisfiable, there exists $\mathbf{x}_0 \in Sol(g)$, and $\alpha \geq \mathbf{c}^T\mathbf{x}_0$. Since $Sol(g) \subseteq Sol(c)$, we have that $\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in Sol(c)\} \neq \emptyset$. We claim that $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in Sol(c)\} \leq \alpha$, which yields the conclusion. Assume, by absurd, that there exists $\mathbf{x}_1 \in Sol(c)$ such that $\mathbf{c}^T\mathbf{x}_1 = \beta > \alpha \geq \mathbf{c}^T\mathbf{x}_0$. By Lemma A.8, there exists $0 < \lambda < 1$ such that $\mathbf{z} = \lambda\mathbf{x}_0 + (1-\lambda)\mathbf{x}_1 \in Sol(g)$ for every $0 < \lambda_0 \leq \lambda$. For $\lambda_0 = min\{\lambda, (\beta/2 - \alpha/2)/(\beta - \mathbf{c}^T\mathbf{x}_0)\}$, we have: $\mathbf{c}^T\mathbf{z} = \lambda_0\mathbf{c}^T\mathbf{x}_0 + (1-\lambda_0)\mathbf{c}^T\mathbf{x}_1 = \lambda_0(\mathbf{c}^T\mathbf{x}_0 - \beta) + \beta \geq -(\beta/2 - \alpha/2) + \beta = (\beta+\alpha)/2 > \alpha$. But this is impossible by definition of α. □

Theorem 3.31.

Proof. The if-part is immediate: since $\mathcal{R} \models g \rightarrow c$, by reasoning as in the proof of Lemma 2.8, which does not rely on linearity of the underlying constraints, we have the conclusion. Consider now the only-if part. The proof is by induction on the length of $\mathbf{d}_2$. When it is empty, there is nothing to be shown. Assume now $\mathbf{d}_2 = x : \tau, \mathbf{d}_3$. By inductive hypothesis, we have $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_3$. Thus we are left with showing $\mathbf{d}_1 \vdash c \rightarrow x : \tau$. Let us consider the possible values for type τ.

($x : \star$) There is nothing to be shown.

($x : \sqcap$) Since g is satisfiable, $g \wedge \phi(\mathbf{d}_1)$ is satisfiable as well in the space of variables plus parameters. Let $[\mathbf{x}_0|\mathbf{u}_0] \in Sol(g \wedge\ \phi(\mathbf{d}_1))$. Since $\mathbf{d}_1 \vdash g \rightarrow x : \sqcap$ is valid, $sup\{x \mid \mathbf{x} \in Sol(g \wedge \phi(\mathbf{d}_1) \wedge v(\mathbf{d}_1) = \mathbf{u}_0)\} \in \mathcal{R}$. By Lemma A.9, we have:

$$max\{x \mid \mathbf{x} \in Sol(c \wedge \phi(\mathbf{d}_1) \wedge v(\mathbf{d}_1) = \mathbf{u}_0)\} \in \mathcal{R}.$$

Since c contains no disequality, this can be rewritten in the space of variables as $max\{x \mid \mathbf{x} \in Sol(c \wedge \phi(\mathbf{d}_1), \mathbf{u}_0)\} \in \mathcal{R}$. We are now in the hypotheses of Theorem A.3 to conclude that $max\{x \mid \mathbf{x} \in Sol(\mathcal{H})\} = 0$, where $\mathcal{H}$ is the homogeneous version of $c \wedge \phi(\mathbf{d}_1)$. By Lemma 3.5, we conclude that $\mathbf{d}_1 \vdash c \rightarrow x : \sqcap$.

($x : \sqcup$) Same reasoning as in ($x : \sqcap$) but maximizing $-x$ instead of x.

($x : \Box$) It reduces to the cases ($x : \sqcap$) and ($x : \sqcup$).

$(x : \Box_r)$ Since g is satisfiable, $g \wedge \phi(\mathbf{d}_1)$ is satisfiable as well in the space of variables plus parameters. Let $[\mathbf{x}_0|\mathbf{u}_0] \in Sol(g \wedge \phi(\mathbf{d}_1))$.

Consider now an instance $\mathbf{u}$ of parameters in $\upsilon(\mathbf{d}_1)$. In order to conclude $\mathbf{d}_1 \vdash c \rightarrow x : \Box_r$, we have to show that for every $y, z \in \{x \mid \mathbf{x} \in Sol(c \wedge \phi(\mathbf{d}_1), \mathbf{u})\}$, $abs(y-z) \leq r$. By absurd, assume that, without any loss of generality, $y - z > r$ for some y, z. Then, there exist $[\mathbf{y}|\mathbf{u}], [\mathbf{z}|\mathbf{u}] \in Sol(c \wedge \phi(\mathbf{d}_1))$ such that the x dimension values of $\mathbf{y}$ and $\mathbf{z}$ are y and z respectively. By Lemma A.8, there exist $0 < \overline{\lambda}_y$ and $0 < \overline{\lambda}_z$ such that for every $0 < \lambda_y \leq \overline{\lambda}_y$ and $0 < \lambda_z \leq \overline{\lambda}_z$:

$$\begin{bmatrix} \lambda_y \mathbf{x}_0 + (1-\lambda_y)\mathbf{y} \\ \lambda_y \mathbf{u}_0 + (1-\lambda_y)\mathbf{u} \end{bmatrix}, \begin{bmatrix} \lambda_z \mathbf{x}_0 + (1-\lambda_z)\mathbf{z} \\ \lambda_z \mathbf{u}_0 + (1-\lambda_z)\mathbf{u} \end{bmatrix} \in Sol(g \wedge \phi(\mathbf{d}_1)).$$

By choosing $\lambda < min\{\overline{\lambda}_y, \overline{\lambda}_z, 1 - r/(y-z)\}$, and called:

$$\mathbf{y}_1 = \lambda \mathbf{x}_0 + (1-\lambda)\mathbf{y}, \quad \mathbf{z}_1 = \lambda \mathbf{x}_0 + (1-\lambda)\mathbf{z}, \quad \mathbf{u}_1 = \lambda \mathbf{u}_0 + (1-\lambda)\mathbf{u},$$

we have: $[\mathbf{y}_1|\mathbf{u}_1], [\mathbf{z}_1|\mathbf{u}_1] \in Sol(g \wedge \phi(\mathbf{d}_1))$. This implies that:

$$\{x \mid \mathbf{x} \in Sol(g \wedge \phi(\mathbf{d}_1), \mathbf{u}_1)\} \supseteq \{y_1, z_1\},$$

where, called x_0 the value of the x dimension of $\mathbf{x}_0$, $y_1 = \lambda x_0 + (1-\lambda)y$ and $z_1 = \lambda x_0 + (1-\lambda)z$. We have that $y_1 - z_1 = (1-\lambda)(y-z) > r$ since $\lambda < 1 - r/(y-z)$. But this is impossible. $\Box$

A.7 Section 3.6

First, we introduce notation for denoting type variables occurring in an atd $x : \Box_e$, and for denoting the primitive constraint $e \geq 0$ which is part of $\phi(x : \Box_e)$.

Definition A.10. *We define the set of type variables occurring in an atd as:* $pvars(x : \Box_e) = vars(e)$ *and* $pvars(x : \tau) = \emptyset$ *for* $\tau \in \mathcal{BT}_\Box$.

Also, we define the constraint on parametric expressions in an atd as: $\Phi(x : \Box_e)$ *is* $e \geq 0$*, and* $\Phi(x : \tau)$ *is* `true` *for* $\tau \in \mathcal{BT}_\Box$.

The notions readily extend to type declarations by defining:

$$pvars(d_1, \ldots, d_n) = \cup_{i=1..n} pvars(d_i) \qquad \Phi(d_1, \ldots, d_n) = \wedge_{i=1..n} \Phi(d_i).$$

Let us show first two results dealing with local type variables.

Lemma 3.36.

Proof. Let us denote $\mathbf{pv} = pvars(\mathbf{d}_1)$. Called s a local type variable in e, and $\mathbf{l}$ the remaining local type variables (possibly none), e can be written as $e = \alpha s + \mathbf{c}_1^T \mathbf{l} + \mathbf{c}_2^T \mathbf{pv} + r$, where $\alpha \neq 0$ and $r \in \mathcal{R}$. Validity of $\mathbf{d}_1 \vdash c \rightarrow x : \Box_e$ consists of showing the formula (1) in Definition 2.4, which can be rewritten as follows:

$$\forall \mathbf{pv} \forall \upsilon(\mathbf{d}_1) \setminus \mathbf{pv} \exists a, s, \mathbf{l}\ \forall \mathbf{v}.(\phi(\mathbf{d}_1) \wedge c) \rightarrow (a \leq x \leq a + e \wedge e \geq 0). \qquad (6)$$

If this is true, by defining $b = a + e$ (all the variables in $\upsilon(\mathbf{d}_1)$ being fixed) the following holds:

$$\forall \mathbf{pv} \forall \upsilon(\mathbf{d}_1) \setminus \mathbf{pv} \exists a, b\ \forall \mathbf{v}.(\phi(\mathbf{d}_1) \wedge c) \rightarrow (a \leq x \leq b),$$

namely $\mathbf{d}_1 \vdash c \rightarrow x : \Box$ is valid. Conversely, if this is true, by defining $\mathbf{l} = \mathbf{0}, s = (b - a - \mathbf{c}_2^T \mathbf{pv} - r)/\alpha$, we have that $e = b - a$ and then (6) holds. $\Box$

Lemma 3.38.

Proof. Let us denote $\mathbf{pv} = pvars(\mathbf{d}_1)$. Validity of $\mathbf{d}_1 \vdash c \rightarrow (\mathbf{d}_2, \mathbf{d}_3)$ consists of showing the formula (1) in Definition 2.4, which can be rewritten as follows:

$$\forall \mathbf{pv} \forall \upsilon(\mathbf{d}_1) \setminus \mathbf{pv} \exists \upsilon(\mathbf{d}_2, \mathbf{d}_3) \setminus \mathbf{pv} \forall \mathbf{v}.(\phi(\mathbf{d}_1) \wedge c) \rightarrow (\phi(\mathbf{d}_2) \wedge \phi(\mathbf{d}_3)).$$

No local type variable means that an existentially quantified variable in $\upsilon(\mathbf{d}_2, \mathbf{d}_3) \setminus \mathbf{pv}$ appears either in $\phi(\mathbf{d}_2)$ or in $\phi(\mathbf{d}_3)$, but not in both. Therefore, the formula above can be rewritten as:

$$\begin{aligned}\forall \mathbf{pv} \forall \upsilon(\mathbf{d}_1) \setminus \mathbf{pv} \quad & (\exists \upsilon(\mathbf{d}_2) \setminus \mathbf{pv} \forall \mathbf{v}.(\phi(\mathbf{d}_1) \wedge c) \rightarrow \phi(\mathbf{d}_2)) \\ & \wedge(\exists \upsilon(\mathbf{d}_3) \setminus \mathbf{pv} \forall \mathbf{v}.(\phi(\mathbf{d}_1) \wedge c) \rightarrow \phi(\mathbf{d}_3)),\end{aligned}$$

which reduces to validity of both $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ and $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_3$. $\square$

Lemma 3.40.

Proof. The only-if part is trivial. Concerning the if part, consider any solution $(\mathbf{v}_0, \mathbf{a}_0) \in Sol(\mathbf{A_d v} \leq \mathbf{B_d a}) \neq \emptyset$. Let $\mathbf{u}_0$ be the restriction of $\mathbf{a}_0$ to variables in $\mathbf{pv} = pvars(\mathbf{d})$. Let $\mathbf{d}'$ be obtained by replacing every $x : \square_e$ in $\mathbf{d}$ by $x : \square_{e[\mathbf{pv}/\mathbf{u}_0]}$. Since $e[\mathbf{pv}/\mathbf{u}_0] \geq 0$ is in $\mathbf{A_d v}_0 \leq \mathbf{B_d a}_0$, we have that $e[\mathbf{pv}/\mathbf{u}_0]$ reduces to a non-negative real number. Thus, $\mathbf{d}'$ is a type declaration in $\mathcal{BT}$, and:

$$\phi(\mathbf{d}') = \phi(\mathbf{d})[\mathbf{pv}/\mathbf{u}_0] \quad \upsilon(\mathbf{d}') = \upsilon(\mathbf{d}) \setminus \mathbf{pv},$$

modulo trivially true primitive constraints $e[\mathbf{pv}/\mathbf{u}_0] \geq 0$. Consider now the parameterized linear system $\mathcal{P}' = \phi(\mathbf{d}') \wedge c$. By Lemma 3.4, c satisfiable implies that there exists $\mathbf{u}_1$ instance of parameters $\upsilon(\mathbf{d}')$ such that $Sol(\mathcal{P}', \mathbf{u}_1) \neq \emptyset$. By observing that $\mathcal{P}' = \mathcal{P}[\mathbf{pv}/\mathbf{u}_0]$, we conclude that $(\mathbf{u}_0, \mathbf{u}_1)$ is an instance of $\upsilon(\mathbf{d})$ such that $Sol(\mathcal{P}, (\mathbf{u}_0, \mathbf{u}_1)) = Sol(\mathcal{P}', \mathbf{u}_1) \neq \emptyset$. $\square$

Type assertions in presence of parametric types can be instantiated by assigning real number values $\mathbf{u}$ to type variables $\mathbf{pv}$. The instantiation is a type assertions in the type system $\mathcal{BT}$ if all instantiated type expressions $e[\mathbf{pv}/\mathbf{u}]$ satisfy the requirement $e[\mathbf{pv}/\mathbf{u}] \geq 0$, namely if $\square_{e[\mathbf{pv}/\mathbf{u}]} \in \mathcal{BT}$. Next definition introduces such instances for type assertions without local variables.

Definition A.11. *Let $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ be such that $pvars(\mathbf{d}_2) \subseteq pvars(\mathbf{d}_1)$.*
A consistent instance of $\mathbf{pv} = pvars(\mathbf{d}_1)$ is any $\mathbf{u} \in \mathcal{R}^{|\mathbf{pv}|}$ such that $\Phi(\mathbf{d}_1)[\mathbf{pv}/\mathbf{u}] \wedge \Phi(\mathbf{d}_2)[\mathbf{pv}/\mathbf{u}]$ is true.

The requirement $pvars(\mathbf{d}_2) \subseteq pvars(\mathbf{d}_1)$ is equivalent to assume no local type variable in the type assertion. We are now in the position to restate validity of type assertions with parametric types in terms of validity of type assertions in the $\mathcal{BT}$ type system and a further condition. Condition *(ii)* in the next lemma is needed to cover contrived type assertions, such as the one highlighted in Example 3.50, where for the type assertion $x : \square_s, y : \square_r \vdash z = 0 \rightarrow z : \square_{s-r}$: all consistent instances lead to valid type assertions, i.e., *(i)* holds; but the type assertion is not valid due to inconsistent instances such as $s = 0, r = 1$.

Lemma A.12. *Let $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ be a type assertion such that $pvars(\mathbf{d}_2) \subseteq pvars(\mathbf{d}_1)$. $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is valid iff called $\mathbf{pv} = pvars(\mathbf{d}_1)$:*

(i). for every $\mathbf{u}$ *consistent instance of* $\mathbf{pv}$ *we have that* $\mathbf{d}_1[\mathbf{pv}/\mathbf{u}] \vdash c \rightarrow \mathbf{d}_2[\mathbf{pv}/\mathbf{u}]$ *is valid in the* $\mathcal{BT}$ *type system;*

(ii). and, $Sol(c) = \emptyset$ *or* $\forall \mathbf{pv}.\Phi(\mathbf{d}_1) \rightarrow \Phi(\mathbf{d}_2)$ *is valid in the domain of the reals.*

Proof. Validity of $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ consists of showing validity of the formula (1) in Definition 2.4, which can be rewritten as:

$$\forall \mathbf{pv} \forall v(\mathbf{d}_1) \setminus \mathbf{pv} \exists v(\mathbf{d}_2) \setminus \mathbf{pv} \forall \mathbf{v}.(\phi(\mathbf{d}_1) \wedge c) \rightarrow \phi(\mathbf{d}_2).$$

This holds iff for every $\mathbf{u} \in \mathcal{R}^{|\mathbf{pv}|}$, the following holds:

$$\forall v(\mathbf{d}_1) \setminus \mathbf{pv} \exists v(\mathbf{d}_2) \setminus \mathbf{pv} \forall \mathbf{v}.(\phi(\mathbf{d}_1)[\mathbf{pv}/\mathbf{u}] \wedge c) \rightarrow \phi(\mathbf{d}_2)[\mathbf{pv}/\mathbf{u}]. \tag{7}$$

(If-part). We show that (7) holds. When $\mathbf{u}$ is a consistent instance, this follows from *(i)*, since (7) is equivalent to the Definition 2.4 (1) formula of $\mathbf{d}_1[\mathbf{pv}/\mathbf{u}] \vdash c \rightarrow \mathbf{d}_2[\mathbf{pv}/\mathbf{u}]$, except for the fact that the $\Phi(\mathbf{d}_1)$ and $\Phi(\mathbf{d}_2)$ constraints appear in the formula. However, since $\mathbf{u}$ is a consistent instance, they are trivially satisfied.

Assume now that $\mathbf{u}$ is not a consistent instance.

If $Sol(c) = \emptyset$ then the conclusion readily follows. Also, it follows if $\Phi(\mathbf{d}_1)[\mathbf{pv}/\mathbf{u}]$ is false, since it is a conjunct of $\phi(\mathbf{d}_1)[\mathbf{pv}/\mathbf{u}]$. Finally, when $\Phi(\mathbf{d}_1)[\mathbf{pv}/\mathbf{u}]$ is true, the hypothesis $\forall \mathbf{pv}.\Phi(\mathbf{d}_1) \rightarrow \Phi(\mathbf{d}_2)$ implies that $\Phi(\mathbf{d}_2)[\mathbf{pv}/\mathbf{u}]$ is true as well. However, this and $\Phi(\mathbf{d}_1)[\mathbf{pv}/\mathbf{u}]$ contradict the assumption that $\mathbf{u}$ is not a consistent instance.

(Only-if part). *(i)* follows from (7), which is equivalent to the Definition 2.4 (1) formula of $\mathbf{d}_1[\mathbf{pv}/\mathbf{u}] \vdash c \rightarrow \mathbf{d}_2[\mathbf{pv}/\mathbf{u}]$, except for the fact that the $\Phi(\mathbf{d}_1)$ and $\Phi(\mathbf{d}_2)$ constraints appear in the formula. However, since $\mathbf{u}$ is a consistent instance, we have that they reduce to the `true` constraint.

(ii) If $Sol(c) = \emptyset$ then then *(ii)* readily follows. Assume then $Sol(c) \neq \emptyset$ and that for some $\mathbf{u}_0$ instance of $\mathbf{pv}$ we have that $\Phi(\mathbf{d}_1)[\mathbf{pv}/\mathbf{u}_0] \rightarrow \Phi(\mathbf{d}_2)[\mathbf{pv}/\mathbf{u}_0]$ is false, namely $\Phi(\mathbf{d}_1)[\mathbf{pv}/\mathbf{u}_0] \wedge \neg\Phi(\mathbf{d}_2)[\mathbf{pv}/\mathbf{u}_0]$ is true. Given $\mathbf{v}_0 \in Sol(c)$, it can be extended to a solution of $\phi(\mathbf{d}_1)[\mathbf{pv}/\mathbf{u}_0] \wedge c$ by setting parameters in $\mathbf{a} = v(\mathbf{d}_1) \setminus \mathbf{pv}$ to the value of variables in $\mathbf{v}_0$ they constraint. As an example, for $\phi(x : \Box_s) = a \leq x \leq a + s \wedge s \geq 0$ we choose for a the value of x in $\mathbf{v}_0$ (this choice also satisfies $x \leq a + s$ since $s \geq 0$ is assumed by $\Phi(\mathbf{d}_1)[\mathbf{pv}/\mathbf{u}_0]$). By (7), $\phi(\mathbf{d}_1)[\mathbf{pv}/\mathbf{u}_0] \wedge c$ implies $\phi(\mathbf{d}_2)[\mathbf{pv}/\mathbf{u}_0]$, and then $\Phi(\mathbf{d}_2)[\mathbf{pv}/\mathbf{u}_0]$ which is part of it. This is absurd since $\neg\Phi(\mathbf{d}_2)[\mathbf{pv}/\mathbf{u}_0]$ was assumed. □

By using the characterization of validity of type assertions with parametric types in terms of validity of type assertions in $\mathcal{BT}$, we can extend to parametric types results that hold in $\mathcal{BT}$.

Theorem 3.43 [(Definiteness III)].

Proof. In the following we can ignore condition *(ii)* of Lemma A.12 since either it holds for both $\mathbf{d} \vdash c \rightarrow x : \Box_e$ and $\mathbf{d}|_{\mathcal{B}} \vdash c \rightarrow x : \Box_e$ or for none.

By Lemma A.12, $\mathbf{d} \vdash c \rightarrow x : \Box_e$ is valid iff for every $\mathbf{u}$ consistent instance of $\mathbf{pv} = pvars(\mathbf{d}_1)$, the type assertion $\mathbf{d}[\mathbf{pv}/\mathbf{u}] \vdash c \rightarrow x : \Box_e[\mathbf{pv}/\mathbf{u}]$ is valid. By Theorem 3.22, called $\mathcal{B}' = \mathcal{BT} \setminus \mathcal{BT}_\Box$, this holds iff $\mathbf{d}[\mathbf{pv}/\mathbf{u}]|_{\mathcal{B}'} \vdash c \rightarrow x : \Box_e[\mathbf{pv}/\mathbf{u}]$ is valid. Since the restriction to $\mathcal{B}'$ simply removes atd's with types in $\mathcal{BT}_\Box$, we observe that $\mathbf{d}[\mathbf{pv}/\mathbf{u}]|_{\mathcal{B}'} = \mathbf{d}|_{\mathcal{B}}[\mathbf{pv}/\mathbf{u}]$. Summarizing, $\mathbf{d} \vdash c \rightarrow x : \Box_e$ is valid iff for every $\mathbf{u}$ consistent instance of $\mathbf{pv}$, the type assertion $\mathbf{d}|_{\mathcal{B}}[\mathbf{pv}/\mathbf{u}] \vdash c \rightarrow x : \Box_e[\mathbf{pv}/\mathbf{u}]$ is valid. By Lemma A.12, this is equivalent to validity of $\mathbf{d}|_{\mathcal{B}} \vdash c \rightarrow x : \Box_e$. □

LEMMA 3.46.

PROOF. Notice that since $Sol(\mathbf{A}\mathbf{x} \leq \mathbf{b}) \neq \emptyset$, its generating and vertex matrices $\mathbf{R}$ and $\mathbf{V}$ exist. Let us set $\mathbf{c} = \mathbf{c}_1 - \mathbf{c}_2$ and $\alpha = \alpha_1 - \alpha_2$. It is immediate to observe that $\mathbf{c}_1^T\mathbf{x} + \alpha_1 \preceq_e \mathbf{c}_2^T\mathbf{x} + \alpha_2$ iff $\mathbf{c}^T\mathbf{x} + \alpha \preceq_e 0$.

(*If-part*). For every $\mathbf{x}_0$ such that $\mathbf{A}\mathbf{x}_0 \leq \mathbf{b}$ holds, there exists $\boldsymbol{\lambda}, \boldsymbol{\mu} \geq \mathbf{0}$ with $\Sigma\boldsymbol{\mu} = \mathbf{1}^T\boldsymbol{\mu} = 1$ such that $\mathbf{x}_0 = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}\boldsymbol{\mu}$. We distinguish two cases.

First, consider $abs(\mathbf{c}^T\mathbf{x}_0 + \alpha) = \mathbf{c}^T\mathbf{x}_0 + \alpha$. We have $\mathbf{c}^T\mathbf{x}_0 + \alpha = \mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} + \mathbf{c}^T\mathbf{V}\boldsymbol{\mu} + \mathbf{1}^T\boldsymbol{\mu}\alpha \leq \mathbf{c}_e^T\mathbf{R}\boldsymbol{\lambda} + \mathbf{c}_e^T\mathbf{V}\boldsymbol{\mu} + \mathbf{1}^T\boldsymbol{\mu}r = \mathbf{c}_e^T\mathbf{x}_0 + r$ by exploiting the hypotheses $(\mathbf{c} - \mathbf{c}_e)^T\mathbf{R} \leq \mathbf{0}$ and $(\mathbf{c} - \mathbf{c}_e)^T\mathbf{V} + (\alpha - r)\mathbf{1}^T \leq \mathbf{0}$.

Second, consider $abs(\mathbf{c}^T\mathbf{x}_0 + \alpha) = -\mathbf{c}^T\mathbf{x}_0 - \alpha$. We have $-\mathbf{c}^T\mathbf{x}_0 - \alpha = -\mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} - \mathbf{c}^T\mathbf{V}\boldsymbol{\mu} - \mathbf{1}^T\boldsymbol{\mu}\alpha \leq \mathbf{c}_e^T\mathbf{R}\boldsymbol{\lambda} + \mathbf{c}_e^T\mathbf{V}\boldsymbol{\mu} + \mathbf{1}^T\boldsymbol{\mu}r = \mathbf{c}_e^T\mathbf{x}_0 + r$ by exploiting the hypotheses $(-\mathbf{c} - \mathbf{c}_e)^T\mathbf{R} \leq \mathbf{0}$ and $(-\mathbf{c} - \mathbf{c}_e)^T\mathbf{V} + (-\alpha - r)\mathbf{1}^T \leq \mathbf{0}$.

(*Only-if part*) Let $\mathbf{V}_k$ be any column of $\mathbf{V}$.

If $(\mathbf{c}-\mathbf{c}_e)^T\mathbf{R} \not\leq \mathbf{0}$ then there exists a column $\mathbf{R}_j$ of $\mathbf{R}$ such that $(\mathbf{c}-\mathbf{c}_e)^T\mathbf{R}_j > 0$. By choosing $\boldsymbol{\lambda}_i = 0$ for $i \neq j$, $\boldsymbol{\mu}_i = 0$ for $i \neq k$ and $\boldsymbol{\mu}_k = 1$, for $\mathbf{x}_0 = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}\boldsymbol{\mu}$ we get $\mathbf{c}^T\mathbf{x}_0 + \alpha = \mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} + \mathbf{c}^T\mathbf{V}\boldsymbol{\mu} + \alpha = \mathbf{c}^T\mathbf{R}_j\boldsymbol{\lambda}_j + \mathbf{c}^T\mathbf{V}_k + \alpha \not\leq \mathbf{c}_e^T\mathbf{R}_j\boldsymbol{\lambda}_j + \mathbf{c}_e^T\mathbf{V}_k + r = \mathbf{c}_e^T\mathbf{x}_0 + r$, since $(\mathbf{c}-\mathbf{c}_e)^T\mathbf{R}_j > 0$ and $\boldsymbol{\lambda}_j$ can be any positive real number. For some $\boldsymbol{\lambda}_j$, we can then conclude $abs(\mathbf{c}^T\mathbf{x}_0 + \alpha) \geq \mathbf{c}^T\mathbf{x}_0 + \alpha > \mathbf{c}_e^T\mathbf{x}_0 + r$.

If $(-\mathbf{c}-\mathbf{c}_e)^T\mathbf{R} \not\leq \mathbf{0}$ then there exists a column $\mathbf{R}_j$ of $\mathbf{R}$ such that $(-\mathbf{c}-\mathbf{c}_e)^T\mathbf{R}_j > 0$. By choosing $\boldsymbol{\lambda}_i = 0$ for $i \neq j$, $\boldsymbol{\mu}_i = 0$ for $i \neq k$ and $\boldsymbol{\mu}_k = 1$, for $\mathbf{x}_0 = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}\boldsymbol{\mu}$ we get $-\mathbf{c}^T\mathbf{x}_0 - \alpha = -\mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} - \mathbf{c}^T\mathbf{V}\boldsymbol{\mu} - \alpha = -\mathbf{c}^T\mathbf{R}_j\boldsymbol{\lambda}_j - \mathbf{c}^T\mathbf{V}_k - \alpha \not\leq \mathbf{c}_e^T\mathbf{R}_j\boldsymbol{\lambda}_j + \mathbf{c}_e^T\mathbf{V}_k + r = \mathbf{c}_e^T\mathbf{x}_0 + r$, since $(-\mathbf{c}-\mathbf{c}_e)^T\mathbf{R}_j > 0$ and $\boldsymbol{\lambda}_j$ can be any positive real number. For some $\boldsymbol{\lambda}_j$, we can then conclude $abs(\mathbf{c}^T\mathbf{x}_0 + \alpha) \geq -\mathbf{c}^T\mathbf{x}_0 - \alpha > \mathbf{c}_e^T\mathbf{x}_0 + r$.

If $(\mathbf{c} - \mathbf{c}_e)^T\mathbf{V} + (\alpha - r)\mathbf{1}^T \not\leq \mathbf{0}$ then there exists a column $\mathbf{V}_j$ of $\mathbf{V}$ such that $(\mathbf{c} - \mathbf{c}_e)^T\mathbf{V}_j + (\alpha - r) > 0$. By choosing $\boldsymbol{\lambda} = \mathbf{0}$, $\boldsymbol{\mu}_i = 0$ for $i \neq j$ and $\boldsymbol{\mu}_j = 1$, for $\mathbf{x}_0 = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}\boldsymbol{\mu}$ we get $\mathbf{c}^T\mathbf{x}_0 + \alpha = \mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} + \mathbf{c}^T\mathbf{V}\boldsymbol{\mu} + \alpha = \mathbf{c}^T\mathbf{V}_j + \alpha \not\leq \mathbf{c}_e^T\mathbf{V}_j + r = \mathbf{c}_e^T\mathbf{x}_0 + r$. This implies $abs(\mathbf{c}^T\mathbf{x}_0 + \alpha) \geq \mathbf{c}^T\mathbf{x}_0 + \alpha > \mathbf{c}_e^T\mathbf{x}_0 + r$.

If $(-\mathbf{c} - \mathbf{c}_e)^T\mathbf{V} + (-\alpha - r)\mathbf{1}^T \not\leq \mathbf{0}$ then there exists a column $\mathbf{V}_j$ of $\mathbf{V}$ such that $(-\mathbf{c} - \mathbf{c}_e)^T\mathbf{V}_j + (-\alpha - r) > 0$. By choosing $\boldsymbol{\lambda} = \mathbf{0}$, $\boldsymbol{\mu}_i = 0$ for $i \neq j$ and $\boldsymbol{\mu}_j = 1$, for $\mathbf{x}_0 = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}\boldsymbol{\mu}$ we get $-\mathbf{c}^T\mathbf{x}_0 - \alpha = -\mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} - \mathbf{c}^T\mathbf{V}\boldsymbol{\mu} - \alpha = -\mathbf{c}^T\mathbf{V}_j - \alpha \not\leq \mathbf{c}_e^T\mathbf{V}_j + r = \mathbf{c}_e^T\mathbf{x}_0 + r$. This implies $abs(\mathbf{c}^T\mathbf{x}_0 + \alpha) \geq -\mathbf{c}^T\mathbf{x}_0 - \alpha > \mathbf{c}_e^T\mathbf{x}_0 + r$. □

LEMMA 3.47.

PROOF. For a parameter instance $\mathbf{u}$ let us denote by $\mathcal{V}^{\mathbf{u}}$ the set of (instances of the) parameterized vertices whose domain includes $\mathbf{u}$, namely: $\mathcal{V}^{\mathbf{u}} = \{\mathbf{v}^{\mathbf{u}}(i) \mid i = 1..k, \mathbf{C}_i\mathbf{u} \leq \mathbf{c}_i\ \}$. If $\mathcal{V}^{\mathbf{u}} \neq \emptyset$, let us denote by $\mathbf{V}^{\mathbf{u}}$ the matrix whose columns are the vectors in $\mathcal{V}^{\mathbf{u}}$.

Consider the if-part. Let $\mathbf{u}$ be fixed. If $\mathcal{S}_{\mathbf{u}} = \emptyset$, we are done. Otherwise, $\mathcal{V}^{\mathbf{u}}$ cannot be empty. Since $\mathbf{c}^T\mathbf{R} = \mathbf{0}$ (assumption *(i)*), we calculate:

$$\begin{aligned}\mathcal{S}_{\mathbf{u}} &= \{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}^{\mathbf{u}}\boldsymbol{\mu}, \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\}\\ &= \{z \mid z = \mathbf{c}^T\mathbf{V}^{\mathbf{u}}\boldsymbol{\mu}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\}.\end{aligned}$$

Notice that for $x, y \in \mathcal{S}_{\mathbf{u}}$, $abs(x - y)$ can then be rewritten as $abs(\mathbf{c}^T\mathbf{V}^{\mathbf{u}}(\boldsymbol{\mu}_x - \boldsymbol{\mu}_y))$, for some $\boldsymbol{\mu}_x \geq \mathbf{0}, \Sigma\boldsymbol{\mu}_x = 1, \boldsymbol{\mu}_y \geq \mathbf{0}, \Sigma\boldsymbol{\mu}_y = 1$. Moreover, by *(iii)*, $\mathbf{c}_e^T\mathbf{u} + r \geq 0$. These last two statements and assumption *(ii)* allow for applying Lemma A.5 to conclude $abs(x - y) \leq \mathbf{c}_e^T\mathbf{u} + r$ for every $x, y \in \mathcal{S}_{\mathbf{u}}$.

Consider now the only-if part. We show the contrapositive.

(i). If $\mathbf{c}^T\mathbf{R} \neq \mathbf{0}$, consider any $\mathbf{u}$ in $Sol(\mathbf{C}_1\mathbf{a} \leq \mathbf{c}_1)$. Notice that $k \geq 1$ since the parameterized polyhedron is assumed not to be empty, and $Sol(\mathbf{C}_1\mathbf{a} \leq \mathbf{c}_1) \neq \emptyset$ by Theorem 3.24. We have:

$$\begin{aligned} \mathcal{S}_\mathbf{u} &= \{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}^\mathbf{u}\boldsymbol{\mu}, \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\} \\ &= \{z \mid z = \mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} + \mathbf{c}^T\mathbf{V}^\mathbf{u}\boldsymbol{\mu}, \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\}. \end{aligned}$$

Let $\mathbf{R}_j$ be a column of $\mathbf{R}$ such that $\mathbf{c}^T\mathbf{R}_j = \alpha \neq 0$. By fixing $\boldsymbol{\lambda}_i = 0$ for $i \neq j$, we get that $\mathbf{c}^T\mathbf{R}\boldsymbol{\lambda} = \alpha\boldsymbol{\lambda}_j$. By fixing $\boldsymbol{\mu}$ in any way such that $\boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1$, we also have $\mathbf{c}^T\mathbf{V}^\mathbf{u}\boldsymbol{\mu} = \beta$, for some β. Summarizing, $\mathcal{S}_\mathbf{u}$ includes $\alpha\boldsymbol{\lambda}_j + \beta$ for every $\boldsymbol{\lambda}_j \geq 0$. Hence, $abs(x - y)$ cannot be bounded by $\mathbf{c}_e^T\mathbf{u} + r$ for every $x, y \in \mathcal{S}_\mathbf{u}$.

(ii). Consider a pair $1 \leq m < n \leq k$ such that $\mathbf{c}^T\mathbf{v}^\mathbf{a}(m) \not\preceq_e \mathbf{c}^T\mathbf{v}^\mathbf{a}(n)$ over $P_{m,n}$. Then there exists $\mathbf{u} \in Sol(P_{m,n})$ such that $abs(\mathbf{c}^T\mathbf{v}^\mathbf{u}(m) - \mathbf{c}^T\mathbf{v}^\mathbf{u}(n)) > \mathbf{c}_e^T\mathbf{u} + r$. By fixing $\boldsymbol{\mu}_i = 0$ for $i \neq m$ and $\boldsymbol{\mu}_m = 1$, we define $\mathbf{x} = \mathbf{V}^\mathbf{u}\boldsymbol{\mu} = \mathbf{v}^\mathbf{u}(m)$. Analogously, by fixing $\boldsymbol{\mu}_i = 0$ for $i \neq n$ and $\boldsymbol{\mu}_n = 1$, we define $\mathbf{y} = \mathbf{V}^\mathbf{u}\boldsymbol{\mu} = \mathbf{v}^\mathbf{u}(n)$. It turns out $x = \mathbf{c}^T\mathbf{x}, y = \mathbf{c}^T\mathbf{y}$ belong to:

$$\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} = \mathbf{V}^\mathbf{u}\boldsymbol{\mu}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\} = \{z \mid z = \mathbf{c}^T\mathbf{V}^\mathbf{u}\boldsymbol{\mu}, \boldsymbol{\mu} \geq \mathbf{0}, \Sigma\boldsymbol{\mu} = 1\} \subseteq \mathcal{S}_\mathbf{u},$$

and $abs(x - y) > \mathbf{c}_e^T\mathbf{u} + r$.

(iii) By Theorem 3.24, $Sol(\mathbf{C}_i\mathbf{a} \leq \mathbf{c}_i) \neq \emptyset$ for $i = 1..k$. Therefore, the negation of condition *(iii)* is equivalent to assume that for some i in $1..k$, there exists $\mathbf{u} \in Sol(\mathbf{C}_i\mathbf{a} \leq \mathbf{c}_i)$ such that $\mathbf{c}_e^T\mathbf{u} + r < 0$. By Theorem 3.24 again, $\mathcal{S}_\mathbf{u} \neq \emptyset$ – at least $\mathbf{v}^\mathbf{u}(i)$ belongs to it. Therefore, for any $x \in \mathcal{S}_\mathbf{u}$, we have $abs(x - x) = 0 > \mathbf{c}_e^T\mathbf{u} + r$. □

LEMMA 3.49.

PROOF. $\mathbf{x}_0 \in Sol(\mathbf{A}\mathbf{x} \leq \mathbf{b})$ iff $\mathbf{x}_0 = \mathbf{R}\boldsymbol{\lambda} + \mathbf{V}\boldsymbol{\mu}$ for some $\boldsymbol{\lambda}, \boldsymbol{\mu} \geq \mathbf{0}$ and $\Sigma\boldsymbol{\mu} = \mathbf{1}^T\boldsymbol{\mu} = 1$. Then: $\mathbf{c}_e^T\mathbf{x}_0 + r = \mathbf{c}_e^T\mathbf{R}\boldsymbol{\lambda} + \mathbf{c}_e^T\mathbf{V}\boldsymbol{\mu} + r\mathbf{1}^T\boldsymbol{\mu} = \mathbf{c}_e^T\mathbf{R}\boldsymbol{\lambda} + (\mathbf{c}_e^T\mathbf{V} + r\mathbf{1}^T)\boldsymbol{\mu}$.

The if part is immediate, namely $\mathbf{c}_e^T\mathbf{x}_0 + r \geq 0$ since $\mathbf{c}_e^T\mathbf{R}$, $\boldsymbol{\lambda}$, $(\mathbf{c}_e^T\mathbf{V} + r\mathbf{1}^T)$ and $\boldsymbol{\mu}$ contains only non-negative elements. The only-if part follows by contraposition. If for some $\mathbf{x}_0$, we have $\mathbf{c}_e^T\mathbf{x}_0 + r < 0$ then $\mathbf{c}_e^T\mathbf{R}\boldsymbol{\lambda}$ or $(\mathbf{c}_e^T\mathbf{V} + r\mathbf{1}^T)\boldsymbol{\mu}$ is a negative number. Since $\boldsymbol{\lambda}, \boldsymbol{\mu} \geq \mathbf{0}$, this implies that $\mathbf{c}_e^T\mathbf{R} \not\geq \mathbf{0}$ or $(\mathbf{c}_e^T\mathbf{V} + r\mathbf{1}^T) \not\geq \mathbf{0}$. □

THEOREM 3.51 [(PARCHECK - SOUNDNESS AND COMPLETENESS)].

PROOF. Working on $\mathbf{d} = \mathbf{d}_1|_\mathcal{B}$ is sound and complete by Theorem 3.43.

(Soundness) Step 2 is justified by the if-part of Lemma 3.40.

Step 4 is justified by the if-part of Lemma 3.47 instantiated by fixing $\mathbf{c}_j = 0$ for $j \neq i$, and $\mathbf{c}_i = 1$, so that $\mathbf{c}^T\mathbf{v} = x$. Also, since there is no local type variable in $x : \Box_e$, e can be written as a linear combination of type variables in $\mathbf{d}_1$. Since $pvars(\mathbf{d}_1) \subseteq v(\mathbf{d}_1) = \mathbf{a}$, the assumption of Lemma 3.47 that e is of the form $\mathbf{c}_e^T\mathbf{a} + r$ is satisfied.

(Completeness) Assume that $\mathbf{d} \vdash c \rightarrow x : \Box_e$ is valid, and consider the parameterized system $\mathcal{P} = \phi(\mathbf{d}) \wedge c$.

Unsatisfiability of $\mathcal{P}$ is checked by *Step 2*, using the conditions of Lemma 3.40.

Step 4 deals with the case when $\mathcal{P}$ is satisfiable by checking the conditions of Lemma 3.47. □

Lemma 3.52.

Proof. Let $\mathcal{P}$ be the parameterized linear system $\phi(\mathbf{d}_1) \wedge c$. Since $\phi(\mathbf{d}_1)$ is satisfiable, by Lemma 3.40, $\mathcal{P}$ is satisfiable iff c is satisfiable. Therefore, for $\mathcal{P}$ unsatisfiable the conclusion holds since both $\mathbf{d}_1 \vdash c \rightarrow x : \tau$ and $\mathbf{d}'_1 \vdash c \rightarrow x : \tau$ are valid. Assume now $\mathcal{P}$ satisfiable. Let $\mathcal{P}'$ be the parameterized linear system $\phi(\mathbf{d}'_1) \wedge c$. By construction of $\mathbf{d}'_1$, the homogeneous versions of $\mathcal{P}$ and $\mathcal{P}'$ coincide. For instance, the homogeneous version of $\phi(x : \Box_e) = a \leq x \leq a + e \wedge 0 \leq e$ is $x = 0 \wedge 0 \leq 0$, which (up to the true inequality $0 \leq 0$) coincides with the homogeneous version of $\phi(x : \Box) = a \leq x \leq b$. By Lemma 3.5: $\mathbf{d}'_1 \vdash c \rightarrow x : \sqcap$ is valid iff $max\{\mathbf{v}_i \mid \mathbf{v} \in Sol(\mathcal{H}')\} = 0$, where $\mathbf{v}_i = x$ an $\mathcal{H}'$ is the homogeneous version of $\mathcal{P}'$. Since $\mathcal{P}$ is satisfiable, an immediate extension of Lemma 3.5 to parametric types allows for concluding: $\mathbf{d}_1 \vdash c \rightarrow x : \sqcap$ is valid iff $max\{\mathbf{v}_i \mid \mathbf{v} \in Sol(\mathcal{H})\} = 0$, where $\mathcal{H}$ is the homogeneous version of $\mathcal{P}$. Our conclusion then follows since $\mathcal{H} = \mathcal{H}'$. □

A.8 Section 3.7

Theorem 3.53.

Proof. By Theorem 3.19 and Lemma 2.17, Check(IEInfer) is a decision procedure, complete for $\mathcal{BT}_!$. Its complexity is polynomial by observing that:

—LPInfer has polynomial time complexity when adopting a polynomial time algorithm for linear programming [Khachiyan 1979];

—computing the set of implicit equalities for a system of n inequalities can be done by solving at most n linear programming problems [Greenberg 1996, Corollary 7.4], which, again, requires polynomial time;

—Gaussian elimination has polynomial time complexity [Schrijver 1986].

□

A.9 Section 4.2

The operational semantics of CLP($\mathcal{R}$) programs is specified in terms of *derivations* from states to states [Jaffar and Maher 1994; Jaffar et al. 1998]. Here, we consider derivations via the leftmost selection rule. A state $\langle \leftarrow [c,]A_1, \ldots, A_n \| c' \rangle$ is reduced to another state, called a *resolvent*, as follows:

$\mathcal{R}1$. If a linear constraint c appears at the left of the query, and $c \wedge c'$ is satisfiable, the resolvent is $\langle \leftarrow A_1, \ldots, A_n \| c' \wedge c \rangle$.

$\mathcal{R}2$. If an atom $A_1 = p(x_1, \ldots, x_h)$ appears at the left of the query, and $p(y_1, \ldots, y_h) \leftarrow c, B_1, \ldots, B_k$ is a renamed apart clause from P, then the resolvent is:

$$\langle \leftarrow c, B_1, \ldots, B_k, A_2, \ldots, A_n \| c' \wedge \bigwedge_{i=1..h} x_i = y_i \rangle.$$

A *derivation* from a state S is a (finite or infinite) maximal sequence of states $S_0 = S, S_1, \ldots, S_n, \ldots$ such that there is a reduction from S_i to S_{i+1}, for $i \geq 0$. A derivation for a query Q is a derivation from $\langle Q \| \texttt{true} \rangle$. The last state of a finite derivation is of the form $\langle Q \| c \rangle$. If Q is not empty, the derivation is *failed.* Otherwise, it is *successful*, or a *refutation*, and c is called the answer constraint.

Theorem 4.5.

Proof. (Persistency). It is a special case of the more general Theorem 4.18, which will be shown in Appendix A.11.

(Call patterns). The state $\langle \leftarrow q(\mathbf{x} : \boldsymbol{\tau} \times \boldsymbol{\mu}), R \| \ c' \rangle$ is well-moded by persistency. From the definition of well-moded states ($i = 1$ and $c = \mathtt{true}$), we have that $\vdash c' \rightarrow \mathbf{x} : \boldsymbol{\tau}$ is valid.

(Answers). Consider the program P' defined as P plus the unit clause:

$$p(\mathbf{x}_1 : \boldsymbol{\mu}_1 \times \star, \ldots, \mathbf{x}_n : \boldsymbol{\mu}_n \times \star) \leftarrow \mathtt{true}.$$

where p is a fresh predicate symbol, and the query $Q' = Q, \ p(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. P' and Q' are well-moded since P and Q are well-moded. Under our hypotheses, there exists a derivation from the query Q' to the state $\langle \leftarrow p(\mathbf{x}_1, \ldots, \mathbf{x}_n) \mid c' \rangle$. By the call pattern conclusion, we obtain $\vdash c' \rightarrow \mathbf{x}_1 : \boldsymbol{\mu}_1, \ldots, \mathbf{x}_n : \boldsymbol{\mu}_n$. □

A.10 Section 4.3

We recall that a term substitution θ is represented by a finite set $\{x_1/t_1, \ldots, x_n/t_n\}$ with $x_1, \ldots, x_n$ distinct variables, and $t_1, \ldots, t_n$ terms, and $n \geq 0$. Also, we can assume $t_i \neq x_i$. $dom(\theta) = \{x_1, \ldots, x_n\}$ is called the domain of variables in θ.

Lemma 4.11.

Proof. By Lemma A.1, we have to show that $\mathbf{d}_1\theta \vdash c \rightarrow (x : \tau)\theta$ is valid for every $x : \tau$ in $\mathbf{d}_2$. Also, we can assume $\tau \neq \star$, otherwise the conclusion is trivial. Moreover, notice that the assumption $c\theta = c$ implies that $vars(c) \cap dom(\theta) = \emptyset$.

First, consider the case $x/t \in \theta$ for some t. Thus, $x \notin vars(c)$. By Lemma A.2 and $\tau \neq \star$, $\mathbf{d}_1 \vdash c \rightarrow x : \tau$ valid implies that there exists $x : \mu$ in $nf(\mathbf{d}_1)$ such that $\mu \geq_t \tau$. By Definition 4.9, for every $y \in vars(t)$, $y : \mu$ is in $nf(\mathbf{d}_1)\theta = nf(\mathbf{d}_1\theta)$. By Lemma 2.11, this and $\mu \geq_t \tau$ imply that $\mathbf{d}_1\theta \vdash \mathtt{true} \rightarrow y : \tau$ is valid. By monotonicity, $\mathbf{d}_1\theta \vdash c \rightarrow y : \tau$ is valid for every $y \in vars(t)$. By Definition 4.9 and Lemma A.1, this is equivalent to validity of $\mathbf{d}_1\theta \vdash c \rightarrow (x : \tau)\theta$.

Consider now the case $x \notin dom(\theta)$. Thus, $(x : \tau)\theta$ is $x : \tau$.

If $x \notin vars(c)$, by Lemma A.2 and $\tau \neq \star$, $\mathbf{d}_1 \vdash c \rightarrow x : \tau$ valid implies that there exists $x : \mu$ in $nf(\mathbf{d}_1)$ such that $\mu \geq_t \tau$. Since $x \notin dom(\theta)$, $x : \mu$ is in $nf(\mathbf{d}_1\theta)$ as well. Again by Lemma A.2, we conclude that $\mathbf{d}_1\theta \vdash c \rightarrow x : \tau$ valid.

If $x \in vars(c)$, by Lemma A.2, $\mathbf{d}_1 \vdash c \rightarrow x : \tau$ valid implies $\mathbf{d} \vdash c \rightarrow x : \tau$ valid, where $\mathbf{d} = \mathbf{d}_1|_{vars(c)}$. Since $vars(c) \cap dom(\theta) = \emptyset$, we have $\mathbf{d}\theta = \mathbf{d}$, and then $\mathbf{d}\theta \vdash c \rightarrow x : \tau$ is valid. By monotonicity, from $\mathbf{d}_1\theta \geq_t \mathbf{d}\theta$, we conclude that $\mathbf{d}_1\theta \vdash c \rightarrow x : \tau$ is valid. □

A.11 Section 4.4

Theorem 4.18 [(Persistency)].

Proof. Let $S = \langle \leftarrow [c,] \ A_1, \ldots, A_n \| c' \rangle$ be well-moded using type substitutions $\vartheta_1^s, \ldots, \vartheta_n^s$. For $A_i = p(\mathbf{x} : \boldsymbol{\tau} \times \boldsymbol{\mu})$, $i = 1..n$, we denote $pre(A_i) = \mathbf{x} : \vartheta_i^s(\boldsymbol{\tau})$ and $post(A_i) = \mathbf{x} : \vartheta_i^s(\boldsymbol{\mu})$. We will show that every resolvent of S and a well-moded clause is well-moded. The final result follows since the initial state $\langle Q \| \mathtt{true} \rangle$ is well-moded by hypothesis. If rule $\mathcal{R}_1$ is applied, the result is trivial since, by definition, S is well-moded iff $\langle \leftarrow A_1, \ldots, A_n \| c' \wedge c \rangle$ is well-moded.

Consider now rule $\mathcal{R}_2$. Let $A_1 = p(x_1, \ldots, x_h)$ and let:

$$\langle \leftarrow c, B_1, \ldots, B_k, A_2, \ldots, A_n \| c' \wedge \bigwedge_{i=1..h} x_i = y_i \rangle$$

be the resolvent obtained by a renamed apart clause $H \leftarrow\ c, B_1\ , \ldots\ , B_k$, where $H = p(y_1, \ldots, y_h)$. It is readily checked that renaming apart a well-moded clause yields a well-moded clause as well. Let $\vartheta_1^r, \ldots, \vartheta_k^r$ be the type substitutions as in Definition 4.17. For $B_i = p(\mathbf{x} : \boldsymbol{\tau} \times \boldsymbol{\mu})$, $i = 1..k$, we denote $pre(B_i) = \mathbf{x} : \vartheta_i^r(\boldsymbol{\tau})$ and $post(B_i) = \mathbf{x} : \vartheta_i^r(\boldsymbol{\mu})$. Also, for $H = p(\mathbf{y} : \boldsymbol{\tau} \times \boldsymbol{\mu})$, we define $pre(H) = \mathbf{y} : \boldsymbol{\tau}$ and $post(H) = \mathbf{y} : \boldsymbol{\mu}$.

We will show that the resolvent is well-moded by using the type substitutions $\vartheta_1, \ldots, \vartheta_{k+n-1}$ defined as follows: $\vartheta_i = \vartheta_1^s \circ \vartheta_i^r$ for $i = 1..k$; $\vartheta_{k+i-1} = \vartheta_i^s$ for $i = 2..n$.

By well-moding of S, $\vdash c' \rightarrow pre(A_1)$ is valid and has no local type variable. Since c' is satisfiable (it is a simple induction on the length of derivations), this implies that $pre(A_1)$ is a type declaration in $\mathcal{BT}$, namely $\vartheta_1^s(e) \geq 0$ for every $\Box_e$ in $pre(A_1)$. Called $c_1 = c' \wedge \bigwedge_{i=1..h} x_i = y_i$, this implies that $\vdash c_1 \rightarrow \overline{pre}(H)$ is valid, where, for $H = p(\mathbf{y} : \boldsymbol{\tau} \times \boldsymbol{\mu})$, $\overline{pre}(H) = \mathbf{y} : \vartheta_1^s(\boldsymbol{\tau})$ is a type declaration in $\mathcal{BT}$.

Let us split the rest of the proof in two steps **(a)** and **(b)**.

(a) For $i = 1..k$, from well-moding of the (renamed-apart) clause, we have that:

$$pre(H), post(B_1), \ldots, post(B_{i-1}) \vdash c \rightarrow pre(B_i)$$

is valid and has no local type variable. Since $\mathcal{R} \models c_1 \wedge c \rightarrow c$, this implies that:

$$pre(H), post(B_1), \ldots, post(B_{i-1}) \vdash c_1 \wedge c \rightarrow pre(B_i) \tag{8}$$

is valid and has no local type variable. By recalling that no local to output type variable is admitted in parametric modes (see Definition 4.12), a simple induction on i reveals that all type variables in (8) are included in $pre(H)$. We instantiate all of them by ϑ_1^s as follows: for $i = 1..k$ and $B_i = p(\mathbf{x} : \boldsymbol{\tau} \times \boldsymbol{\mu})$, we denote $\overline{pre}(B_i) = \mathbf{x} : \vartheta_1^s(\vartheta_i^r(\boldsymbol{\tau}))$ and $\overline{post}(B_i) = \mathbf{x} : \vartheta_1^s(\vartheta_i^r(\boldsymbol{\mu}))$. The following is an instance of (8):

$$\overline{pre}(H), \overline{post}(B_1), \ldots, \overline{post}(B_{i-1}) \vdash c_1 \wedge c \rightarrow \overline{pre}(B_i) \tag{9}$$

and it is valid. This comes directly from Definition 2.4 of validity of type assertions. Moreover, since $\overline{pre}(H)$ is in $\mathcal{BT}$, and (9) is an instance of (8), there is no type variable at all in (9). Assume for the moment that it is a type assertion in $\mathcal{BT}$. By monotonicity in $\mathcal{BT}$, $\vdash c_1 \rightarrow \overline{pre}(H)$ valid implies that:

$$\overline{post}(B_1), \ldots, \overline{post}(B_{i-1}) \vdash c_1 \wedge c \rightarrow \overline{pre}(H) \tag{10}$$

is valid, and in $\mathcal{BT}$. By the transitivity Lemma 2.12, (9) and (10) imply that:

$$\overline{post}(B_1), \ldots, \overline{post}(B_{i-1}) \vdash c_1 \wedge c \rightarrow \overline{pre}(B_i) \tag{11}$$

is valid. This concludes the proof, since this is the proof obligation of well-moding for type substitutions $\vartheta_1, \ldots, \vartheta_{k+n-1}$ and $i = 1..k$.

Assume now that (9) is not a type assertion in $\mathcal{BT}$. If $c_1 \wedge c$ is not satisfiable, then validity of (11) is immediate. Let $c_1 \wedge c$ be satisfiable. Since (9) contains no type variable, it is not in $\mathcal{BT}$ only if for some type $\Box_e$ in $\overline{post}(B_1), \ldots, \overline{post}(B_{i-1}), \overline{pre}(B_i)$ we have $e < 0$, namely (9) is not in $\mathcal{BT}$ due to malformed syntax.

If some type $\square_e$ in $\overline{post}(B_1), \ldots, \overline{post}(B_{i-1})$ is such that $e < 0$, then (11) is valid, since $\phi(\overline{post}(B_1), \ldots, \overline{post}(B_{i-1}))$ has no solution (it includes the always false constraint $e \geq 0$).

Finally, consider the case that for all types $\square_e$ in $\overline{post}(B_1), \ldots, \overline{post}(B_{i-1})$, we have $e \geq 0$. This means that there is some type $\square_e$ in $\overline{pre}(B_i)$ such that $e < 0$. Since $c_1 \wedge c$ is satisfiable, we can extend one of its solutions to a solution of $\phi(\mathbf{d}) \wedge c_1 \wedge c$, where $\mathbf{d}$ is $\overline{pre}(H), \overline{post}(B_1), \ldots, \overline{post}(B_{i-1})$. This solution would show that $\mathbf{d} \vdash c_1 \wedge c \rightarrow \overline{pre}(B_i)$ is invalid (as a type assertion in $\mathcal{PT}$), which is absurd since such a type assertion is (9), which was shown to be valid.

(b) Since S is well-moded, for $i = 1..n$:

$$post(A_1), post(A_2), \ldots, post(A_{i-1}) \vdash c' \rightarrow pre(A_i) \tag{12}$$

is valid and has no local type variable. By recalling that no local to output type variable is admitted in parametric modes (see Definition 4.12), a simple induction on i reveals that there is no type variable at all. Also, by reasoning as in **(a)**:

$$\overline{post}(B_1), \ldots, \overline{post}(B_k) \vdash c_1 \wedge c \rightarrow \overline{post}(H)$$

is valid and contains no type variable, where $\overline{post}(H) = \mathbf{y} : \vartheta_1^s(\boldsymbol{\mu})$ for $H = p(\mathbf{y} : \boldsymbol{\tau} \times \boldsymbol{\mu})$. Since $A_1 = p(\mathbf{x} : \boldsymbol{\tau} \times \boldsymbol{\mu})$, $post(A_1) = \mathbf{x} : \vartheta_1^s(\boldsymbol{\mu})$, and $\mathbf{x} = \mathbf{y}$ is in c_1, validity of the last type assertion implies that:

$$\overline{post}(B_1), \ldots, \overline{post}(B_k) \vdash c_1 \wedge c \rightarrow post(A_1) \tag{13}$$

is valid and contains no type variable. Assume for the moment that the type assertions (12) and (13) are in $\mathcal{BT}$. First, by monotonicity on (12), for $i = 2..n$:

$$\overline{post}(B_1), \ldots, \overline{post}(B_k), post(A_1), post(A_2), \ldots, post(A_{i-1}) \vdash c_1 \wedge c \rightarrow pre(A_i) \tag{14}$$

is valid. Second, by monotonicity on (13), for $i = 2..n$:

$$\overline{post}(B_1), \ldots, \overline{post}(B_k), post(A_2), \ldots, post(A_{i-1}) \vdash c_1 \wedge c \rightarrow post(A_1) \tag{15}$$

is valid. By the transitivity Lemma 2.12, (15) and (14) imply, for $i = 2..n$:

$$\overline{post}(B_1), \ldots, \overline{post}(B_k), post(A_2), \ldots, post(A_{i-1}) \vdash c_1 \wedge c \rightarrow pre(A_i) \tag{16}$$

is valid. This concludes the proof, since this is the proof obligation of well-moding for type substitutions $\vartheta_1, \ldots, \vartheta_{k+n-1}$ and $i = k+1..k+n-1$.

Assume now that (12) or (13) is not in $\mathcal{BT}$. If $c_1 \wedge c$ is unsatisfiable then (16) is trivially valid. Let $c_1 \wedge c$ be satisfiable. Since (12) and (13) contain no type variable, this means that for some type $\square_e$ in (12) or in (13), we have $e < 0$.

If for some type $\square_e$ in $\overline{post}(B_1), \ldots, \overline{post}(B_k), post(A_2), \ldots, post(A_{i-1})$ we have $e < 0$, then (16) is trivially valid.

Also, $post(A_1)$ does not contain any $\square_e$ with $e < 0$. Otherwise, since $c_1 \wedge c$ is satisfiable, we can extend one of its solutions to a solution of $\phi(\mathbf{d}) \wedge c_1 \wedge c$, where $\mathbf{d}$ is $\overline{post}(B_1), \ldots, \overline{post}(B_k)$. This solution would show that $\mathbf{d} \vdash c_1 \wedge c \rightarrow post(A_1)$ is invalid (as a type assertion in $\mathcal{PT}$), which is absurd since such a type assertion is (13), which was shown to be valid.

Finally, with the same reasoning we can show that $pre(A_i)$ in (12) does not contain any $\square_e$ with $e < 0$. $\square$