Farkas' Lemma and Motzkin's Transposition Theorem*

Ralph Bottesch Max W. Haslbeck René Thiemann

March 17, 2025

Abstract

We formalize a proof of Motzkin's transposition theorem and Farkas' lemma in Isabelle/HOL. Our proof is based on the formalization of the simplex algorithm which, given a set of linear constraints, either returns a satisfying assignment to the problem or detects unsatisfiability. By reusing facts about the simplex algorithm we show that a set of linear constraints is unsatisfiable if and only if there is a linear combination of the constraints which evaluates to a trivially unsatisfiable inequality.

Contents

*Supported by FWF (Austrian Science Fund) project Y757. The authors are listed in alphabetical order regardless of individual contributions or seniority.

1 Introduction

This formalization augments the existing formalization of the simplex algorithm [3, 5, 7]. Given a system of linear constraints, the simplex implementation in [3] produces either a satisfying assignment or a subset of the given constraints that is itself unsatisfiable. Here we prove some variants of Farkas' Lemma. In essence, it states that if a set of constraints is unsatisfiable, then there is a linear combination of these constraints that evaluates to an unsatisfiable inequality of the form $0 \leq c$, for some negative c.

Our proof of Farkas' Lemma [4, Cor. 7.1e] relies on the formalized simplex algorithm: Under the assumption that the algorithm has detected unsatisfiability, we show that there exist coefficients for the above-mentioned linear combination of the input constraints.

Since the formalized algorithm follows the structure of the simplex-algorithm by Dutertre and de Moura [2], it first goes through a number of preprocessing phases, before starting the simplex procedure in earnest. These are relevant for proving Farkas' Lemma. We distinguish four *layers* of the algorithm; at each layer, it operates on data that is a refinement of the data available at the previous layer.

- *Layer 1. Data*: the input – a set of linear constraints with rational coefficients. These can be equalities or strict/non-strict inequalities. *Preprocessing*: Each equality is split into two non-strict inequalities, strict inequalities are replaced by non-strict inequalities involving δ-rationals.

- *Layer 2. Data*: a set of linear constraints that are non-strict inequalities with δ-rationals. *Preprocessing*: Linear constraints are simplified so that each constraint involves a single variable, by introducing so-called slack variables where necessary. The equations defining the slack variables are collected in a *tableau*. The constraints are normalized so that they are of the form $y \leq c$ or $y \geq c$ (these are called *atoms*).

- *Layer 3. Data*: A tableau and a set of atoms. Here the algorithm initializes the simplex algorithm.

- *Layer 4. Data*: A tableau, a set of atoms and an assignment of the variables. The simplex procedure is run.

At the point in the execution where the simplex algorithm detects unsatisfiability, we can directly obtain coefficients for the desired linear combination. However, these coefficients must then be propagated backwards through the different layers, where the constraints themselves have been modified, in order to obtain coefficients for a linear combination of *input* constraints. These propagation steps make up a large part of the formalized proof, since we must show, at each of the layers 1–3, that the existence of coefficients at the layer below translates into the existence of such coefficients for the current layer. This means, in particular, that we formulate and prove a version of Farkas' Lemma for each of the four layers, in terms of the data available at the respective level. The theorem we obtain at Layer 1 is actually a more general version of Farkas' lemma, in the sense that it allows for strict as well as non-strict inequalities, known as Motzkin's Transposition Theorem [4, Cor. 7.1k] or the Kuhn–Fourier Theorem [6, Thm. 1.1.9].

Since the implementation of the simplex algorithm in [3], which our work relies on, is restricted to systems of constraints over the rationals, this formalization is also subject to the same restriction.

2 Farkas Coefficients via the Simplex Algorithm of Duterte and de Moura

Let $c_1, \ldots, c_n$ be a finite list of linear inequalities. Let C be a list of pairs (r_i, c_i) where r_i is a rational number. We say that C is a list of *Farkas coefficients* if the sum of all products $r_i \cdot c_i$ results in an inequality that is trivially unsatisfiable.

Farkas' Lemma states that a finite set of non-strict linear inequalities is unsatisfiable if and only if Farkas coefficients exist. We will prove this lemma with the help of the simplex algorithm of Dutertre and de Moura's.

Note that the simplex implementation works on four layers, and we will formulate and prove a variant of Farkas' Lemma for each of these layers.

theory *Farkas*
  imports *Simplex.Simplex*
begin

2.1 Linear Inequalities

Both Farkas' Lemma and Motzkin's Transposition Theorem require linear combinations of inequalities. To this end we define one type that permits strict and non-strict inequalities which are always of the form "polynomial R constant" where R is either $\leq$ or $<$. On this type we can then define a commutative monoid.

A type for the two relations: less-or-equal and less-than.

datatype *le-rel* = *Leq-Rel* | *Lt-Rel*

primrec *rel-of* :: *le-rel* ⇒ *'a* :: *lrv* ⇒ *'a* ⇒ *bool* **where**
  rel-of Leq-Rel = (≤)
| *rel-of Lt-Rel* = (<)

instantiation *le-rel* :: *comm-monoid-add* **begin**
definition *zero-le-rel* = *Leq-Rel*

fun *plus-le-rel* **where**
  plus-le-rel Leq-Rel Leq-Rel = *Leq-Rel*
| *plus-le-rel* - - = *Lt-Rel*
instance
⟨*proof*⟩
end

lemma *Leq-Rel-0*: *Leq-Rel* = *0* ⟨*proof*⟩

datatype *'a le-constraint* = *Le-Constraint* (*lec-rel*: *le-rel*) (*lec-poly*: *linear-poly*) (*lec-const*: *'a*)

abbreviation (*input*) *Leqc* ≡ *Le-Constraint Leq-Rel*

instantiation *le-constraint* :: (*lrv*) *comm-monoid-add* **begin**
fun *plus-le-constraint* :: *'a le-constraint* ⇒ *'a le-constraint* ⇒ *'a le-constraint* **where**
  plus-le-constraint (*Le-Constraint r1 p1 c1*) (*Le-Constraint r2 p2 c2*) =
    (*Le-Constraint* (*r1* + *r2*) (*p1* + *p2*) (*c1* + *c2*))

definition *zero-le-constraint* :: *'a le-constraint* **where**
  zero-le-constraint = *Leqc 0 0*

instance ⟨*proof*⟩
end

primrec *satisfiable-le-constraint* :: *'a*::*lrv valuation* ⇒ *'a le-constraint* ⇒ *bool* (**infixl** ‹$\models_{le}$› *100*) **where**
  ($v \models_{le}$ (*Le-Constraint rel l r*)) ⟷ (*rel-of rel* ($l\{\!|v|\!\}$) *r*)

lemma *satisfies-zero-le-constraint*: $v \models_{le} 0$
  ⟨*proof*⟩

lemma *satisfies-sum-le-constraints*:
  assumes $v \models_{le} c$ $v \models_{le} d$
  shows $v \models_{le} (c + d)$
⟨*proof*⟩

lemma *satisfies-sumlist-le-constraints*:
  assumes ⋀ *c*. *c* ∈ *set* (*cs* :: *'a* :: *lrv le-constraint list*) ⟹ $v \models_{le} c$
  shows $v \models_{le}$ *sum-list cs*
  ⟨*proof*⟩

lemma *sum-list-lec*:
  sum-list ls = *Le-Constraint*
    (*sum-list* (*map lec-rel ls*))
    (*sum-list* (*map lec-poly ls*))
    (*sum-list* (*map lec-const ls*))
⟨*proof*⟩

lemma *sum-list-Leq-Rel*: $((\sum x{\leftarrow}C.\ \textit{lec-rel}\ (f\ x)) = \textit{Leq-Rel}) \longleftrightarrow (\forall\ x \in \textit{set}\ C.\ \textit{lec-rel}\ (f\ x) = \textit{Leq-Rel})$
⟨*proof*⟩

2.2 Farkas' Lemma on Layer 4

On layer 4 the algorithm works on a state containing a tableau, atoms (or bounds), an assignment and a satisfiability flag. Only non-strict inequalities appear at this level. In order to even state a variant of Farkas' Lemma on layer 4, we need conversions from atoms to non-strict constraints and then further to linear inequalities of type *le-constraint*. The latter conversion is a partial operation, since non-strict constraints of type *ns-constraint* permit greater-or-equal constraints, whereas *le-constraint* allows only less-or-equal.

The advantage of first going via *ns-constraint* is that this type permits a multiplication with arbitrary rational numbers (the direction of the inequality must be flipped when multiplying by a negative number, which is not possible with *le-constraint*).

instantiation *ns-constraint* :: (*scaleRat*) *scaleRat*
begin
fun *scaleRat-ns-constraint* :: *rat* ⇒ *'a ns-constraint* ⇒ *'a ns-constraint* **where**
  scaleRat-ns-constraint r (*LEQ-ns p c*) =
    (*if* ($r < 0$) *then GEQ-ns* (r *R p) (r *R c) *else LEQ-ns* (r *R p) (r *R c))
| *scaleRat-ns-constraint r* (*GEQ-ns p c*) =
    (*if* ($r > 0$) *then GEQ-ns* (r *R p) (r *R c) *else LEQ-ns* (r *R p) (r *R c))

instance ⟨*proof*⟩
end

lemma *sat-scale-rat-ns*: **assumes** $v \models_{ns} ns$
  shows $v \models_{ns} (f *R\ ns)$
⟨*proof*⟩

lemma *scaleRat-scaleRat-ns-constraint*: **assumes** $a \neq 0 \Longrightarrow b \neq 0$
  shows $a *R\ (b *R\ (c :: 'a :: \textit{lrv ns-constraint})) = (a * b) *R\ c$
⟨*proof*⟩

fun *lec-of-nsc* **where**
  lec-of-nsc (*LEQ-ns p c*) = (*Leqc p c*)

fun *is-leq-ns* **where**
  is-leq-ns (*LEQ-ns p c*) = *True*
| *is-leq-ns* (*GEQ-ns p c*) = *False*

lemma *lec-of-nsc*:
  assumes *is-leq-ns c*
  shows $(v \models_{le} \textit{lec-of-nsc}\ c) \longleftrightarrow (v \models_{ns} c)$
  ⟨*proof*⟩

fun *nsc-of-atom* **where**
  nsc-of-atom (*Leq var b*) = *LEQ-ns* (*lp-monom 1 var*) *b*
| *nsc-of-atom* (*Geq var b*) = *GEQ-ns* (*lp-monom 1 var*) *b*

lemma *nsc-of-atom*: $v \models_{ns}$ *nsc-of-atom a* $\longleftrightarrow v \models_a a$
  ⟨*proof*⟩

We say that C is a list of Farkas coefficients *for a given tableau t and atom set as*, if it is a list of pairs (r, a) such that $a \in as$, r is non-zero, $r \cdot a$ is a 'less-than-or-equal'-constraint, and the linear combination of inequalities must result in an inequality of the form $p \leq c$, where $c < 0$ and $t \models p = 0$.

definition *farkas-coefficients-atoms-tableau* **where**
  farkas-coefficients-atoms-tableau (*as* :: *'a* :: *lrv atom set*) *t C* = (∃ *p c*.
    (∀ (*r*,*a*) ∈ *set C*. *a* ∈ *as* ∧ *is-leq-ns* (*r* ∗*R* *nsc-of-atom a*) ∧ *r* ≠ *0*) ∧
    (∑ (*r*,*a*) ← *C*. *lec-of-nsc* (*r* ∗*R* *nsc-of-atom a*)) = *Leqc p c* ∧
    c < *0* ∧
    (∀ *v* :: *'a valuation*. $v \models_t t$ ⟶(*p*⦃*v*⦄ = *0*)))

We first prove that if the check-function detects a conflict, then Farkas coefficients do exist for the tableau and atom set for which the conflict is detected.

definition *bound-atoms* :: (*'i*, *'a*) *state* ⇒ *'a atom set* (‹$\mathcal{B}_A$›) **where**
  bound-atoms s = (λ(*v*,*x*). *Geq v x*) ‘ (*set-of-map* ($\mathcal{B}_l$ *s*)) ∪
                  (λ(*v*,*x*). *Leq v x*) ‘ (*set-of-map* ($\mathcal{B}_u$ *s*))

context *PivotUpdateMinVars*
begin

lemma *farkas-check*:
  assumes *check*: *check s'* = *s* **and** *U*: $\mathcal{U}$ *s* ¬ $\mathcal{U}$ *s'*
    and *inv*: ∇ *s'* △ ($\mathcal{T}$ *s'*) $\models_{nolhs}$ *s'* ◊ *s'*
    and *index*: *index-valid as s'*
  shows ∃ *C*. *farkas-coefficients-atoms-tableau* (*snd* ‘ *as*) ($\mathcal{T}$ *s'*) *C*
⟨*proof*⟩

end

Next, we show that a conflict found by the assert-bound function also gives rise to Farkas coefficients.

context *Update*
begin

lemma *farkas-assert-bound*: **assumes** *inv*: ¬ $\mathcal{U}$ *s* $\models_{nolhs}$ *s* △ ($\mathcal{T}$ *s*) ∇ *s* ◊ *s*
  and *index*: *index-valid as s*
  and *U*: $\mathcal{U}$ (*assert-bound ia s*)
shows ∃ *C*. *farkas-coefficients-atoms-tableau* (*snd* ‘ (*insert ia as*)) ($\mathcal{T}$ *s*) *C*
⟨*proof*⟩
end

Moreover, we prove that all other steps of the simplex algorithm on layer 4, such as pivoting, asserting bounds without conflict, etc., preserve Farkas coefficients.

lemma *farkas-coefficients-atoms-tableau-mono*: **assumes** *as* ⊆ *bs*
shows *farkas-coefficients-atoms-tableau as t C* ⟹ *farkas-coefficients-atoms-tableau bs t C*
⟨*proof*⟩

locale *AssertAllState'''* = *AssertAllState'' init ass-bnd chk* + *Update update* + *PivotUpdateMinVars eq-idx-for-lvar min-lvar-not-in-bounds min-rvar-incdec-eq pivot-and-update*
for *init* **and** *ass-bnd* :: *'i* × *'a* :: *lrv atom* ⇒ - **and** *chk* :: (*'i*, *'a*) *state* ⇒ (*'i*, *'a*) *state* **and** *update* :: *nat* ⇒ *'a* :: *lrv* ⇒ (*'i*, *'a*) *state* ⇒ (*'i*, *'a*) *state*
and *eq-idx-for-lvar* :: *tableau* ⇒ *var* ⇒ *nat* **and**
min-lvar-not-in-bounds :: (*'i*,*'a*::*lrv*) *state* ⇒ *var option* **and**
min-rvar-incdec-eq :: (*'i*,*'a*) *Direction* ⇒ (*'i*,*'a*) *state* ⇒ *eq* ⇒ *'i list* + *var* **and**
pivot-and-update :: *var* ⇒ *var* ⇒ *'a* ⇒ (*'i*,*'a*) *state* ⇒ (*'i*,*'a*) *state*
\+ **assumes** *ass-bnd*: *ass-bnd* = *Update.assert-bound update* **and**
chk: *chk* = *PivotUpdateMinVars.check eq-idx-for-lvar min-lvar-not-in-bounds min-rvar-incdec-eq pivot-and-update*

context *AssertAllState'''*
begin

lemma *farkas-assert-bound-loop*: **assumes** $\mathcal{U}$ (*assert-bound-loop as* (*init t*))
and *norm*: △ *t*
shows ∃ *C*. *farkas-coefficients-atoms-tableau* (*snd* ' *set as*) *t C*
⟨*proof*⟩

Now we get to the main result for layer 4: If the main algorithm returns unsat, then there are Farkas coefficients for the tableau and atom set that were given as input for this layer.

lemma *farkas-assert-all-state*: **assumes** *U*: $\mathcal{U}$ (*assert-all-state t as*)
and *norm*: △ *t*
shows ∃ *C*. *farkas-coefficients-atoms-tableau* (*snd* ' *set as*) *t C*
⟨*proof*⟩

2.3 Farkas' Lemma on Layer 3

There is only a small difference between layers 3 and 4, namely that there is no simplex algorithm (*assert-all-state*) on layer 3, but just a tableau and atoms.

Hence, one task is to link the unsatisfiability flag on layer 4 with unsatisfiability of the original tableau and atoms (layer 3). This can be done via the existing soundness results of the simplex algorithm. Moreover, we give an easy proof that the existence of Farkas coefficients for a tableau and set of atoms implies unsatisfiability.

end

lemma *farkas-coefficients-atoms-tableau-unsat*:
assumes *farkas-coefficients-atoms-tableau as t C*
shows $\nexists$ *v*. $v \models_t t \wedge v \models_{as} as$
⟨*proof*⟩

Next is the main result for layer 3: a tableau and a finite set of atoms are unsatisfiable if and only if there is a list of Farkas coefficients for the set of atoms and the tableau.

lemma *farkas-coefficients-atoms-tableau*: **assumes** *norm*: $\triangle$ *t*
and *fin*: *finite as*
shows ($\exists$ *C*. *farkas-coefficients-atoms-tableau as t C*) $\longleftrightarrow$ ($\nexists$ *v*. $v \models_t t \wedge v \models_{as}$ *as*)
⟨*proof*⟩

2.4 Farkas' Lemma on Layer 2

The main difference between layers 2 and 3 is the introduction of slack-variables in layer 3 via the preprocess-function. Our task here is to show that Farkas coefficients at layer 3 (where slack-variables are used) can be converted into Farkas coefficients for layer 2 (before the preprocessing).

We also need to adapt the previos notion of Farkas coefficients, which was used in *farkas-coefficients-atoms-tableau*, for layer 2. At layer 3, Farkas coefficients are the coefficients in a linear combination of atoms that evaluates to an inequality of the form $p \leq c$, where p is a linear polynomial, $c < 0$, and $t \models p = 0$ holds. At layer 2, the atoms are replaced by non-strict constraints where the left-hand side is a polynomial in the original variables, but the corresponding linear combination (with Farkas coefficients) evaluates directly to the inequality $0 \leq c$, with $c < 0$. The implication $t \models p = 0$ is no longer possible in this layer, since there is no tableau t, nor is it needed, since p is 0. Thus, the statement defining Farkas coefficients must be changed accordingly.

definition *farkas-coefficients-ns* **where**
farkas-coefficients-ns ns C = ($\exists$ *c*.
($\forall$ (*r*, *n*) $\in$ *set C*. *n* $\in$ *ns* $\wedge$ *is-leq-ns* (*r* $*R$ *n*) $\wedge$ *r* $\neq$ *0*) $\wedge$
($\sum$ (*r*, *n*) $\leftarrow$ *C*. *lec-of-nsc* (*r* $*R$ *n*)) = *Leqc 0 c* $\wedge$
c < *0*)

The easy part is to prove that Farkas coefficients imply unsatisfiability.

lemma *farkas-coefficients-ns-unsat*:
assumes *farkas-coefficients-ns ns C*
shows $\nexists$ *v*. $v \models_{nss}$ *ns*
⟨*proof*⟩

In order to eliminate the need for a tableau, we require the notion of an arbitrary substitution on polynomials, where all variables can be replaced at once. The existing simplex formalization provides only a function to replace one variable at a time.

definition *subst-poly* :: (*var* ⇒ *linear-poly*) ⇒ *linear-poly* ⇒ *linear-poly* **where**
subst-poly σ *p* = (∑ *x* ∈ *vars p*. *coeff p x* ∗*R* σ *x*)

lemma *subst-poly-0*[*simp*]: *subst-poly* σ *0* = *0* ⟨*proof*⟩

lemma *valuate-subst-poly*: (*subst-poly* σ *p*) ⦃ *v* ⦄ = (*p* ⦃ (λ *x*. ((σ *x*) ⦃ *v* ⦄)) ⦄)
⟨*proof*⟩

lemma *subst-poly-add*: *subst-poly* σ (*p* + *q*) = *subst-poly* σ *p* + *subst-poly* σ *q*
⟨*proof*⟩

fun *subst-poly-lec* :: (*var* ⇒ *linear-poly*) ⇒ *'a le-constraint* ⇒ *'a le-constraint*
where
subst-poly-lec σ (*Le-Constraint rel p c*) = *Le-Constraint rel* (*subst-poly* σ *p*) *c*

lemma *subst-poly-lec-0*[*simp*]: *subst-poly-lec* σ *0* = *0* ⟨*proof*⟩

lemma *subst-poly-lec-add*: *subst-poly-lec* σ (*c1* + *c2*) = *subst-poly-lec* σ *c1* +
subst-poly-lec σ *c2*
⟨*proof*⟩

lemma *subst-poly-lec-sum-list*: *subst-poly-lec* σ (*sum-list ps*) = *sum-list* (*map* (*subst-poly-lec*
σ) *ps*)
⟨*proof*⟩

lemma *subst-poly-lp-monom*[*simp*]: *subst-poly* σ (*lp-monom r x*) = *r* ∗*R* σ *x*
⟨*proof*⟩

lemma *subst-poly-scaleRat*: *subst-poly* σ (*r* ∗*R* *p*) = *r* ∗*R* (*subst-poly* σ *p*)
⟨*proof*⟩

We need several auxiliary properties of the preprocess-function which are not present in the simplex formalization.

lemma *Tableau-is-monom-preprocess'*:
assumes (*x*, *p*) ∈ *set* (*Tableau* (*preprocess' cs start*))
shows ¬ *is-monom p*
⟨*proof*⟩

lemma *preprocess'-atoms-to-constraints'*: **assumes** *preprocess' cs start* = *S*
shows *set* (*Atoms S*) ⊆ {(*i*,*qdelta-constraint-to-atom c v*) | *i c v*. (*i*,*c*) ∈ *set cs*
∧
(¬ *is-monom* (*poly c*) ⟶ *Poly-Mapping S* (*poly c*) = *Some v*)}
⟨*proof*⟩

lemma *monom-of-atom-coeff*:
assumes *is-monom* (*poly ns*) *a* = *qdelta-constraint-to-atom ns v*
shows (*monom-coeff* (*poly ns*)) ∗*R* *nsc-of-atom a* = *ns*

⟨*proof*⟩

The next lemma provides the functionality that is required to convert an atom back to a non-strict constraint, i.e., it is a kind of inverse of the preprocess-function.

lemma *preprocess′-atoms-to-constraints*: **assumes** *S*: *preprocess′ cs start* = *S*
and *start*: *start* = *start-fresh-variable cs*
and *ns*: *ns* = (*case a of Leq v c* ⇒ *LEQ-ns q c* | *Geq v c* ⇒ *GEQ-ns q c*)
and *a* ∈ *snd* ‘ *set* (*Atoms S*)
shows (*atom-var a* ∉ *fst* ‘ *set* (*Tableau S*) ⟶ (∃ *r*. *r* ≠ *0* ∧ *r* ∗*R nsc-of-atom a* ∈ *snd* ‘ *set cs*))
∧ ((*atom-var a*, *q*) ∈ *set* (*Tableau S*) ⟶ *ns* ∈ *snd* ‘ *set cs*)
⟨*proof*⟩

Next follows the major technical lemma of this part, namely that Farkas coefficients on layer 3 for preprocessed constraints can be converted into Farkas coefficients on layer 2.

lemma *farkas-coefficients-preprocess′*:
assumes *pp*: *preprocess′ cs* (*start-fresh-variable cs*) = *S* **and**
ft: *farkas-coefficients-atoms-tableau* (*snd* ‘ *set* (*Atoms S*)) (*Tableau S*) *C*
shows ∃ *C*. *farkas-coefficients-ns* (*snd* ‘ *set cs*) *C*
⟨*proof*⟩

lemma *preprocess′-unsat-indexD*: *i* ∈ *set* (*UnsatIndices* (*preprocess′ ns j*)) ⟹
∃ *c*. *poly c* = *0* ∧ ¬ *zero-satisfies c* ∧ (*i*,*c*) ∈ *set ns*
⟨*proof*⟩

lemma *preprocess′-unsat-index-farkas-coefficients-ns*:
assumes *i* ∈ *set* (*UnsatIndices* (*preprocess′ ns j*))
shows ∃ *C*. *farkas-coefficients-ns* (*snd* ‘ *set ns*) *C*
⟨*proof*⟩

The combination of the previous results easily provides the main result of this section: a finite set of non-strict constraints on layer 2 is unsatisfiable if and only if there are Farkas coefficients. Again, here we use results from the simplex formalization, namely soundness of the preprocess-function.

lemma *farkas-coefficients-ns*: **assumes** *finite* (*ns* :: *QDelta ns-constraint set*)
shows (∃ *C*. *farkas-coefficients-ns ns C*) ⟷ (∄ *v*. *v* $\models_{nss}$ *ns*)
⟨*proof*⟩

2.5 Farkas' Lemma on Layer 1

The main difference of layers 1 and 2 is the restriction to non-strict constraints via delta-rationals. Since we now work with another constraint type, *constraint*, we again need translations into linear inequalities of type *le-constraint*. Moreover, we also need to define scaling of constraints where flipping the comparison sign may be required.

fun *is-le* :: *constraint* ⇒ *bool* **where**

is-le (*LT* - -) = *True*
| *is-le* (*LEQ* - -) = *True*
| *is-le* - = *False*

fun *lec-of-constraint* **where**
lec-of-constraint (*LEQ p c*) = (*Le-Constraint Leq-Rel p c*)
| *lec-of-constraint* (*LT p c*) = (*Le-Constraint Lt-Rel p c*)

lemma *lec-of-constraint*:
assumes *is-le c*
shows ($v \models_{le}$ (*lec-of-constraint c*)) $\longleftrightarrow$ ($v \models_c c$)
⟨*proof*⟩

instantiation *constraint* :: *scaleRat*
begin
fun *scaleRat-constraint* :: *rat* ⇒ *constraint* ⇒ *constraint* **where**
scaleRat-constraint r cc = (*if r* = *0 then LEQ 0 0 else*
(*case cc of*
LEQ p c ⇒
(*if* (*r* < *0*) *then GEQ* (*r* ∗*R p*) (*r* ∗*R c*) *else LEQ* (*r* ∗*R p*) (*r* ∗*R c*))
| *LT p c* ⇒
(*if* (*r* < *0*) *then GT* (*r* ∗*R p*) (*r* ∗*R c*) *else LT* (*r* ∗*R p*) (*r* ∗*R c*))
| *GEQ p c* ⇒
(*if* (*r* > *0*) *then GEQ* (*r* ∗*R p*) (*r* ∗*R c*) *else LEQ* (*r* ∗*R p*) (*r* ∗*R c*))
| *GT p c* ⇒
(*if* (*r* > *0*) *then GT* (*r* ∗*R p*) (*r* ∗*R c*) *else LT* (*r* ∗*R p*) (*r* ∗*R c*))
| *EQ p c* ⇒ *LEQ* (*r* ∗*R p*) (*r* ∗*R c*) — We do not keep equality, since the aim is to convert the scaled constraints into inequalities, which will then be summed up.
))

instance ⟨*proof*⟩
end

lemma *sat-scale-rat*: **assumes** (v :: *rat valuation*) $\models_c c$
shows $v \models_c$ (*r* ∗*R c*)
⟨*proof*⟩

In the following definition of Farkas coefficients (for layer 1), the main difference to *farkas-coefficients-ns* is that the linear combination evaluates either to a strict inequality where the constant must be non-positive, or to a non-strict inequality where the constant must be negative.

definition *farkas-coefficients* **where**
farkas-coefficients cs C = (∃ *d rel*.
(∀ (*r*,*c*) ∈ *set C*. *c* ∈ *cs* ∧ *is-le* (*r* ∗*R c*) ∧ *r* ≠ *0*) ∧
(∑ (*r*,*c*) ← *C*. *lec-of-constraint* (*r* ∗*R c*)) = *Le-Constraint rel 0 d* ∧
(*rel* = *Leq-Rel* ∧ *d* < *0* ∨ *rel* = *Lt-Rel* ∧ *d* ≤ *0*))

Again, the existence Farkas coefficients immediately implies unsatisfiability.

lemma *farkas-coefficients-unsat*:
assumes *farkas-coefficients cs C*
shows $\nexists$ *v*. *v* $\models_{cs}$ *cs*
⟨*proof*⟩

Now follows the difficult implication. The major part is proving that the translation *constraint-to-qdelta-constraint* preserves the existence of Farkas coefficients via pointwise compatibility of the sum. Here, compatibility links a strict or non-strict inequality from the input constraint to a translated non-strict inequality over delta-rationals.

fun *compatible-cs* **where**
compatible-cs (*Le-Constraint Leq-Rel p c*) (*Le-Constraint Leq-Rel q d*) = (*q* = *p* ∧ *d* = *QDelta c 0*)
| *compatible-cs* (*Le-Constraint Lt-Rel p c*) (*Le-Constraint Leq-Rel q d*) = (*q* = *p* ∧ *qdfst d* = *c*)
| *compatible-cs* - - = *False*

lemma *compatible-cs-0-0*: *compatible-cs 0 0* ⟨*proof*⟩

lemma *compatible-cs-plus*: *compatible-cs c1 d1* $\Longrightarrow$ *compatible-cs c2 d2* $\Longrightarrow$ *compatible-cs* (*c1* + *c2*) (*d1* + *d2*)
⟨*proof*⟩

lemma *unsat-farkas-coefficients*: **assumes** $\nexists$ *v*. *v* $\models_{cs}$ *cs*
and *fin*: *finite cs*
shows ∃ *C*. *farkas-coefficients cs C*
⟨*proof*⟩

Finally we can prove on layer 1 that a finite set of constraints is unsatisfiable if and only if there are Farkas coefficients.

lemma *farkas-coefficients*: **assumes** *finite cs*
shows (∃ *C*. *farkas-coefficients cs C*) ⟷ ($\nexists$ *v*. *v* $\models_{cs}$ *cs*)
⟨*proof*⟩

3 Corollaries from the Literature

In this section, we convert the previous variations of Farkas' Lemma into more well-known forms of this result. Moreover, instead of referring to the various constraint types of the simplex formalization, we now speak solely about constraints of type *le-constraint*.

3.1 Farkas' Lemma on Delta-Rationals

We start with Lemma 2 of [1], a variant of Farkas' Lemma for delta-rationals. To be more precise, it states that a set of non-strict inequalities over delta-rationals is unsatisfiable if and only if there is a linear combination of the inequalities that results in a trivial unsatisfiable constraint $0 < const$ for some negative constant *const*. We can easily prove this statement via the lemma *farkas-coefficients-ns* and some conversions between the different constraint types.

lemma *Farkas′-Lemma-Delta-Rationals*: **fixes** *cs* :: *QDelta le-constraint set*
assumes *only-non-strict*: *lec-rel* ‘ *cs* $\subseteq$ {*Leq-Rel*}
and *fin*: *finite cs*
shows ($\nexists$ *v*. $\forall$ *c* $\in$ *cs*. *v* $\models_{le}$ *c*) $\longleftrightarrow$
($\exists$ *C const*. ($\forall$ (*r*, *c*) $\in$ *set C*. *r* > *0* $\wedge$ *c* $\in$ *cs*)
$\wedge$ ($\sum$ (*r*,*c*) $\leftarrow$ *C*. *Leqc* (*r* $*R$ *lec-poly c*) (*r* $*R$ *lec-const c*)) = *Leqc 0 const*
$\wedge$ *const* < *0*)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

3.2 Motzkin's Transposition Theorem or the Kuhn-Fourier Theorem

Next, we prove a generalization of Farkas' Lemma that permits arbitrary combinations of strict and non-strict inequalities: Motzkin's Transposition Theorem which is also known as the Kuhn–Fourier Theorem.

The proof is mainly based on the lemma *farkas-coefficients*, again requiring conversions between constraint types.

theorem *Motzkin′s-transposition-theorem*: **fixes** *cs* :: *rat le-constraint set*
assumes *fin*: *finite cs*
shows ($\nexists$ *v*. $\forall$ *c* $\in$ *cs*. *v* $\models_{le}$ *c*) $\longleftrightarrow$
($\exists$ *C const rel*. ($\forall$ (*r*, *c*) $\in$ *set C*. *r* > *0* $\wedge$ *c* $\in$ *cs*)
$\wedge$ ($\sum$ (*r*,*c*) $\leftarrow$ *C*. *Le-Constraint* (*lec-rel c*) (*r* $*R$ *lec-poly c*) (*r* $*R$ *lec-const c*))
= *Le-Constraint rel 0 const*
$\wedge$ (*rel* = *Leq-Rel* $\wedge$ *const* < *0* $\vee$ *rel* = *Lt-Rel* $\wedge$ *const* $\leq$ *0*))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

3.3 Farkas' Lemma

Finally we derive the commonly used form of Farkas' Lemma, which easily follows from *Motzkin′s-transposition-theorem*. It only permits non-strict inequalities and, as a result, the sum of inequalities will always be non-strict.

lemma *Farkas′-Lemma*: **fixes** *cs* :: *rat le-constraint set*
assumes *only-non-strict*: *lec-rel* ‘ *cs* $\subseteq$ {*Leq-Rel*}
and *fin*: *finite cs*
shows ($\nexists$ *v*. $\forall$ *c* $\in$ *cs*. *v* $\models_{le}$ *c*) $\longleftrightarrow$
($\exists$ *C const*. ($\forall$ (*r*, *c*) $\in$ *set C*. *r* > *0* $\wedge$ *c* $\in$ *cs*)
$\wedge$ ($\sum$ (*r*,*c*) $\leftarrow$ *C*. *Leqc* (*r* $*R$ *lec-poly c*) (*r* $*R$ *lec-const c*)) = *Leqc 0 const*
$\wedge$ *const* < *0*)
(**is** - = *?rhs*)
⟨*proof*⟩

We also present slightly modified versions

lemma *sum-list-map-filter-sum*: **fixes** f :: $'a \Rightarrow 'b$ :: *comm-monoid-add*
shows *sum-list* (*map f* (*filter g xs*)) + *sum-list* (*map f* (*filter* (*Not o g*) *xs*)) = *sum-list* (*map f xs*)
⟨*proof*⟩

A version where every constraint obtains exactly one coefficient and where 0 coefficients are allowed.

lemma *Farkas'-Lemma-set-sum*: **fixes** *cs* :: *rat le-constraint set*
assumes *only-non-strict*: *lec-rel* ‘ *cs* ⊆ {*Leq-Rel*}
and *fin*: *finite cs*
shows ($\nexists$ *v*. ∀ *c* ∈ *cs*. $v \models_{le} c$) ⟷
(∃ *C const*. (∀ *c* ∈ *cs*. *C c* ≥ *0*)
∧ (∑ *c* ∈ *cs*. *Leqc* ((*C c*) ∗*R lec-poly c*) ((*C c*) ∗*R lec-const c*)) = *Leqc 0 const*
∧ *const* < *0*)
⟨*proof*⟩

A version with indexed constraints, i.e., in particular where constraints may occur several times.

lemma *Farkas'-Lemma-indexed*: **fixes** *c* :: *nat* ⇒ *rat le-constraint*
assumes *only-non-strict*: *lec-rel* ‘ *c* ‘ *Is* ⊆ {*Leq-Rel*}
and *fin*: *finite Is*
shows ($\nexists$ *v*. ∀ *i* ∈ *Is*. $v \models_{le} c\ i$) ⟷
(∃ *C const*. (∀ *i* ∈ *Is*. *C i* ≥ *0*)
∧ (∑ *i* ∈ *Is*. *Leqc* ((*C i*) ∗*R lec-poly* (*c i*)) ((*C i*) ∗*R lec-const* (*c i*))) = *Leqc 0 const*
∧ *const* < *0*)
⟨*proof*⟩

end

3.4 Farkas Lemma for Matrices

In this part we convert the simplex-structures like linear polynomials, etc., into equivalent formulations using matrices and vectors. As a result we present Farkas' Lemma via matrices and vectors.

theory *Matrix-Farkas*
imports *Farkas*
Jordan-Normal-Form.Matrix
begin

lift-definition *poly-of-vec* :: *rat vec* ⇒ *linear-poly* **is**
λ *v x*. *if* (*x* < *dim-vec v*) *then v* $ *x else 0*
⟨*proof*⟩

definition *val-of-vec* :: *rat vec* ⇒ *rat valuation* **where**

val-of-vec v x = v \$ x

lemma *valuate-poly-of-vec*: **assumes** $w \in$ *carrier-vec* n
and $v \in$ *carrier-vec* n
shows *valuate* (*poly-of-vec* v) (*val-of-vec* w) = $v \cdot w$
⟨*proof*⟩

definition *constraints-of-mat-vec* :: *rat mat* ⇒ *rat vec* ⇒ *rat le-constraint set*
where
constraints-of-mat-vec A b = (λ i . *Leqc* (*poly-of-vec* (*row* A i)) (b \$ i)) ‘ {*0* ..< *dim-row* A}

lemma *constraints-of-mat-vec-solution-main*: **assumes** A: $A \in$ *carrier-mat nr nc*
and x: $x \in$ *carrier-vec nc*
and b: $b \in$ *carrier-vec nr*
and *sol*: $A *_v x \leq b$
and c: $c \in$ *constraints-of-mat-vec* A b
shows *val-of-vec* $x \models_{le} c$
⟨*proof*⟩

lemma *vars-poly-of-vec*: *vars* (*poly-of-vec* v) ⊆ { *0* ..< *dim-vec* v}
⟨*proof*⟩

lemma *finite-constraints-of-mat-vec*: *finite* (*constraints-of-mat-vec* A b)
⟨*proof*⟩

lemma *lec-rec-constraints-of-mat-vec*: *lec-rel* ‘ *constraints-of-mat-vec* A b ⊆ {*Leq-Rel*}

⟨*proof*⟩

lemma *constraints-of-mat-vec-solution-1*:
assumes A: $A \in$ *carrier-mat nr nc*
and b: $b \in$ *carrier-vec nr*
and *sol*: ∃ $x \in$ *carrier-vec nc*. $A *_v x \leq b$
shows ∃ v. ∀ $c \in$ *constraints-of-mat-vec* A b. $v \models_{le} c$
⟨*proof*⟩

lemma *constraints-of-mat-vec-solution-2*:
assumes A: $A \in$ *carrier-mat nr nc*
and b: $b \in$ *carrier-vec nr*
and *sol*: ∃ v. ∀ $c \in$ *constraints-of-mat-vec* A b. $v \models_{le} c$
shows ∃ $x \in$ *carrier-vec nc*. $A *_v x \leq b$
⟨*proof*⟩

lemma *constraints-of-mat-vec-solution*:
assumes A: $A \in$ *carrier-mat nr nc*
and b: $b \in$ *carrier-vec nr*
shows (∃ $x \in$ *carrier-vec nc*. $A *_v x \leq b$) =
(∃ v. ∀ $c \in$ *constraints-of-mat-vec* A b. $v \models_{le} c$)

$\langle proof \rangle$

lemma *farkas-lemma-matrix*: **fixes** A :: *rat mat*
assumes A: $A \in$ *carrier-mat nr nc*
and b: $b \in$ *carrier-vec nr*
shows $(\exists\ x \in \textit{carrier-vec nc}.\ A *_v x \leq b) \longleftrightarrow$
$(\forall\ y.\ y \geq 0_v\ nr \longrightarrow \textit{mat-of-row}\ y * A = 0_m\ 1\ nc \longrightarrow y \cdot b \geq 0)$
$\langle proof \rangle$

lemma *farkas-lemma-matrix'*: **fixes** A :: *rat mat*
assumes A: $A \in$ *carrier-mat nr nc*
and b: $b \in$ *carrier-vec nr*
shows $(\exists\ x \geq 0_v\ nc.\ A *_v x = b) \longleftrightarrow$
$(\forall\ y \in \textit{carrier-vec nr}.\ \textit{mat-of-row}\ y * A \geq 0_m\ 1\ nc \longrightarrow y \cdot b \geq 0)$
$\langle proof \rangle$

end

4 Unsatisfiability over the Reals

By using Farkas' Lemma we prove that a finite set of linear rational inequalities is satisfiable over the rational numbers if and only if it is satisfiable over the real numbers. Hence, the simplex algorithm either gives a rational solution or shows unsatisfiability over the real numbers.

theory *Simplex-for-Reals*
imports
Farkas
Simplex.Simplex-Incremental
begin

instantiation *real* :: *lrv*
begin
definition *scaleRat-real* :: *rat* ⇒ *real* ⇒ *real* **where**
[*simp*]: $x *R\ y = \textit{real-of-rat}\ x * y$
instance $\langle proof \rangle$
end

abbreviation *real-satisfies-constraints* :: *real valuation* ⇒ *constraint set* ⇒ *bool*
(**infixl** ‹$\models_{rcs}$› *100*) **where**
$v \models_{rcs} cs \equiv \forall\ c \in cs.\ v \models_c c$

definition *of-rat-val* :: *rat valuation* ⇒ *real valuation* **where**
of-rat-val v x = *of-rat* (*v x*)

lemma *of-rat-val-eval*: *p* ⦃*of-rat-val v*⦄ = *of-rat* (*p* ⦃*v*⦄)
$\langle proof \rangle$

lemma *of-rat-val-constraint*: *of-rat-val v* $\models_c$ *c* ⟷ *v* $\models_c$ *c*
⟨*proof*⟩

lemma *of-rat-val-constraints*: *of-rat-val v* $\models_{rcs}$ *cs* ⟷ *v* $\models_{cs}$ *cs*
⟨*proof*⟩

lemma *sat-scale-rat-real*: **assumes** (*v* :: *real valuation*) $\models_c$ *c*
shows *v* $\models_c$ (*r* ∗*R c*)
⟨*proof*⟩

fun *of-rat-lec* :: *rat le-constraint* ⇒ *real le-constraint* **where**
of-rat-lec (*Le-Constraint r p c*) = *Le-Constraint r p* (*of-rat c*)

lemma *lec-of-constraint-real*:
assumes *is-le c*
shows (*v* $\models_{le}$ *of-rat-lec* (*lec-of-constraint c*)) ⟷ (*v* $\models_c$ *c*)
⟨*proof*⟩

lemma *of-rat-lec-add*: *of-rat-lec* (*c* + *d*) = *of-rat-lec c* + *of-rat-lec d*
⟨*proof*⟩

lemma *of-rat-lec-zero*: *of-rat-lec 0* = *0*
⟨*proof*⟩

lemma *of-rat-lec-sum*: *of-rat-lec* (*sum-list c*) = *sum-list* (*map of-rat-lec c*)
⟨*proof*⟩

This is the main lemma: a finite set of linear constraints is satisfiable over Q if and only if it is satisfiable over R.

lemma *rat-real-conversion*: **assumes** *finite cs*
shows (∃ *v* :: *rat valuation*. *v* $\models_{cs}$ *cs*) ⟷ (∃ *v* :: *real valuation*. *v* $\models_{rcs}$ *cs*)
⟨*proof*⟩

The main result of simplex, now using unsatisfiability over the reals.

fun *i-satisfies-cs-real* (**infixl** ‹$\models_{rics}$› *100*) **where**
(*I*,*v*) $\models_{rics}$ *cs* ⟷ *v* $\models_{rcs}$ *Simplex.restrict-to I cs*

lemma *simplex-index-real*:
simplex-index cs = *Unsat I* ⟹ *set I* ⊆ *fst* ‘ *set cs* ∧ ¬ (∃ *v*. (*set I*, *v*) $\models_{rics}$ *set cs*) ∧
(*distinct-indices cs* ⟶ (∀ *J* ⊂ *set I*. (∃ *v*. (*J*, *v*) $\models_{ics}$ *set cs*))) — minimal unsat core over the reals
simplex-index cs = *Sat v* ⟹ ⟨*v*⟩ $\models_{cs}$ (*snd* ‘ *set cs*) — satisfying assingment
⟨*proof*⟩

lemma *simplex-real*:
simplex cs = *Unsat I* ⟹ ¬ (∃ *v*. *v* $\models_{rcs}$ *set cs*) — unsat of original constraints over the reals

$simplex\ cs = Unsat\ I \Longrightarrow set\ I \subseteq \{0..<length\ cs\} \wedge \neg (\exists\ v.\ v \models_{rcs} \{cs\ !\ i \mid i.\ i \in set\ I\})$
$\wedge (\forall J \subset set\ I.\ \exists v.\ v \models_{cs} \{cs\ !\ i \mid i.\ i \in J\})$ — minimal unsat core over reals
$simplex\ cs = Sat\ v \Longrightarrow \langle v\rangle \models_{cs} set\ cs$ — satisfying assignment over the rationals
⟨*proof*⟩

Define notion of minimal unsat core over the reals: the subset has to be unsat over the reals, and every proper subset has to be satisfiable over the rational numbers.

definition *minimal-unsat-core-real* :: *'i set* ⇒ *'i i-constraint list* ⇒ *bool* **where**
$minimal\text{-}unsat\text{-}core\text{-}real\ I\ ics\ = ((I \subseteq fst\ `\ set\ ics) \wedge (\neg (\exists\ v.\ (I,v) \models_{rics} set\ ics))$
$\wedge (distinct\text{-}indices\ ics \longrightarrow (\forall\ J.\ J \subset I \longrightarrow (\exists\ v.\ (J,v) \models_{ics} set\ ics))))$

Because of equi-satisfiability the two notions of minimal unsat cores coincide.

lemma *minimal-unsat-core-real-conv*: *minimal-unsat-core-real I ics* = *minimal-unsat-core I ics*
⟨*proof*⟩

Easy consequence: The incremental simplex algorithm is also sound wrt. minimal-unsat-cores over the reals.

lemmas *incremental-simplex-real* =
init-simplex
assert-simplex-ok
assert-simplex-unsat[*folded minimal-unsat-core-real-conv*]
assert-all-simplex-ok
assert-all-simplex-unsat[*folded minimal-unsat-core-real-conv*]
check-simplex-ok
check-simplex-unsat[*folded minimal-unsat-core-real-conv*]
solution-simplex
backtrack-simplex
checked-invariant-simplex

end

References

[1] M. Bromberger and C. Weidenbach. New techniques for linear arithmetic: cubes and equalities. *Formal Methods in System Design*, 51(3):433–461, Dec 2017.

[2] B. Dutertre and L. M. de Moura. A fast linear-arithmetic solver for DPLL(T). In T. Ball and R. B. Jones, editors, *CAV'06*, volume 4144 of *LNCS*, pages 81–94, 2006.

[3] F. Marić, M. Spasić, and R. Thiemann. An incremental simplex algorithm with unsatisfiable core generation. *Archive of Formal Proofs*,

Aug. 2018. http://isa-afp.org/entries/Simplex.html, Formal proof development.

[4] A. Schrijver. *Theory of linear and integer programming.* Wiley, 1999.

[5] M. Spasić and F. Marić. Formalization of incremental simplex algorithm by stepwise refinement. In D. Giannakopoulou and D. Méry, editors, *FM'12*, volume 7436 of *LNCS*, pages 434–449, 2012.

[6] J. Stoer and C. Witzgall. *Convexity and Optimization in Finite Dimensions I.* Die Grundlehren der mathematischen Wissenschaften 163. Springer-Verlag Berlin Heidelberg, 1 edition, 1970.

[7] R. Thiemann. Extending a verified simplex algorithm. In G. Barthe, K. Korovin, S. Schulz, M. Suda, G. Sutcliffe, and M. Veanes, editors, *LPAR-22 Workshop and Short Paper Proceedings*, volume 9 of *Kalpa Publications in Computing*, pages 37–48. EasyChair, 2018.