

Farkas' Lemma and Motzkin's Transposition Theorem*

Ralph Bottesch Max W. Haslbeck René Thiemann

June 19, 2024

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

1	Introduction	2
2	Farkas Coefficients via the Simplex Algorithm of Duterte and de Moura	3
2.1	Linear Inequalities	3
2.2	Farkas' Lemma on Layer 4	5
2.3	Farkas' Lemma on Layer 3	16
2.4	Farkas' Lemma on Layer 2	17
2.5	Farkas' Lemma on Layer 1	26
3	Corollaries from the Literature	30
3.1	Farkas' Lemma on Delta-Rationals	30
3.2	Motzkin's Transposition Theorem or the Kuhn-Fourier Theorem	32
3.3	Farkas' Lemma	34
3.4	Farkas Lemma for Matrices	38
4	Unsatisfiability over the Reals	45

*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, \dots, 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 Duterte 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  $\Rightarrow$  'a :: lrv  $\Rightarrow$  'a  $\Rightarrow$  bool where
  rel-of Leq-Rel = ( $\leq$ )
| 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
  fix a b c :: le-rel
  show a + b + c = a + (b + c) by (cases a; cases b; cases c, auto)
  show a + b = b + a by (cases a; cases b, auto)
  show 0 + a = a unfolding zero-le-rel-def by (cases a, auto)
qed
end

lemma Leq-Rel-0: Leq-Rel = 0 unfolding zero-le-rel-def by simp

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
  fix a b c :: 'a le-constraint
  show 0 + a = a
  by (cases a, auto simp: zero-le-constraint-def Leq-Rel-0)
  show a + b = b + a by (cases a; cases b, auto simp: ac-simps)
  show a + b + c = a + (b + c) by (cases a; cases b; cases c, auto simp: ac-simps)
qed
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))  $\longleftrightarrow$  (rel-of rel (l  $\{v\}$ ) r)

lemma satisfies-zero-le-constraint: v  $\models_{le}$  0
  by (simp add: valuate-zero zero-le-constraint-def)

lemma satisfies-sum-le-constraints:
  assumes v  $\models_{le}$  c v  $\models_{le}$  d
  shows v  $\models_{le}$  (c + d)
proof –
  obtain lc rc ld rd rel1 rel2 where cd: c = Le-Constraint rel1 lc rc d = Le-Constraint
  rel2 ld rd

```

```

  by (cases c; cases d, auto)
  have 1: rel-of rel1 (lc{v}) rc using assms cd by auto
  have 2: rel-of rel2 (ld{v}) rd using assms cd by auto
  from 1 have le1: lc{v} ≤ rc by (cases rel1, auto)
  from 2 have le2: ld{v} ≤ rd by (cases rel2, auto)
  from 1 2 le1 le2 have rel-of (rel1 + rel2) ((lc{v}) + (ld{v})) (rc + rd)
    apply (cases rel1; cases rel2; simp add: add-mono)
    by (metis add.commute le-less-trans order.strict-iff-order plus-less)+
  thus ?thesis by (auto simp: cd valuate-add)
qed

```

lemma *satisfies-sumlist-le-constraints*:

```

  assumes  $\bigwedge c. c \in \text{set } (cs :: 'a :: \text{lrV le-constraint list}) \implies v \models_{le} c$ 
  shows  $v \models_{le} \text{sum-list } cs$ 
  using assms
  by (induct cs, auto intro: satisfies-zero-le-constraint satisfies-sum-le-constraints)

```

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 (induct ls)

case Nil

show ?case by (auto simp: zero-le-constraint-def Leq-Rel-0)

next

case (Cons l ls)

show ?case by (cases l, auto simp: Cons)

qed

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

proof (induct C)

case (Cons c C)

show ?case

proof (cases lec-rel (f c))

case Leq-Rel

show ?thesis using Cons by (simp add: Leq-Rel Leq-Rel-0)

qed simp

qed (simp add: Leq-Rel-0)

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 \Rightarrow 'a *ns-constraint* \Rightarrow '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 ..

end

lemma *sat-scale-rat-ns*: **assumes** v \models_{ns} ns

shows v \models_{ns} (f *R ns)

proof –

have f < 0 | f = 0 | f > 0 **by** *auto*

then show ?thesis **using** *assms* **by** (*cases ns, auto simp: valuate-scaleRat scaleRat-leq1 scaleRat-leq2*)

qed

lemma *scaleRat-scaleRat-ns-constraint*: **assumes** a \neq 0 \implies b \neq 0

shows a *R (b *R (c :: 'a :: lrv *ns-constraint*)) = (a * b) *R c

proof –

have b > 0 \vee b < 0 \vee b = 0 **by** *linarith*

moreover have a > 0 \vee a < 0 \vee a = 0 **by** *linarith*

ultimately show ?thesis **using** *assms*

by (*elim disjE; cases c, auto simp add: not-le not-less*

mult-neg-pos mult-neg-neg mult-nonpos-nonneg mult-nonpos-nonpos mult-nonneg-nonpos mult-pos-neg)

qed

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} *lec-of-nsc* c) \longleftrightarrow (v \models_{ns} c)

using *assms* **by** (*cases c, auto*)

fun *nsc-of-atom* **where**

$$\begin{aligned} nsc\text{-of-atom } (Leq \text{ var } b) &= LEQ\text{-ns } (lp\text{-monom } 1 \text{ var}) b \\ | nsc\text{-of-atom } (Geq \text{ var } b) &= GEQ\text{-ns } (lp\text{-monom } 1 \text{ var}) b \end{aligned}$$

lemma *nsc-of-atom*: $v \models_{ns} nsc\text{-of-atom } a \longleftrightarrow v \models_a a$
by (*cases a, auto*)

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**

$$\begin{aligned} farkas\text{-coefficients-atoms-tableau } (as :: 'a :: lrv \text{ atom set}) t C &= (\exists p c. \\ &(\forall (r, a) \in set C. a \in as \wedge is\text{-leq-ns } (r *R nsc\text{-of-atom } a) \wedge r \neq 0) \wedge \\ &(\sum (r, a) \leftarrow C. lec\text{-of-nsc } (r *R nsc\text{-of-atom } a)) = Leqc p c \wedge \\ &c < 0 \wedge \\ &(\forall v :: 'a \text{ valuation. } v \models_t t \longrightarrow (p\{v\} = 0))) \end{aligned}$$

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* :: $('i, 'a) \text{ state} \Rightarrow 'a \text{ atom set } (\mathcal{B}_A)$ **where**

$$\begin{aligned} bound\text{-atoms } s &= (\lambda(v, x). Geq v x) \text{ ‘(set-of-map } (\mathcal{B}_l s)) \cup \\ &(\lambda(v, x). Leq v x) \text{ ‘(set-of-map } (\mathcal{B}_u s)) \end{aligned}$$

context *PivotUpdateMinVars*
begin

lemma *farkas-check*:

$$\begin{aligned} \text{assumes } check: check \ s' = s \text{ and } U: \mathcal{U} \ s \neg \mathcal{U} \ s' \\ \text{and } inv: \nabla s' \Delta (\mathcal{T} \ s') \models_{no\text{lhs}} s' \diamond s' \\ \text{and } index: index\text{-valid as } s' \\ \text{shows } \exists C. farkas\text{-coefficients-atoms-tableau } (snd \text{ ‘ } as) (\mathcal{T} \ s') C \end{aligned}$$

proof –

$$\begin{aligned} \text{let } ?Q &= \lambda s f p c C. set C \subseteq \mathcal{B}_A \ s \wedge \\ &distinct C \wedge \\ &(\forall a \in set C. is\text{-leq-ns } (f (atom\text{-var } a) *R nsc\text{-of-atom } a) \wedge f (atom\text{-var } a) \neq \\ 0) \wedge \\ &(\sum a \leftarrow C. lec\text{-of-nsc } (f (atom\text{-var } a) *R nsc\text{-of-atom } a)) = Leqc p c \wedge \\ &c < 0 \wedge \\ &(\forall v :: 'a \text{ valuation. } v \models_t \mathcal{T} \ s \longrightarrow (p\{v\} = 0)) \\ \text{let } ?P &= \lambda s. \mathcal{U} \ s \longrightarrow (\exists f p c C. ?Q \ s f p c C) \\ \text{have } ?P &(check \ s') \\ \text{proof } (induct \text{ rule: } check\text{-induct''}[OF \ inv, \text{ of } ?P]) \\ \text{case } (\exists s x_i \text{ dir } I) \\ \text{have } dir: dir &= Positive \vee dir = Negative \text{ by fact} \\ \text{let } ?eq &= (eq\text{-for-lvar } (\mathcal{T} \ s) x_i) \\ \text{define } X_j &\text{ where } X_j = rvars\text{-eq } ?eq \\ \text{define } XL_j &\text{ where } XL_j = Abstract\text{-Linear-Poly.vars-list } (rhs \ ?eq) \\ \text{have } [simp]: set XL_j &= X_j \text{ unfolding } XL_j\text{-def } X_j\text{-def} \end{aligned}$$

```

    using set-vars-list by blast
  have  $XL_j$ -distinct: distinct  $XL_j$ 
    unfolding  $XL_j$ -def using distinct-vars-list by simp
  define  $A$  where  $A = \text{coeff } (rhs \ ?eq)$ 
    have bounds-id:  $\mathcal{B}_A \ (set\text{-unsat } I \ s) = \mathcal{B}_A \ s \ \mathcal{B}_u \ (set\text{-unsat } I \ s) = \mathcal{B}_u \ s \ \mathcal{B}_l$ 
    ( $set\text{-unsat } I \ s) = \mathcal{B}_l \ s$ 
    by (auto simp: boundsl-def boundsu-def bound-atoms-def)
  have t-id:  $\mathcal{T} \ (set\text{-unsat } I \ s) = \mathcal{T} \ s$  by simp
  have u-id:  $\mathcal{U} \ (set\text{-unsat } I \ s) = \text{True}$  by simp
  let  $?p = rhs \ ?eq - lp\text{-monom } 1 \ x_i$ 
  have p-eval-zero:  $?p \ \llbracket v \rrbracket = 0$  if  $v \models_t \mathcal{T} \ s$  for  $v :: 'a \ \text{valuation}$ 
  proof -
    have eqT:  $?eq \in set \ (\mathcal{T} \ s)$ 
      by (simp add:  $\exists(\mathcal{T}) \ eq\text{-for-lvar local.min-lvar-not-in-bounds-lvars}$ )
    have  $v \models_e \ ?eq$  using that eqT satisfies-tableau-def by blast
    also have  $?eq = (lhs \ ?eq, rhs \ ?eq)$  by (cases  $?eq$ , auto)
    also have  $lhs \ ?eq = x_i$  by (simp add:  $\exists(\mathcal{T}) \ eq\text{-for-lvar local.min-lvar-not-in-bounds-lvars}$ )
    finally have  $v \models_e \ (x_i, rhs \ ?eq)$  .
    then show  $?thesis$  by (auto simp: satisfies-eq-iff valuate-minus)
  qed
  have  $X_j$ -rvars:  $X_j \subseteq rvars \ (\mathcal{T} \ s)$  unfolding  $X_j$ -def
    using  $\exists \ min\text{-lvar-not-in-bounds-lvars rvars-of-lvar-rvars}$  by blast
  have  $xi$ -lvars:  $x_i \in lvars \ (\mathcal{T} \ s)$ 
    using  $\exists \ min\text{-lvar-not-in-bounds-lvars rvars-of-lvar-rvars}$  by blast
  have  $lvars \ (\mathcal{T} \ s) \cap rvars \ (\mathcal{T} \ s) = \{\}$ 
    using  $\exists \ normalized\text{-tableau-def}$  by auto
  with  $xi$ -lvars  $X_j$ -rvars have  $xi$ - $X_j$ :  $x_i \notin X_j$ 
    by blast
  have rhs-eval- $x_i$ :  $(rhs \ (eq\text{-for-lvar} \ (\mathcal{T} \ s) \ x_i)) \ \llbracket \langle \mathcal{V} \ s \rangle \rrbracket = \langle \mathcal{V} \ s \rangle \ x_i$ 
  proof -
    have *:  $(rhs \ eq) \ \llbracket v \rrbracket = v \ (lhs \ eq)$  if  $v \models_e \ eq$  for  $v :: 'a \ \text{valuation}$  and  $eq$ 
      using satisfies-eq-def that by metis
    moreover have  $\langle \mathcal{V} \ s \rangle \models_e \ eq\text{-for-lvar} \ (\mathcal{T} \ s) \ x_i$ 
      using  $\exists \ satisfies\text{-tableau-def eq-for-lvar curr-val-satisfies-no-lhs-def xi-lvars}$ 
      by blast
    ultimately show  $?thesis$ 
      using eq-for-lvar  $xi$ -lvars by simp
  qed
  let  $?B_l = Direction.LB \ dir$ 
  let  $?B_u = Direction.UB \ dir$ 
  let  $?lt = Direction.lt \ dir$ 
  let  $?le = Simplex.le \ ?lt$ 
  let  $?Geq = Direction.GE \ dir$ 
  let  $?Leq = Direction.LE \ dir$ 

  have 0: (if  $A \ x < 0$  then  $?B_l \ s \ x = Some \ (\langle \mathcal{V} \ s \rangle \ x)$  else  $?B_u \ s \ x = Some \ (\langle \mathcal{V} \ s \rangle \ x)$ )  $\wedge A \ x \neq 0$ 
    if  $x: x \in X_j$  for  $x$ 
  proof -

```

have $\text{Some } (\langle \mathcal{V} \ s \rangle x) = (? \mathcal{B}_l \ s \ x)$ **if** $A \ x < 0$
proof –
have $\text{cmp}: \neg \triangleright_{lb} \ ?lt \ (\langle \mathcal{V} \ s \rangle x) \ (? \mathcal{B}_l \ s \ x)$
using x **that** $\text{dir } \text{min-rvar-incdec-eq-None}[OF \ 3(9)]$ **unfolding** $X_j\text{-def}$
A-def **by** *auto*
then obtain c **where** $c: ? \mathcal{B}_l \ s \ x = \text{Some } c$
by (*cases* $? \mathcal{B}_l \ s \ x$, *auto simp: bound-compare-defs*)
also have $c = \langle \mathcal{V} \ s \rangle x$
proof –
have $x \in \text{rvars } (\mathcal{T} \ s)$ **using** x $X_j\text{-rvars}$ **by** *blast*
then have $x \in (- \text{lvars } (\mathcal{T} \ s))$
using 3 **unfolding** *normalized-tableau-def* **by** *auto*
moreover have $\forall x \in (- \text{lvars } (\mathcal{T} \ s)). \text{in-bounds } x \ \langle \mathcal{V} \ s \rangle (\mathcal{B}_l \ s, \mathcal{B}_u \ s)$
using 3 **unfolding** *curr-val-satisfies-no-lhs-def*
by (*simp add: satisfies-bounds-set.simps*)
ultimately have $\text{in-bounds } x \ \langle \mathcal{V} \ s \rangle (\mathcal{B}_l \ s, \mathcal{B}_u \ s)$
by *blast*
moreover have $?le \ (\langle \mathcal{V} \ s \rangle x) \ c$
using $\text{cmp } c \ \text{dir}$ **unfolding** *bound-compare-defs* **by** *auto*
ultimately show *?thesis*
using c **dir** **by** (*auto simp del: Simplex.bounds-lg*)
qed
then show *?thesis*
using c **by** *simp*
qed
moreover have $\text{Some } (\langle \mathcal{V} \ s \rangle x) = (? \mathcal{B}_u \ s \ x)$ **if** $0 < A \ x$
proof –
have $\text{cmp}: \neg \triangleleft_{ub} \ ?lt \ (\langle \mathcal{V} \ s \rangle x) \ (? \mathcal{B}_u \ s \ x)$
using x **that** $\text{min-rvar-incdec-eq-None}[OF \ 3(9)]$ **unfolding** $X_j\text{-def}$ *A-def*
by auto
then obtain c **where** $c: ? \mathcal{B}_u \ s \ x = \text{Some } c$
by (*cases* $? \mathcal{B}_u \ s \ x$, *auto simp: bound-compare-defs*)
also have $c = \langle \mathcal{V} \ s \rangle x$
proof –
have $x \in \text{rvars } (\mathcal{T} \ s)$ **using** x $X_j\text{-rvars}$ **by** *blast*
then have $x \in (- \text{lvars } (\mathcal{T} \ s))$
using 3 **unfolding** *normalized-tableau-def* **by** *auto*
moreover have $\forall x \in (- \text{lvars } (\mathcal{T} \ s)). \text{in-bounds } x \ \langle \mathcal{V} \ s \rangle (\mathcal{B}_l \ s, \mathcal{B}_u \ s)$
using 3 **unfolding** *curr-val-satisfies-no-lhs-def*
by (*simp add: satisfies-bounds-set.simps*)
ultimately have $\text{in-bounds } x \ \langle \mathcal{V} \ s \rangle (\mathcal{B}_l \ s, \mathcal{B}_u \ s)$
by *blast*
moreover have $?le \ c \ (\langle \mathcal{V} \ s \rangle x)$
using $\text{cmp } c \ \text{dir}$ **unfolding** *bound-compare-defs* **by** *auto*
ultimately show *?thesis*
using c **dir** **by** (*auto simp del: Simplex.bounds-lg*)
qed
then show *?thesis*
using c **by** *simp*

qed
moreover have $A \ x \neq 0$
using *that coeff-zero unfolding A-def X_j-def by auto*
ultimately show *?thesis*
using *that by auto*
qed

have $l\text{-Ba}: l \in \mathcal{B}_A \ s \text{ if } l \in \{?Geq \ x_i \ (the \ (?B_l \ s \ x_i))\}$ **for** l
proof –
from *that have* $l: l = ?Geq \ x_i \ (the \ (?B_l \ s \ x_i))$ **by** *simp*
from $\exists(8)$ **obtain** c **where** $bl': ?B_l \ s \ x_i = Some \ c$
by *(cases ?B_l s x_i, auto simp: bound-compare-defs)*
hence $bl: (x_i, c) \in set\text{-of-map} \ (?B_l \ s)$ **unfolding** *set-of-map-def* **by** *auto*
show $l \in \mathcal{B}_A \ s$ **unfolding** *l bound-atoms-def* **using** $bl \ bl'$ *dir* **by** *auto*
qed

let $?negA = filter \ (\lambda \ x. \ A \ x < 0) \ XL_j$
let $?posA = filter \ (\lambda \ x. \ \neg \ A \ x < 0) \ XL_j$

define neg **where** $neg = (if \ dir = Positive \ then \ (\lambda \ x :: rat. \ x) \ else \ uminus)$
define $negP$ **where** $negP = (if \ dir = Positive \ then \ (\lambda \ x :: linear\text{-poly}. \ x) \ else \ uminus)$

define $nega$ **where** $nega = (if \ dir = Positive \ then \ (\lambda \ x :: 'a. \ x) \ else \ uminus)$
from dir **have** $dirn: dir = Positive \wedge \ neg = (\lambda \ x. \ x) \wedge \ negP = (\lambda \ x. \ x) \wedge \ nega = (\lambda \ x. \ x)$
 $\vee \ dir = Negative \wedge \ neg = uminus \wedge \ negP = uminus \wedge \ nega = uminus$
unfolding *neg-def negP-def nega-def* **by** *auto*

define C **where** $C = map \ (\lambda \ x. \ ?Geq \ x \ (the \ (?B_l \ s \ x))) \ ?negA$
 $\quad @ \ map \ (\lambda \ x. \ ?Leq \ x \ (the \ (?B_u \ s \ x))) \ ?posA$
 $\quad @ \ [?Geq \ x_i \ (the \ (?B_l \ s \ x_i))]$

define f **where** $f = (\lambda \ x. \ if \ x = x_i \ then \ neg \ (-1) \ else \ neg \ (A \ x))$
define c **where** $c = (\sum \ x \leftarrow C. \ lec\text{-const} \ (lec\text{-of-nsc} \ (f \ (atom\text{-var} \ x) \ *R \ nsc\text{-of-atom} \ x)))$

let $?q = negP \ ?p$

show *?case unfolding bounds-id t-id u-id*
proof *(intro exI impI conjI allI)*
show $v \models_t \mathcal{T} \ s \implies ?q \ \Downarrow \ v \ \Downarrow = 0$ **for** $v :: 'a \ \text{valuation}$ **using** *dirn p-eval-zero[of v]*
by *(auto simp: valuate-minus)*

show $set \ C \subseteq \mathcal{B}_A \ s$
unfolding *C-def set-append set-map set-filter list.simps* **using** $0 \ l\text{-Ba} \ dir$
by *(intro Un-least subsetI) (force simp: bound-atoms-def set-of-map-def)+*

show *is-leq: $\forall a \in set \ C. \ is\text{-leq}\text{-ns} \ (f \ (atom\text{-var} \ a) \ *R \ nsc\text{-of-atom} \ a) \wedge \ f \ (atom\text{-var} \ a) \neq 0$*
using *dirn xi-Xj 0* **unfolding** *C-def f-def*

by (*elim disjE*, *auto*)

show $(\sum a \leftarrow C. \text{lec-of-nsc } (f \text{ (atom-var } a) *R \text{ nsc-of-atom } a)) = \text{Leq } ?q \ c$

unfolding *sum-list-lec le-constraint.simps map-map o-def*

proof (*intro conjI*)

define *scale-poly* :: 'a atom \Rightarrow linear-poly **where**

scale-poly = $(\lambda x. \text{lec-poly } (\text{lec-of-nsc } (f \text{ (atom-var } x) *R \text{ nsc-of-atom } x)))$

have $(\sum x \leftarrow C. \text{scale-poly } x) =$

$(\sum x \leftarrow ?negA. \text{scale-poly } (?Geq \ x \text{ (the } (?B_l \ s \ x))))$

$+ (\sum x \leftarrow ?posA. \text{scale-poly } (?Leq \ x \text{ (the } (?B_u \ s \ x))))$

$- \text{negP } (\text{lp-monom } 1 \ x_i)$

unfolding *C-def using dirn by (auto simp add: comp-def scale-poly-def*

f-def)

also have $(\sum x \leftarrow ?negA. \text{scale-poly } (?Geq \ x \text{ (the } (?B_l \ s \ x))))$

$= (\sum x \leftarrow ?negA. \text{negP } (A \ x *R \ \text{lp-monom } 1 \ x))$

unfolding *scale-poly-def f-def using dirn xi-Xj by (subst map-cong) auto*

also have $(\sum x \leftarrow ?posA. \text{scale-poly } (?Leq \ x \text{ (the } (?B_u \ s \ x))))$

$= (\sum x \leftarrow ?posA. \text{negP } (A \ x *R \ \text{lp-monom } 1 \ x))$

unfolding *scale-poly-def f-def using dirn xi-Xj by (subst map-cong) auto*

also have $(\sum x \leftarrow ?negA. \text{negP } (A \ x *R \ \text{lp-monom } 1 \ x)) +$

$(\sum x \leftarrow ?posA. \text{negP } (A \ x *R \ \text{lp-monom } 1 \ x))$

$= \text{negP } (\text{rhs } (\text{eq-for-lvar } (\mathcal{T} \ s) \ x_i))$

using *dirn XL_j-distinct coeff-zero*

by (*elim disjE; intro poly-eqI, auto intro!: poly-eqI simp add: coeff-sum-list*

A-def X_j-def

$\text{uminus-sum-list-map}[\text{unfolded } o\text{-def, symmetric}]$

finally show $(\sum x \leftarrow C. \text{lec-poly } (\text{lec-of-nsc } (f \text{ (atom-var } x) *R \text{ nsc-of-atom}$

x))) = ?q

unfolding *scale-poly-def using dirn by auto*

show $(\sum x \leftarrow C. \text{lec-rel } (\text{lec-of-nsc } (f \text{ (atom-var } x) *R \text{ nsc-of-atom } x))) =$

Leq-Rel

unfolding *sum-list-Leq-Rel*

proof

fix *c*

assume *c: c ∈ set C*

show *lec-rel (lec-of-nsc (f (atom-var c) *R nsc-of-atom c)) = Leq-Rel*

using *is-leq[rule-format, OF c] by (cases f (atom-var c) *R nsc-of-atom*

c, auto)

qed

qed (*simp add: c-def*)

show $c < 0$

proof –

define *scale-const-f* :: 'a atom \Rightarrow 'a **where**

scale-const-f $x = \text{lec-const } (\text{lec-of-nsc } (f \text{ (atom-var } x) *R \text{ nsc-of-atom } x))$

for *x*

obtain *d* **where** *bl': ?B_l s x_i = Some d*

using *3 by (cases ?B_l s x_i, auto simp: bound-compare-defs)*

have $c = (\sum x \leftarrow \text{map } (\lambda x. ?Geq \ x \text{ (the } (?B_l \ s \ x))) ?negA. \text{scale-const-f } x)$

x)

 $+ (\sum x \leftarrow \text{map } (\lambda x. ?\text{Leq } x \text{ (the } (?B_u \text{ } s \text{ } x))) ?\text{pos}A. \text{scale-const-f}$

 $- \text{nega } d$

unfolding *c-def C-def f-def scale-const-f-def* **using** *dirn rhs-eval-xi bl'* **by**

auto

also have $(\sum x \leftarrow \text{map } (\lambda x. ?\text{Geq } x \text{ (the } (?B_l \text{ } s \text{ } x))) ?\text{neg}A. \text{scale-const-f } x)$

 $=$

 $(\sum x \leftarrow ?\text{neg}A. \text{nega } (A \text{ } *R \text{ the } (?B_l \text{ } s \text{ } x)))$

using *xi-Xj dirn* **by** *(subst map-cong) (auto simp add: f-def scale-const-f-def)*

also have $\dots = (\sum x \leftarrow ?\text{neg}A. \text{nega } (A \text{ } *R \langle \mathcal{V} \text{ } s \rangle x))$

using *0* **by** *(subst map-cong) auto*

also have $(\sum x \leftarrow \text{map } (\lambda x. ?\text{Leq } x \text{ (the } (?B_u \text{ } s \text{ } x))) ?\text{pos}A. \text{scale-const-f } x)$

 $=$

 $(\sum x \leftarrow ?\text{pos}A. \text{nega } (A \text{ } *R \text{ the } (?B_u \text{ } s \text{ } x)))$

using *xi-Xj dirn* **by** *(subst map-cong) (auto simp add: f-def scale-const-f-def)*

also have $\dots = (\sum x \leftarrow ?\text{pos}A. \text{nega } (A \text{ } *R \langle \mathcal{V} \text{ } s \rangle x))$

using *0* **by** *(subst map-cong) auto*

also have $(\sum x \leftarrow ?\text{neg}A. \text{nega } (A \text{ } *R \langle \mathcal{V} \text{ } s \rangle x)) + (\sum x \leftarrow ?\text{pos}A. \text{nega } (A \text{ } *R \langle \mathcal{V} \text{ } s \rangle x))$

 $= (\sum x \leftarrow ?\text{neg}A \text{ @ } ?\text{pos}A. \text{nega } (A \text{ } *R \langle \mathcal{V} \text{ } s \rangle x))$

by *auto*

also have $\dots = (\sum x \in X_j. \text{nega } (A \text{ } *R \langle \mathcal{V} \text{ } s \rangle x))$

using *XLj-distinct* **by** *(subst sum-list-distinct-conv-sum-set) (auto intro!: sum.cong)*

also have $\dots = \text{nega } (\sum x \in X_j. (A \text{ } *R \langle \mathcal{V} \text{ } s \rangle x))$ **using** *dirn* **by** *(auto simp: sum-negf)*

also have $(\sum x \in X_j. (A \text{ } *R \langle \mathcal{V} \text{ } s \rangle x)) = ((\text{rhs } ?\text{eq}) \{\langle \mathcal{V} \text{ } s \rangle\})$

unfolding *A-def Xj-def* **by** *(subst linear-poly-sum) (auto simp add: sum-negf)*

also have $\dots = \langle \mathcal{V} \text{ } s \rangle x_i$

using *rhs-eval-xi* **by** *blast*

also have $\text{nega } (\langle \mathcal{V} \text{ } s \rangle x_i) - \text{nega } d < 0$

proof $-$

have $?lt \ (\langle \mathcal{V} \text{ } s \rangle x_i) \ d$

using *dirn 3(2-) bl'* **by** *(elim disjE, auto simp: bound-compare-defs)*

thus *?thesis* **using** *dirn unfolding minus-lt[symmetric]* **by** *auto*

qed

finally show *?thesis* .

qed

show *distinct C*

unfolding *C-def* **using** *XLj-distinct xi-Xj dirn* **by** *(auto simp add: inj-on-def distinct-map)*

qed

qed *(insert U, blast+)*

then obtain *f p c C* **where** *Qs: ?Q s f p c C* **using** *U* **unfolding** *check* **by**

blast

from *index[folded check-tableau-index-valid[OF U(2) inv(3,4,2,1)]]* *check*

have *index: index-valid as s* **by** *auto*

```

from check-tableau-equiv[OF U(2) inv(3,4,2,1), unfolded check]
have id:  $v \models_t \mathcal{T} s = v \models_t \mathcal{T} s'$  for  $v :: 'a \text{ valuation by auto}$ 
let  $?C = \text{map } (\lambda a. (f \text{ (atom-var } a), a)) C$ 
have set  $C \subseteq \mathcal{B}_A s$  using Qs by blast
also have  $\dots \subseteq \text{snd } 'as$  using index
  unfolding bound-atoms-def index-valid-def set-of-map-def boundsl-def boundsu-def
o-def by force
  finally have sub:  $\text{snd } 'set ?C \subseteq \text{snd } 'as$  by force
  show ?thesis unfolding farkas-coefficients-atoms-tableau-def
    by (intro exI[of - p] exI[of - c] exI[of - ?C] conjI,
      insert Qs[unfolded id] sub, (force simp: o-def)+)
qed

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:  $\neg \mathcal{U} s \models_{\text{noIhs}} s \Delta (\mathcal{T} s) \nabla s \diamond s$ 
  and index: index-valid as s
  and U:  $\mathcal{U}$  (assert-bound ia s)
shows  $\exists C. \text{farkas-coefficients-atoms-tableau } (\text{snd } '(\text{insert ia as})) (\mathcal{T} s) C$ 
proof –
  obtain i a where ia[simp]:  $ia = (i, a)$  by force
  let  $?A = \text{snd } ' \text{insert ia as}$ 
  have  $\exists x c d. \text{Leq } x c \in ?A \wedge \text{Geq } x d \in ?A \wedge c < d$ 
  proof (cases a)
    case (Geq x d)
      let  $?s = \text{updateBI } (\text{Direction.UBI-upd } (\text{Direction } (\lambda x y. y < x) \mathcal{B}_{iu} \mathcal{B}_{il} \mathcal{B}_u \mathcal{B}_l$ 
I_u I_l B_il-update Geq Leq ( $\leq$ )))
         $i x d s$ 
      have id:  $\mathcal{U} ?s = \mathcal{U} s$  by auto
      have norm:  $\Delta (\mathcal{T} ?s)$  using inv by auto
      have val:  $\nabla ?s$  using inv(4) unfolding tableau-valuated-def by simp
      have idd:  $x \notin \text{lvars } (\mathcal{T} ?s) \implies \mathcal{U} (\text{update } x d ?s) = \mathcal{U} ?s$ 
        by (rule update-unsat-id[OF norm val])
      from U[unfolded ia Geq] inv(1) id idd
      have  $\triangleleft_{ib} (\lambda x y. y < x) d (\mathcal{B}_u s x)$  by (auto split: if-splits simp: Let-def)
      then obtain c where Bu:  $\mathcal{B}_u s x = \text{Some } c$  and lt:  $c < d$ 
        by (cases B_u s x, auto simp: bound-compare-defs)
      from Bu obtain j where Mapping.lookup ( $\mathcal{B}_{iu} s$ )  $x = \text{Some } (j, c)$ 
        unfolding boundsu-def by auto
      with index[unfolded index-valid-def] have  $(j, \text{Leq } x c) \in as$  by auto
      hence xc:  $\text{Leq } x c \in ?A$  by force
      have xd:  $\text{Geq } x d \in ?A$  unfolding ia Geq by force
      from xc xd lt show ?thesis by auto
next

```

```

case (Leq x c)
  let ?s = updateBI (Direction.UBI-upd (Direction (<) Bil Biu Bl Bu Il Iu
Biu-update Leq Geq (≥))) i x c s
  have id: U ?s = U s by auto
  have norm: Δ (T ?s) using inv by auto
  have val: ∇ ?s using inv(4) unfolding tableau-valuated-def by simp
  have idd: x ∉ lvars (T ?s) ⇒ U (update x c ?s) = U ?s
    by (rule update-unsat-id[OF norm val])
  from U[unfolded ia Leq] inv(1) id idd
  have <lb (<) c (Bl s x) by (auto split: if-splits simp: Let-def)
  then obtain d where Bl: Bl s x = Some d and lt: c < d
    by (cases Bl s x, auto simp: bound-compare-defs)
  from Bl obtain j where Mapping.lookup (Bil s) x = Some (j,d)
    unfolding boundsl-def by auto
  with index[unfolded index-valid-def] have (j, Geq x d) ∈ as by auto
  hence xd: Geq x d ∈ ?A by force
  have xc: Leq x c ∈ ?A unfolding ia Leq by force
  from xc xd lt show ?thesis by auto
qed
then obtain x c d where c: Leq x c ∈ ?A and d: Geq x d ∈ ?A and cd: c < d
by blast
show ?thesis unfolding farkas-coefficients-atoms-tableau-def
proof (intro exI conjI allI)
  let ?C = [(-1, Geq x d), (1, Leq x c)]
  show ∀(r,a)∈set ?C. a ∈ ?A ∧ is-leq-ns (r *R nsc-of-atom a) ∧ r ≠ 0 using
c d by auto
  show c - d < 0 using cd using minus-lt by auto
  qed (auto simp: valuate-zero)
qed
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
using assms **unfolding** farkas-coefficients-atoms-tableau-def **by** force

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*: Δt
shows $\exists C. \text{farkas-coefficients-atoms-tableau (snd ' set as) t C}$
proof –
let $?P = \lambda \text{ as } s. \mathcal{U} s \longrightarrow (\exists C. \text{farkas-coefficients-atoms-tableau (snd ' as) } (\mathcal{T} s) C)$
let $?s = \text{assert-bound-loop as (init t)}$
have $\neg \mathcal{U} (\text{init } t)$ **by** (*rule init-unsat-flag*)
have $\mathcal{T} (\text{assert-bound-loop as (init t)}) = t \wedge$
 $(\mathcal{U} (\text{assert-bound-loop as (init t)}) \longrightarrow (\exists C. \text{farkas-coefficients-atoms-tableau (snd ' set as) } (\mathcal{T} (\text{init } t)) C))$
proof (*rule AssertAllState''Induct[OF norm], unfold ass-bnd, goal-cases*)
case 1
have $\neg \mathcal{U} (\text{init } t)$ **by** (*rule init-unsat-flag*)
moreover **have** $\mathcal{T} (\text{init } t) = t$ **by** (*rule init-tableau-id*)
ultimately show *?case* **by** *auto*
next
case (2 *as bs s*)
hence $\text{snd ' as} \subseteq \text{snd ' bs}$ **by** *auto*
from *farkas-coefficients-atoms-tableau-mono*[*OF this*] 2(2) **show** *?case* **by** *auto*
next
case (3 *s a ats*)
let $?s = \text{assert-bound } a s$
have $\text{tab: } \mathcal{T} ?s = \mathcal{T} s$ **unfolding** *ass-bnd* **by** (*rule assert-bound-nolhs-tableau-id, insert 3, auto*)
have $t: t = \mathcal{T} s$ **using** 3 **by** *simp*
show *?case* **unfolding** $t \text{ tab}$
proof (*intro conjI impI refl*)
assume $\mathcal{U} ?s$
from *farkas-assert-bound*[*OF 3(1,3-6,8) this*]
show $\exists C. \text{farkas-coefficients-atoms-tableau (snd ' insert a (set ats)) } (\mathcal{T} (\text{init } (\mathcal{T} s))) C$
unfolding $t[\text{symmetric}] \text{init-tableau-id}$.
qed
qed
thus *?thesis* **unfolding** *init-tableau-id* **using** *assms* **by** *blast*
qed

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*: Δt

```

shows  $\exists C. \text{farkas-coefficients-atoms-tableau } (\text{snd } \text{' set as}) t C$ 
proof –
  let  $?s = \text{assert-bound-loop as } (\text{init } t)$ 
  show  $?thesis$ 
  proof ( $\text{cases } \mathcal{U} (\text{assert-bound-loop as } (\text{init } t))$ )
    case True
      from  $\text{farkas-assert-bound-loop}[OF \text{ this norm}]$ 
      show  $?thesis$  by auto
    next
      case False
      from  $\text{AssertAllState''-tableau-id}[OF \text{ norm}]$ 
      have  $T: \mathcal{T} \ ?s = t$  unfolding  $\text{init-tableau-id}$  .
      from  $U$  have  $U: \mathcal{U} (\text{check } ?s)$  unfolding  $\text{chk}[symmetric]$  by simp
      show  $?thesis$ 
      proof ( $\text{rule } \text{farkas-check}[OF \text{ refl } U \text{ False, unfolded } T, OF - \text{norm}]$ )
        from  $\text{AssertAllState''-precond}[OF \text{ norm, unfolded } \text{Let-def}] \text{ False}$ 
        show  $\models_{\text{noths}} ?s \diamond ?s \nabla ?s$  by blast+
        from  $\text{AssertAllState''-index-valid}[OF \text{ norm}]$ 
        show  $\text{index-valid } (\text{set as}) ?s$  .
      qed
    qed
  qed

```

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 $\text{farkas-coefficients-atoms-tableau as } t C$

shows $\nexists v. v \models_t t \wedge v \models_{as} as$

proof

assume $\exists v. v \models_t t \wedge v \models_{as} as$

then obtain v **where** $*$: $v \models_t t \wedge v \models_{as} as$ **by** *auto*

then obtain $p \ c$ **where** $\text{isleg}: (\forall (r,a) \in \text{set } C. a \in as \wedge \text{is-leg-ns } (r *R \text{nsc-of-atom } a) \wedge r \neq 0)$

and $\text{leq}: (\sum (r,a) \leftarrow C. \text{lec-of-nsc } (r *R \text{nsc-of-atom } a)) = \text{Leq } p \ c$

and $\text{cltz}: c < 0$

and $p0: p \{v\} = 0$

using $\text{assms } \text{farkas-coefficients-atoms-tableau-def}$ **by** *blast*

have $\text{fa}: \forall (r,a) \in \text{set } C. v \models_a a$ **using** $*$ $\text{isleg } \text{leq}$

```

    satisfies-atom-set-def by force
  {
    fix  $r a$ 
    assume  $a: (r, a) \in \text{set } C$ 
    from  $a \text{ fa}$  have  $va: v \models_a a$  unfolding satisfies-atom-set-def by auto
    hence  $v: v \models_{ns} (r * R \text{ nsc-of-atom } a)$  by (auto simp: nsc-of-atom sat-scale-rat-ns)
    from  $a \text{ isleq}$  have  $\text{is-leq-ns } (r * R \text{ nsc-of-atom } a)$  by auto
    from lec-of-nsc[OF this]  $v$  have  $v \models_{le} \text{lec-of-nsc } (r * R \text{ nsc-of-atom } a)$  by blast
  } note  $v = \text{this}$ 
  have  $v \models_{le} \text{Leqc } p \ c$  unfolding leq[symmetric]
    by (rule satisfies-sumlist-le-constraints, insert v, auto)
  then have  $0 \leq c$  using  $p0$  by auto
  then show False using cltz by auto
qed

```

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:  $\Delta t$ 
  and fin: finite as
shows  $(\exists C. \text{farkas-coefficients-atoms-tableau as } t \ C) \iff (\nexists v. v \models_t t \wedge v \models_{as} as)$ 
proof
  from finite-list[OF fin] obtain  $bs$  where  $as: as = \text{set } bs$  by auto
  assume unsat:  $\nexists v. v \models_t t \wedge v \models_{as} as$ 
  let  $?as = \text{map } (\lambda x. ((), x)) \ bs$ 
  interpret AssertAllState''' init-state assert-bound-code check-code update-code
    eq-id-x-for-lvar min-lvar-not-in-bounds min-rvar-incdec-eq pivot-and-update-code
  by (unfold-locales, auto simp: assert-bound-code-def check-code-def)
  let  $?call = \text{assert-all } t \ ?as$ 
  have  $id: \text{snd } ' \text{set } ?as = as$  unfolding  $as$  by force
  from assert-all-sat[OF norm, of ?as, unfolded id] unsat
  obtain  $I$  where  $?call = \text{Inl } I$  by (cases ?call, auto)
  from this[unfolded assert-all-def Let-def]
  have  $\mathcal{U} (\text{assert-all-state-code } t \ ?as)$ 
    by (auto split: if-splits simp: assert-all-state-code-def)
  from farkas-assert-all-state[OF this[unfolded assert-all-state-code-def] norm, unfolded id]
  show  $\exists C. \text{farkas-coefficients-atoms-tableau as } t \ C$  .
qed (insert farkas-coefficients-atoms-tableau-unsat, auto)

```

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 previous 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 $\nexists v. v \models_{ns} ns$

proof

assume $\exists v. v \models_{ns} ns$

then obtain v **where** $*$: $v \models_{ns} ns$ **by** *auto*

obtain c **where**

isleg: $(\forall (a, n) \in set\ C. n \in ns \wedge is\ leq\ ns\ (a *R\ n) \wedge a \neq 0)$ **and**

leq: $(\sum (a, n) \leftarrow C. lec\ of\ nsc\ (a *R\ n)) = Leqc\ 0\ c$ **and**

cltz: $c < 0$ **using** *assms farkas-coefficients-ns-def* **by** *blast*

{

fix $a\ n$

assume n : $(a, n) \in set\ C$

from $n * isleg$ **have** $v \models_{ns} n$ **by** *auto*

hence $v: v \models_{ns} (a *R\ n)$ **by** (*rule sat-scale-rat-ns*)

from $n isleg$ **have** *is-leq-ns* $(a *R\ n)$ **by** *auto*

from *lec-of-nsc[OF this]* v

have $v \models_{le} lec\ of\ nsc\ (a *R\ n)$ **by** *blast*

} **note** $v = this$

have $v \models_{le} Leqc\ 0\ c$ **unfolding** *leq[symmetric]*

by (*rule satisfies-sumlist-le-constraints, insert v, auto*)

then show *False* **using** *cltz*

by (*metis leD satisfiable-le-constraint.simps evaluate-zero rel-of.simps(1)*)

qed

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* $:: (var \Rightarrow linear\ poly) \Rightarrow linear\ poly \Rightarrow linear\ poly$ **where**

$$\text{subst-poly } \sigma p = (\sum x \in \text{vars } p. \text{coeff } p x *R \sigma x)$$

lemma *subst-poly-0[simp]*: *subst-poly* $\sigma 0 = 0$ **unfolding** *subst-poly-def* **by** *simp*

lemma *valuate-subst-poly*: $(\text{subst-poly } \sigma p) \llbracket v \rrbracket = (p \llbracket (\lambda x. ((\sigma x) \llbracket v \rrbracket)) \rrbracket)$
by (*subst* (2) *linear-poly-sum*, *unfold subst-poly-def* *valuate-sum* *valuate-scaleRat*, *simp*)

lemma *subst-poly-add*: *subst-poly* $\sigma (p + q) = \text{subst-poly } \sigma p + \text{subst-poly } \sigma q$
by (*rule linear-poly-eqI*, *unfold* *valuate-add* *valuate-subst-poly*, *simp*)

fun *subst-poly-lec* :: (*var* \Rightarrow *linear-poly*) \Rightarrow 'a *le-constraint* \Rightarrow 'a *le-constraint*
where

$$\text{subst-poly-lec } \sigma (\text{Le-Constraint rel } p c) = \text{Le-Constraint rel } (\text{subst-poly } \sigma p) c$$

lemma *subst-poly-lec-0[simp]*: *subst-poly-lec* $\sigma 0 = 0$ **unfolding** *zero-le-constraint-def*
by *simp*

lemma *subst-poly-lec-add*: *subst-poly-lec* $\sigma (c1 + c2) = \text{subst-poly-lec } \sigma c1 + \text{subst-poly-lec } \sigma c2$
by (*cases* *c1*; *cases* *c2*, *auto* *simp*: *subst-poly-add*)

lemma *subst-poly-lec-sum-list*: *subst-poly-lec* $\sigma (\text{sum-list } ps) = \text{sum-list } (\text{map } (\text{subst-poly-lec } \sigma) ps)$
by (*induct* *ps*, *auto* *simp*: *subst-poly-lec-add*)

lemma *subst-poly-lp-monom[simp]*: *subst-poly* $\sigma (\text{lp-monom } r x) = r *R \sigma x$
unfolding *subst-poly-def* **by** (*simp* *add*: *vars-lp-monom*)

lemma *subst-poly-scaleRat*: *subst-poly* $\sigma (r *R p) = r *R (\text{subst-poly } \sigma p)$
by (*rule linear-poly-eqI*, *unfold* *valuate-scaleRat* *valuate-subst-poly*, *simp*)

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

lemma *Tableau-is-monom-preprocess'*:
assumes $(x, p) \in \text{set } (\text{Tableau } (\text{preprocess}' cs \text{ start}))$
shows $\neg \text{is-monom } p$
using *assms*
by(*induction* *cs* *start* *rule*: *preprocess'.induct*)
(auto *simp* *add*: *Let-def* *split*: *if-splits* *option.splits*)

lemma *preprocess'-atoms-to-constraints'*: **assumes** *preprocess'* *cs* *start* = *S*
shows $\text{set } (\text{Atoms } S) \subseteq \{(i, \text{qdelta-constraint-to-atom } c v) \mid i c v. (i, c) \in \text{set } cs\}$
 \wedge
 $(\neg \text{is-monom } (\text{poly } c) \longrightarrow \text{Poly-Mapping } S (\text{poly } c) = \text{Some } v)$
unfolding *assms*(1)[*symmetric*]
by (*induct* *cs* *start* *rule*: *preprocess'.induct*, *auto* *simp*: *Let-def* *split*: *option.splits*, *force+*)

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
using *assms is-monom-monom-coeff-not-zero*
by(*cases a*; *cases ns*)
(*auto split: atom.split ns-constraint.split simp add: monom-poly-assemble field-simps*)

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 \Rightarrow LEQ-ns q c | Geq v c \Rightarrow GEQ-ns q c)*
and *a* \in *snd ' set (Atoms S)*
shows (*atom-var a* \notin *fst ' set (Tableau S)* \longrightarrow (\exists r. r \neq 0 \wedge r *R *nsc-of-atom a* \in *snd ' set cs*))
 \wedge ((*atom-var a*, q) \in *set (Tableau S)* \longrightarrow *ns* \in *snd ' set cs*)
proof –
let ?S = *preprocess' cs start*
from *assms(4)* **obtain** *i* **where** *ia*: (*i*,*a*) \in *set (Atoms S)* **by** *auto*
with *preprocess'-atoms-to-constraints'[OF assms(1)]* **obtain** *c v*
where *a*: *a = qdelta-constraint-to-atom c v* **and** *c*: (*i*,*c*) \in *set cs*
and *nmonom*: \neg *is-monom* (poly c) \implies *Poly-Mapping S* (poly c) = *Some v*
by *blast*
hence *c'*: *c* \in *snd ' set cs* **by** *force*
let ?p = *poly c*
show ?thesis
proof (*cases is-monom ?p*)
case *True*
hence *av*: *atom-var a = monom-var ?p* **unfolding** *a* **by** (*cases c*, *auto*)
from *Tableau-is-monom-preprocess'[of - ?p cs start]* *True*
have (*x*, ?p) \notin *set (Tableau ?S)* **for** *x* **by** *blast*
{
assume (*atom-var a*, q) \in *set (Tableau S)*
hence (*monom-var ?p*, q) \in *set (Tableau S)* **unfolding** *av* **by** *simp*
hence *monom-var ?p* \in *lvars (Tableau S)* **unfolding** *lvars-def* **by** *force*
from *lvars-tableau-ge-start[rule-format, OF this[folded S]]*
have *monom-var ?p* \geq *start* **unfolding** *S* .
moreover **have** *monom-var ?p* \in *vars-constraints (map snd cs)* **using** *True*
c
by (*auto intro!*: *bexI[of - (i,c)] simp: monom-var-in-vars*)
ultimately **have** *False* **using** *start-fresh-variable-fresh[of cs, folded start]* **by**
force
}
moreover
from *monom-of-atom-coeff[OF True a]* *True*
have \exists r. r \neq 0 \wedge r *R *nsc-of-atom a* = *c*
by (*intro exI[of - monom-coeff ?p]*, *auto*, *cases a*, *auto*)
ultimately **show** ?thesis **using** *c'* **by** *auto*

```

next
  case False
  hence av: atom-var a = v unfolding a by (cases c, auto)
  from nmonom[OF False] have Poly-Mapping S ?p = Some v .
  from preprocess'-Tableau-Poly-Mapping-Some[OF this[folded S]]
  have tab: (atom-var a, ?p) ∈ set (Tableau (preprocess' cs start)) unfolding av
by simp
  hence atom-var a ∈ fst ' set (Tableau S) unfolding S by force
  moreover
  {
    assume (atom-var a, q) ∈ set (Tableau S)
    from tab this have qp: q = ?p unfolding S using lvvars-distinct[of cs start,
unfolded S lhs-def]
    by (simp add: case-prod-beta' eq-key-imp-eq-value)
    have ns = c unfolding ns qp using av a False by (cases c, auto)
    hence ns ∈ snd ' set cs using c' by blast
  }
  ultimately show ?thesis by blast
qed
qed

```

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 –
  note ft[unfolded farkas-coefficients-atoms-tableau-def]
  obtain p c where 0: ∀ (r,a) ∈ set C. a ∈ snd ' set (Atoms S) ∧ 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 :: QDelta valuation. v ⊨t Tableau S ⇒ p {v} = 0
  using ft unfolding farkas-coefficients-atoms-tableau-def by blast
  note 0 = 0(1)[rule-format, of (a, b) for a b, unfolded split] 0(2-)
  let ?T = Tableau S
  define σ :: var ⇒ linear-poly where σ = (λ x. case map-of ?T x of Some p ⇒
p | None ⇒ lp-monom 1 x)
  let ?P = (λ r a s ns. ns ∈ (snd ' set cs) ∧ is-leq-ns (s *R ns) ∧ s ≠ 0 ∧
  subst-poly-lec σ (lec-of-nsc (r *R nsc-of-atom a)) = lec-of-nsc (s *R ns))
  have ∃ s ns. ?P r a s ns if ra: (r,a) ∈ set C for r a
  proof –
  have a: a ∈ snd ' set (Atoms S)
  using ra 0 by force
  from 0 ra have is-leq: is-leq-ns (r *R nsc-of-atom a) and r0: r ≠ 0 by auto
  let ?x = atom-var a
  show ?thesis

```

proof (*cases map-of ?T ?x*)
case (*Some q*)
hence $\sigma: \sigma \ ?x = q$ **unfolding** σ -def **by** *auto*
from *Some* **have** $xqT: (?x, q) \in \text{set } ?T$ **by** (*rule map-of-SomeD*)
define *ns* **where** $ns = (\text{case } a \text{ of } \text{Leq } v \ c \Rightarrow \text{LEQ-ns } q \ c$
 $\quad \mid \ \text{Geq } v \ c \Rightarrow \text{GEQ-ns } q \ c)$
from *preprocess'-atoms-to-constraints*[*OF pp refl ns-def a*] xqT
have *ns-mem*: $ns \in \text{snd } \text{' set cs}$ **by** *blast*
have *id*: $\text{subst-poly-lec } \sigma \ (\text{lec-of-nsc } (r *R \ \text{nsc-of-atom } a))$
 $= \text{lec-of-nsc } (r *R \ ns)$ **using** *is-leq* σ
by (*cases a, auto simp: ns-def subst-poly-scaleRat*)
from *id is-leq* σ **have** *is-leq*: $\text{is-leq-ns } (r *R \ ns)$ **by** (*cases a, auto simp:*
ns-def)
show *?thesis* **by** (*intro exI[of - r] exI[of - ns] conjI ns-mem id is-leq conjI r0*)
next
case *None*
hence $?x \notin \text{fst } \text{' set } ?T$ **by** (*meson map-of-eq-None-iff*)
from *preprocess'-atoms-to-constraints*[*OF pp refl refl a*] *this*
obtain *rr* **where** $rr: rr *R \ \text{nsc-of-atom } a \in (\text{snd } \text{' set cs})$ **and** $rr0: rr \neq 0$
by *blast*
from *None* **have** $\sigma: \sigma \ ?x = \text{lp-monom } 1 \ ?x$ **unfolding** σ -def **by** *simp*
define *ns* **where** $ns = rr *R \ \text{nsc-of-atom } a$
define *s* **where** $s = r / rr$
from *rr0 r0* **have** $s0: s \neq 0$ **unfolding** *s-def* **by** *auto*
from *is-leq* σ
have $\text{subst-poly-lec } \sigma \ (\text{lec-of-nsc } (r *R \ \text{nsc-of-atom } a))$
 $= \text{lec-of-nsc } (r *R \ \text{nsc-of-atom } a)$
by (*cases a, auto simp: subst-poly-scaleRat*)
moreover **have** $r *R \ \text{nsc-of-atom } a = s *R \ ns$ **unfolding** *ns-def s-def*
 $\text{scaleRat-scaleRat-ns-constraint}$ [*OF rr0*] **using** *rr0* **by** *simp*
ultimately **have** $\text{subst-poly-lec } \sigma \ (\text{lec-of-nsc } (r *R \ \text{nsc-of-atom } a))$
 $= \text{lec-of-nsc } (s *R \ ns)$ *is-leq-ns* $(s *R \ ns)$ **using** *is-leq* **by** *auto*
then show *?thesis*
unfolding *ns-def* **using** *rr s0* **by** *blast*
qed
qed
hence $\forall \ ra. \exists \ s \ ns. (\text{fst } ra, \text{snd } ra) \in \text{set } C \longrightarrow ?P \ (\text{fst } ra) \ (\text{snd } ra) \ s \ ns$ **by**
blast
from *choice*[*OF this*] **obtain** *s* **where** $s: \forall \ ra. \exists \ ns. (\text{fst } ra, \text{snd } ra) \in \text{set } C \longrightarrow$
 $?P \ (\text{fst } ra) \ (\text{snd } ra) \ (s \ ra) \ ns$ **by** *blast*
from *choice*[*OF this*] **obtain** *ns* **where** $ns: \bigwedge \ r \ a. (r, a) \in \text{set } C \implies ?P \ r \ a \ (s$
 $(r, a)) \ (ns \ (r, a))$ **by** *force*
define *NC* **where** $NC = \text{map } (\lambda(r, a). (s \ (r, a), ns \ (r, a))) \ C$
have $(\sum (s, ns) \leftarrow \text{map } (\lambda(r, a). (s \ (r, a), ns \ (r, a))) \ C'. \ \text{lec-of-nsc } (s *R \ ns)) =$
 $(\sum (r, a) \leftarrow C'. \ \text{subst-poly-lec } \sigma \ (\text{lec-of-nsc } (r *R \ \text{nsc-of-atom } a)))$
if $\text{set } C' \subseteq \text{set } C$ **for** C'
using *that proof* (*induction C'*)
case *Nil*
then show *?case* **by** *simp*

```

next
  case (Cons a C')
  have ( $\sum x \leftarrow a \# C'. \text{lec-of-nsc } (s x *R ns x)$ ) =
     $\text{lec-of-nsc } (s a *R ns a) + (\sum x \leftarrow C'. \text{lec-of-nsc } (s x *R ns x))$ 
  by simp
  also have ( $\sum x \leftarrow C'. \text{lec-of-nsc } (s x *R ns x)$ ) = ( $\sum (r, a) \leftarrow C'. \text{subst-poly-lec } \sigma (\text{lec-of-nsc } (r *R \text{nsc-of-atom } a))$ )
  using Cons by (auto simp add: case-prod-beta' comp-def)
  also have  $\text{lec-of-nsc } (s a *R ns a) = \text{subst-poly-lec } \sigma (\text{lec-of-nsc } (\text{fst } a *R \text{nsc-of-atom } (\text{snd } a)))$ 
  proof -
    have  $a \in \text{set } C$ 
    using Cons by simp
    then show ?thesis
    using ns by auto
  qed
  finally show ?case
  by (auto simp add: case-prod-beta' comp-def)
qed
also have ( $\sum (r, a) \leftarrow C. \text{subst-poly-lec } \sigma (\text{lec-of-nsc } (r *R \text{nsc-of-atom } a))$ )
  =  $\text{subst-poly-lec } \sigma (\sum (r, a) \leftarrow C. (\text{lec-of-nsc } (r *R \text{nsc-of-atom } a))$ )
  by (auto simp add: subst-poly-lec-sum-list case-prod-beta' comp-def)
also have ( $\sum (r, a) \leftarrow C. (\text{lec-of-nsc } (r *R \text{nsc-of-atom } a))$ ) =  $\text{Leqc } p c$ 
  using 0 by blast
also have  $\text{subst-poly-lec } \sigma (\text{Leqc } p c) = \text{Leqc } (\text{subst-poly } \sigma p) c$  by simp
also have  $\text{subst-poly } \sigma p = 0$ 
  proof (rule all-valuate-zero)
    fix  $v :: QDelta \text{ valuation}$ 
    have  $(\text{subst-poly } \sigma p) \Downarrow v \Downarrow = (p \Downarrow \lambda x. ((\sigma x) \Downarrow v \Downarrow)) \Downarrow$  by (rule valuate-subst-poly)
    also have  $\dots = 0$ 
  proof (rule 0(4))
    have  $(\sigma a) \Downarrow v \Downarrow = (q \Downarrow \lambda x. ((\sigma x) \Downarrow v \Downarrow)) \Downarrow$  if  $(a, q) \in \text{set } (\text{Tableau } S)$  for  $a q$ 
    proof -
      have distinct (map fst ?T)
      using normalized-tableau-preprocess' assms unfolding normalized-tableau-def lhs-def
      by (auto simp add: case-prod-beta')
      then have  $0: \sigma a = q$ 
      unfolding  $\sigma$ -def using that by auto
      have  $q \Downarrow v \Downarrow = (q \Downarrow \lambda x. ((\sigma x) \Downarrow v \Downarrow)) \Downarrow$ 
      proof -
        have vars  $q \subseteq \text{rvars } ?T$ 
        unfolding rvars-def using that by force
        moreover have  $(\sigma x) \Downarrow v \Downarrow = v x$  if  $x \in \text{rvars } ?T$  for  $x$ 
        proof -
          have  $x \notin \text{lvars } (\text{Tableau } S)$ 
          using that normalized-tableau-preprocess' assms

```

unfolding *normalized-tableau-def* **by** *blast*
then have $x \notin \text{fst } \text{'set (Tableau } S)$
unfolding *lvars-def* **by** *force*
then have $\text{map-of } ?T \ x = \text{None}$
using *map-of-eq-None-iff* **by** *metis*
then have $\sigma \ x = \text{lp-monom } 1 \ x$
unfolding σ -*def* **by** *auto*
also have $(\text{lp-monom } 1 \ x) \ \{\!| \ v \ |\!\} = v \ x$
by *auto*
finally show *?thesis* .
qed
ultimately show *?thesis*
by (*auto intro!*: *valuate-depend*)
qed
then show *?thesis*
using 0 **by** *blast*
qed
then show $(\lambda x. ((\sigma \ x) \ \{\!| \ v \ |\!\})) \models_t \ ?T$
using 0 **by** (*auto simp add: satisfies-tableau-def satisfies-eq-def*)
qed
finally show $(\text{subst-poly } \sigma \ p) \ \{\!| \ v \ |\!\} = 0$.
qed
finally have $(\sum (s, n) \leftarrow \text{NC}. \text{lec-of-nsc } (s *R \ n)) = \text{Le-Constraint Leq-Rel } 0 \ c$
unfolding *NC-def* **by** *blast*
moreover have $\text{ns } (r, a) \in \text{snd } \text{'set } cs \ \text{is-leq-ns } (s \ (r, a) *R \ \text{ns } (r, a)) \ s \ (r, a)$
 $\neq 0$ **if** $(r, a) \in \text{set } C$ **for** $r \ a$
using *ns that* **by** *force+*
ultimately have *farkas-coefficients-ns* ($\text{snd } \text{'set } cs$) *NC*
unfolding *farkas-coefficients-ns-def* *NC-def* **using** 0 **by** *force*
then show *?thesis*
by *blast*
qed

lemma *preprocess'-unsat-indexD*: $i \in \text{set } (\text{UnsatIndices } (\text{preprocess}' \ \text{ns } j)) \implies$
 $\exists \ c. \ \text{poly } c = 0 \wedge \neg \ \text{zero-satisfies } c \wedge (i, c) \in \text{set } \text{ns}$
by (*induct ns j rule: preprocess'.induct, auto simp: Let-def split: if-splits option.splits*)

lemma *preprocess'-unsat-index-farkas-coefficients-ns*:
assumes $i \in \text{set } (\text{UnsatIndices } (\text{preprocess}' \ \text{ns } j))$
shows $\exists \ C. \ \text{farkas-coefficients-ns } (\text{snd } \text{'set } \text{ns}) \ C$
proof –
from *preprocess'-unsat-indexD*[*OF assms*]
obtain c **where** *contr*: $\text{poly } c = 0 \wedge \neg \ \text{zero-satisfies } c$ **and** *mem*: $(i, c) \in \text{set } \text{ns}$ **by**
auto
from *mem* **have** *mem*: $c \in \text{snd } \text{'set } \text{ns}$ **by** *force*
let $?c = \text{ns-constraint-const } c$
define r **where** $r = (\text{case } c \ \text{of } \text{LEQ-ns} \ - \ - \Rightarrow 1 \ | \ - \Rightarrow (-1 \ :: \ \text{rat}))$
define d **where** $d = (\text{case } c \ \text{of } \text{LEQ-ns} \ - \ - \Rightarrow ?c \ | \ - \Rightarrow - \ ?c)$

have [*simp*]: $(-x < 0) = (0 < x)$ **for** $x :: QDelta$ **using** *uminus-less-lrv*[*of - 0*]
by *simp*
show *?thesis unfolding farkas-coefficients-ns-def*
by (*intro exI*[*of - [(r,c)]*]) *exI*[*of - d*], *insert mem contr*, *cases c*,
auto simp: r-def d-def)
qed

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 $(\exists C. \text{farkas-coefficients-ns } ns \ C) \longleftrightarrow (\nexists v. v \models_{nss} ns)$

proof

assume $\exists C. \text{farkas-coefficients-ns } ns \ C$
from *farkas-coefficients-ns-unsat* **this show** $\nexists v. v \models_{nss} ns$ **by** *blast*
next
assume *unsat: $\nexists v. v \models_{nss} ns$*
from *finite-list[OF assms]* **obtain** *nsl* **where** $ns: ns = \text{set } nsl$ **by** *auto*
let $?cs = \text{map } (\text{Pair } ()) \ nsl$
obtain $I \ t \ ias$ **where** *part1: preprocess-part-1 ?cs = (t,ias,I)* **by** (*cases preprocess-part-1 ?cs, auto*)
let $?as = \text{snd ' set } ias$
let $?s = \text{start-fresh-variable } ?cs$
have *fin: finite ?as* **by** *auto*
have *id: ias = Atoms (preprocess' ?cs ?s) t = Tableau (preprocess' ?cs ?s)*
I = UnsatIndices (preprocess' ?cs ?s)
using *part1 unfolding preprocess-part-1-def Let-def* **by** *auto*
have *norm: Δt using normalized-tableau-preprocess'[of ?cs] unfolding id .*
 $\{$
fix v
assume $v \models_{as} ?as \ v \models_t t$
from *preprocess'-sat[OF this[unfolded id], folded id] unsat[unfolded ns]*
have $\text{set } I \neq \{\}$ **by** *auto*
then obtain i **where** $i \in \text{set } I$ **using** *all-not-in-conv* **by** *blast*
from *preprocess'-unsat-index-farkas-coefficients-ns[OF this[unfolded id]]*
have $\exists C. \text{farkas-coefficients-ns } (\text{snd ' set } ?cs) \ C$ **by** *simp*
 $\}$
with *farkas-coefficients-atoms-tableau[OF norm fin]*
obtain C **where** *farkas-coefficients-atoms-tableau ?as t C*
 $\vee (\exists C. \text{farkas-coefficients-ns } (\text{snd ' set } ?cs) \ C)$ **by** *blast*
from *farkas-coefficients-preprocess'[of ?cs, OF refl] this*
have $\exists C. \text{farkas-coefficients-ns } (\text{snd ' set } ?cs) \ C$
using *part1 unfolding preprocess-part-1-def Let-def* **by** *auto*
also have $\text{snd ' set } ?cs = ns$ **unfolding** *ns* **by** *force*
finally show $\exists C. \text{farkas-coefficients-ns } ns \ C$.
qed

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* \Rightarrow *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 ($v \models_{le} (\text{lec-of-constraint } c)$) \longleftrightarrow ($v \models_c c$)

using *assms* **by** (*cases c, auto*)

instantiation *constraint* :: *scaleRat*

begin

fun *scaleRat-constraint* :: *rat* \Rightarrow *constraint* \Rightarrow *constraint* **where**

scaleRat-constraint *r cc* = (*if* $r = 0$ *then* *LEQ* *0 0* *else*
 (*case* *cc* *of*
 LEQ *p c* \Rightarrow
 (*if* ($r < 0$) *then* *GEQ* ($r * R$ *p*) ($r * R$ *c*) *else* *LEQ* ($r * R$ *p*) ($r * R$ *c*))
 | *LT* *p c* \Rightarrow
 (*if* ($r < 0$) *then* *GT* ($r * R$ *p*) ($r * R$ *c*) *else* *LT* ($r * R$ *p*) ($r * R$ *c*))
 | *GEQ* *p c* \Rightarrow
 (*if* ($r > 0$) *then* *GEQ* ($r * R$ *p*) ($r * R$ *c*) *else* *LEQ* ($r * R$ *p*) ($r * R$ *c*))
 | *GT* *p c* \Rightarrow
 (*if* ($r > 0$) *then* *GT* ($r * R$ *p*) ($r * R$ *c*) *else* *LT* ($r * R$ *p*) ($r * R$ *c*))
 | *EQ* *p c* \Rightarrow *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 ..

end

lemma *sat-scale-rat*: **assumes** ($v :: \text{rat valuation}$) $\models_c c$

shows $v \models_c (r * R c)$

proof –

have $r < 0 \vee r = 0 \vee r > 0$ **by** *auto*

then show *?thesis* **using** *assms* **by** (*cases c, auto simp: right-diff-distrib*
valuate-minus valuate-scaleRat scaleRat-leq1 scaleRat-leq2 valuate-zero)

qed

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* $C = (\exists d \text{ rel.}$
 $(\forall (r,c) \in \text{set } C. c \in \text{cs} \wedge \text{is-le } (r *R c) \wedge r \neq 0) \wedge$
 $(\sum (r,c) \leftarrow C. \text{lec-of-constraint } (r *R c)) = \text{Le-Constraint rel } 0 d \wedge$
 $(\text{rel} = \text{Leq-Rel} \wedge d < 0 \vee \text{rel} = \text{Lt-Rel} \wedge d \leq 0))$

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

assume $\exists v. v \models_{cs} cs$

then obtain v **where** $*$: $v \models_{cs} cs$ **by** *auto*

obtain $d \text{ rel}$ **where**

isleg: $(\forall (r,c) \in \text{set } C. c \in \text{cs} \wedge \text{is-le } (r *R c) \wedge r \neq 0)$ **and**

leq: $(\sum (r,c) \leftarrow C. \text{lec-of-constraint } (r *R c)) = \text{Le-Constraint rel } 0 d$ **and**

choice: $\text{rel} = \text{Lt-Rel} \wedge d \leq 0 \vee \text{rel} = \text{Leq-Rel} \wedge d < 0$ **using** *assms farkas-coefficients-def*

by *blast*

{

fix $r \ c$

assume $c: (r,c) \in \text{set } C$

from $c * \text{isleg}$ **have** $v \models_c c$ **by** *auto*

hence $v: v \models_c (r *R c)$ **by** (*rule sat-scale-rat*)

from $c \text{ isleg}$ **have** *is-le* $(r *R c)$ **by** *auto*

from *lec-of-constraint*[*OF this*] v

have $v \models_{le} \text{lec-of-constraint } (r *R c)$ **by** *blast*

} **note** $v = \text{this}$

have $v \models_{le} \text{Le-Constraint rel } 0 d$ **unfolding** *leq*[*symmetric*]

by (*rule satisfies-sumlist-le-constraints, insert v, auto*)

then show *False* **using** *choice*

by (*cases rel, auto simp: valuate-zero*)

qed

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 $(\text{Le-Constraint Leq-Rel } p \ c) (\text{Le-Constraint Leq-Rel } q \ d) = (q = p$
 $\wedge d = \text{QDelta } c \ 0)$

| *compatible-cs* $(\text{Le-Constraint Lt-Rel } p \ c) (\text{Le-Constraint Leq-Rel } q \ d) = (q = p \wedge$
 $\text{qdfst } d = c)$

| compatible-cs - - = False

lemma compatible-cs-0-0: compatible-cs 0 0 **by** code-simp

lemma compatible-cs-plus: compatible-cs c1 d1 \implies compatible-cs c2 d2 \implies compatible-cs (c1 + c2) (d1 + d2)

by (cases c1; cases d1; cases c2; cases d2; cases lec-rel c1; cases lec-rel d1; cases lec-rel c2;

cases lec-rel d2; auto simp: plus-QDelta-def)

lemma unsat-farkas-coefficients: **assumes** $\nexists v. v \models_{cs} cs$

and fin: finite cs

shows $\exists C. \text{farkas-coefficients } cs \ C$

proof -

from finite-list[OF fin] **obtain** csl **where** cs: cs = set csl **by** blast

let ?csl = map (Pair ()) csl

let ?ns = (snd ' set (to-ns ?csl))

let ?nsl = concat (map constraint-to-qdelta-constraint csl)

have id: snd ' set ?csl = cs **unfolding** cs **by** force

have id2: ?ns = set ?nsl **unfolding** to-ns-def set-concat **by** force

from SolveExec'Default.to-ns-sat[of ?csl, unfolded id] **assms**

have unsat: $\nexists v. \langle v \rangle \models_{nss} ?ns$ **by** metis

have fin: finite ?ns **by** auto

have $\nexists v. v \models_{nss} ?ns$

proof

assume $\exists v. v \models_{nss} ?ns$

then obtain v **where** model: v $\models_{nss} ?ns$ **by** blast

let ?v = Mapping.Mapping ($\lambda x. \text{Some } (v \ x)$)

have v = $\langle ?v \rangle$ **by** (intro ext, auto simp: map2fun-def Mapping.lookup.abs-eq)

from model this unsat **show** False **by** metis

qed

from farkas-coefficients-ns[OF fin] this id2 **obtain** N **where**

farkas: farkas-coefficients-ns (set ?nsl) N **by** metis

from this[unfolded farkas-coefficients-ns-def]

obtain d **where**

is-leq: $\bigwedge a \ n. (a, n) \in \text{set } N \implies n \in \text{set } ?nsl \wedge \text{is-leq-ns } (a \ *R \ n) \wedge a \neq 0$ **and**

sum: $(\sum (a, n) \leftarrow N. \text{lec-of-nsc } (a \ *R \ n)) = \text{Le-Constraint Leq-Rel } 0 \ d$ **and**

d0: d < 0 **by** blast

let ?prop = $\lambda \ NN \ C. (\forall (a, c) \in \text{set } C. c \in cs \wedge \text{is-le } (a \ *R \ c) \wedge a \neq 0)$

$\wedge \text{compatible-cs } (\sum (a, c) \leftarrow C. \text{lec-of-constraint } (a \ *R \ c))$

$(\sum (a, n) \leftarrow NN. \text{lec-of-nsc } (a \ *R \ n))$

have set NN \subseteq set N $\implies \exists C. ?prop \ NN \ C$ **for** NN

proof (induct NN)

case Nil

have ?prop Nil Nil **by** (simp add: compatible-cs-0-0)

thus ?case **by** blast

next

case (Cons an NN)

obtain $a\ n$ **where** $an: an = (a,n)$ **by** *force*
from *Cons an* **obtain** C **where** $IH: ?prop\ NN\ C$ **and** $n: (a,n) \in set\ N$ **by**
auto
have $compat-CN: compatible-cs\ (\sum (f, c) \leftarrow C. lec-of-constraint\ (f *R\ c))$
 $(\sum (a,n) \leftarrow NN. lec-of-nsc\ (a *R\ n))$
using IH **by** *blast*
from $n\ is-leq$ **obtain** c **where** $c: c \in cs$ **and** $nc: n \in set\ (constraint-to-qdelta-constraint\ c)$
unfolding cs **by** *force*
from $is-leq[OF\ n]$ **have** $is-leq: is-leq-ns\ (a *R\ n) \wedge a \neq 0$ **by** *blast*
have $is-less: is-le\ (a *R\ c)$ **and**
 $a0: a \neq 0$ **and**
 $compat-cn: compatible-cs\ (lec-of-constraint\ (a *R\ c))\ (lec-of-nsc\ (a *R\ n))$
by (*atomize(full), cases c, insert is-leq nc, auto simp: QDelta-0-0 scaleRat-QDelta-def qdsnd-0 qdfst-0*)
let $?C = Cons\ (a, c)\ C$
let $?N = Cons\ (a, n)\ NN$
have $?prop\ ?N\ ?C$ **unfolding** an
proof (*intro conjI*)
show $\forall (a,c) \in set\ ?C. c \in cs \wedge is-le\ (a *R\ c) \wedge a \neq 0$ **using** $IH\ is-less\ a0$
 c **by** *auto*
show $compatible-cs\ (\sum (a, c) \leftarrow ?C. lec-of-constraint\ (a *R\ c))\ (\sum (a,n) \leftarrow ?N. lec-of-nsc\ (a *R\ n))$
using $compatible-cs-plus[OF\ compat-cn\ compat-CN]$ **by** *simp*
qed
thus $?case$ **unfolding** an **by** *blast*
qed
from $this[OF\ subset-refl, unfolded\ sum]$
obtain C **where**
 $is-less: (\forall (a, c) \in set\ C. c \in cs \wedge is-le\ (a *R\ c) \wedge a \neq 0)$ **and**
 $compat: compatible-cs\ (\sum (f, c) \leftarrow C. lec-of-constraint\ (f *R\ c))\ (Le-Constraint\ Leq-Rel\ 0\ d)$
 $(is\ compatible-cs\ ?sum\ -)$
by *blast*
obtain $rel\ p\ e$ **where** $?sum = Le-Constraint\ rel\ p\ e$ **by** (*cases ?sum*)
with $compat$ **have** $sum: ?sum = Le-Constraint\ rel\ 0\ e$ **by** (*cases rel, auto*)
have $e: (rel = Leq-Rel \wedge e < 0 \vee rel = Lt-Rel \wedge e \leq 0)$ **using** $compat[unfolded\ sum]\ d0$
by (*cases rel, auto simp: less-QDelta-def qdfst-0 qdsnd-0*)
show $?thesis$ **unfolding** $farkas-coefficients-def$
by (*intro exI conjI, rule is-less, rule sum, insert e, auto*)
qed

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 $(\exists\ C. farkas-coefficients\ cs\ C) \longleftrightarrow (\nexists\ v. v \models_{cs}\ cs)$
using $farkas-coefficients-unsat\ unsat-farkas-coefficients[OF\ -\ assms]$ **by** *blast*

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 $0 < \text{const}$ 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 ( $\nexists v. \forall c \in cs. v \models_{le} c$ )  $\longleftrightarrow$ 
    ( $\exists C \text{ const. } (\forall (r, c) \in \text{set } C. r > 0 \wedge c \in cs)$ 
       $\wedge (\sum (r, c) \leftarrow C. \text{Leqc } (r *R \text{lec-poly } c) (r *R \text{lec-const } c)) = \text{Leqc } 0 \text{ const}$ 
       $\wedge \text{const} < 0$ )
    (is ?lhs = ?rhs)
proof -
  {
    fix c
    assume  $c \in cs$ 
    with only-non-strict have lec-rel c = Leq-Rel by auto
    then have  $\exists p \text{ const. } c = \text{Leqc } p \text{ const}$  by (cases c, auto)
  } note leqc = this
  let ?to-ns =  $\lambda c. \text{LEQ-ns } (\text{lec-poly } c) (\text{lec-const } c)$ 
  let ?ns = ?to-ns ' cs
  from fin have fin: finite ?ns by auto
  have  $v \models_{nss} ?ns \longleftrightarrow (\forall c \in cs. v \models_{le} c)$  for v using leqc by fastforce
  hence ?lhs = ( $\nexists v. v \models_{nss} ?ns$ ) by simp
  also have  $\dots = (\exists C. \text{farkas-coefficients-ns } ?ns C)$  unfolding farkas-coefficients-ns[OF
fin] ..
  also have  $\dots = ?rhs$ 
  proof
    assume  $\exists C. \text{farkas-coefficients-ns } ?ns C$ 
    then obtain C const where is-leq:  $\forall (s, n) \in \text{set } C. n \in ?ns \wedge \text{is-leq-ns } (s$ 
     $*R n) \wedge s \neq 0$ 
    and sum:  $(\sum (s, n) \leftarrow C. \text{lec-of-nsc } (s *R n)) = \text{Leqc } 0 \text{ const}$ 
    and c0:  $\text{const} < 0$  unfolding farkas-coefficients-ns-def by blast
    let ?C = map ( $\lambda (s, n). (s, \text{lec-of-nsc } n)$ ) C
    show ?rhs

```

```

proof (intro exI[of - ?C] exI[of - const] conjI c0, unfold sum[symmetric]
map-map o-def set-map,
intro ballI, clarify)
{
  fix s n
  assume sn: (s, n) ∈ set C
  with is-leq
  have n-ns: n ∈ ?ns and is-leq: is-leq-ns (s *R n) s ≠ 0 by force+
  from n-ns obtain c where c: c ∈ cs and n: n = LEQ-ns (lec-poly c)
(lec-const c) by auto
  with leqc[OF c] obtain p d where cs: Leqc p d ∈ cs and n: n = LEQ-ns
p d by auto
  from is-leq[unfolded n] have s0: s > 0 by (auto split: if-splits)
  let ?n = lec-of-nsc n
  from cs n have mem: ?n ∈ cs by auto
  show 0 < s ∧ ?n ∈ cs using s0 mem by blast
  have Leqc (s *R lec-poly ?n) (s *R lec-const ?n) = lec-of-nsc (s *R n)
  unfolding n using s0 by simp
} note id = this
show (∑ x←C. case case x of (s, n) ⇒ (s, lec-of-nsc n) of
(r, c) ⇒ Leqc (r *R lec-poly c) (r *R lec-const c)) =
(∑ (s, n)←C. lec-of-nsc (s *R n)) (is sum-list (map ?f C) = sum-list
(map ?g C))
proof (rule arg-cong[of - - sum-list], rule map-cong[OF refl])
fix pair
assume mem: pair ∈ set C
then obtain s n where pair: pair = (s,n) by force
show ?f pair = ?g pair unfolding pair split using id[OF mem[unfolded
pair]] .
qed
qed
next
assume ?rhs
then obtain C const
where C: ∧ r c. (r,c) ∈ set C ⇒ 0 < r ∧ c ∈ cs
and sum: (∑ (r, c)←C. Leqc (r *R lec-poly c) (r *R lec-const c)) = Leqc
0 const
and const: const < 0
by blast
let ?C = map (λ (r,c). (r, ?to-ns c)) C
show ∃ C. farkas-coefficients-ns ?ns C unfolding farkas-coefficients-ns-def
proof (intro exI[of - ?C] exI[of - const] conjI const, unfold sum[symmetric])
show ∀ (s, n)∈set ?C. n ∈ ?ns ∧ is-leq-ns (s *R n) ∧ s ≠ 0 using C by
fastforce
show (∑ (s, n)←?C. lec-of-nsc (s *R n))
= (∑ (r, c)←C. Leqc (r *R lec-poly c) (r *R lec-const c))
unfolding map-map o-def
by (rule arg-cong[of - - sum-list], rule map-cong[OF refl], insert C, force)
qed

```

qed
 finally show *?thesis* .
 qed

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 $(\nexists v. \forall c \in cs. v \models_{le} c) \longleftrightarrow$
 $(\exists C \text{ const rel. } (\forall (r, c) \in \text{set } C. r > 0 \wedge c \in cs)$
 $\wedge (\sum (r, c) \leftarrow C. \text{Le-Constraint } (lec\text{-rel } c) (r *R \text{lec-poly } c) (r *R \text{lec-const}$
 $c)))$
 $= \text{Le-Constraint rel } 0 \text{ const}$
 $\wedge (\text{rel} = \text{Leq-Rel} \wedge \text{const} < 0 \vee \text{rel} = \text{Lt-Rel} \wedge \text{const} \leq 0)$
 (is *?lhs* = *?rhs*)

proof –

let *?to-cs* = $\lambda c. (\text{case } lec\text{-rel } c \text{ of } \text{Leq-Rel} \Rightarrow \text{LEQ} \mid - \Rightarrow \text{LT}) (lec\text{-poly } c) (lec\text{-const } c)$
 have *to-cs*: $v \models_c ?to\text{-cs } c \longleftrightarrow v \models_{le} c$ for *v c* by (cases *c*, cases *lec-rel c*, auto)
 let *?cs* = *?to-cs* ' *cs*
 from *fin* have *fin*: finite *?cs* by auto
 have $v \models_{cs} ?cs \longleftrightarrow (\forall c \in cs. v \models_{le} c)$ for *v* using *to-cs* by auto
 hence *?lhs* = $(\nexists v. v \models_{cs} ?cs)$ by simp
 also have ... = $(\exists C. \text{farkas-coefficients } ?cs C)$ unfolding *farkas-coefficients*[OF *fin*] ..
 also have ... = *?rhs*

proof

assume $\exists C. \text{farkas-coefficients } ?cs C$
 then obtain *C* const rel where *is-leq*: $\forall (s, n) \in \text{set } C. n \in ?cs \wedge \text{is-le } (s *R n) \wedge s \neq 0$
 and *sum*: $(\sum (s, n) \leftarrow C. \text{lec-of-constraint } (s *R n)) = \text{Le-Constraint rel } 0$
 const
 and *c0*: $(\text{rel} = \text{Leq-Rel} \wedge \text{const} < 0 \vee \text{rel} = \text{Lt-Rel} \wedge \text{const} \leq 0)$
 unfolding *farkas-coefficients-def* by blast
 let *?C* = map $(\lambda (s, n). (s, \text{lec-of-constraint } n)) C$
 show *?rhs*
proof (intro *exI*[of - *?C*] *exI*[of - *const*] *exI*[of - *rel*] *conjI* *c0*, unfold *map-map*
o-def set-map sum[*symmetric*],
 intro *ballI*, clarify)
 {
 fix *s n*
 assume *sn*: $(s, n) \in \text{set } C$

```

with is-leq
have n-ns:  $n \in ?cs$  and is-leq: is-le ( $s *R n$ ) and s0:  $s \neq 0$  by force+
from n-ns obtain c where  $c \in cs$  and  $n = ?to-cs\ c$  by auto
from is-leq[unfolded n] have  $s \geq 0$  by (cases lec-rel c, auto split: if-splits)
with s0 have s0:  $s > 0$  by auto
let  $?c = lec-of-constraint\ n$ 
from  $c\ n$  have mem:  $?c \in cs$  by (cases c, cases lec-rel c, auto)
show  $0 < s \wedge ?c \in cs$  using s0 mem by blast
have lec-of-constraint ( $s *R n$ ) = Le-Constraint (lec-rel ?c) ( $s *R lec-poly\ ?c$ )
unfolding n using s0 by (cases c, cases lec-rel c, auto)
} note id = this
show ( $\sum x \leftarrow C. case\ case\ x\ of\ (s, n) \Rightarrow (s, lec-of-constraint\ n)\ of$ 
  ( $r, c$ )  $\Rightarrow Le-Constraint\ (lec-rel\ c)\ (r *R lec-poly\ c)\ (r *R lec-const\ c)$ ) =
  ( $\sum (s, n) \leftarrow C. lec-of-constraint\ (s *R n)$ )
  (is sum-list (map ?f C) = sum-list (map ?g C))
proof (rule arg-cong[of - - sum-list], rule map-cong[OF refl])
fix pair
assume mem:  $pair \in set\ C$ 
obtain  $r\ c$  where pair:  $pair = (r, c)$  by force
show  $?f\ pair = ?g\ pair$  unfolding pair split id[OF mem[unfolded pair]] ..
qed
qed
next
assume ?rhs
then obtain  $C\ const\ rel$ 
where  $C$ :  $\bigwedge r\ c. (r, c) \in set\ C \Longrightarrow 0 < r \wedge c \in cs$ 
and sum: ( $\sum (r, c) \leftarrow C. Le-Constraint\ (lec-rel\ c)\ (r *R lec-poly\ c)\ (r *R lec-const\ c)$ )
  = Le-Constraint rel 0 const
and const:  $rel = Leq-Rel \wedge const < 0 \vee rel = Lt-Rel \wedge const \leq 0$ 
by blast
let  $?C = map\ (\lambda (r, c). (r, ?to-cs\ c))\ C$ 
show  $\exists C. farkas-coefficients\ ?cs\ C$  unfolding farkas-coefficients-def
proof (intro exI[of - ?C] exI[of - const] exI[of - rel] conjI const, unfold sum[symmetric])
show  $\forall (s, n) \in set\ ?C. n \in ?cs \wedge is-le\ (s *R n) \wedge s \neq 0$  using  $C$  by (fastforce split: le-rel.splits)
show ( $\sum (s, n) \leftarrow ?C. lec-of-constraint\ (s *R n)$ )
  = ( $\sum (r, c) \leftarrow C. Le-Constraint\ (lec-rel\ c)\ (r *R lec-poly\ c)\ (r *R lec-const\ c)$ )
unfolding map-map o-def
by (rule arg-cong[of - - sum-list], rule map-cong[OF refl], insert C, fastforce split: le-rel.splits)
qed
qed
finally show ?thesis .
qed

```

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** $cs :: \text{rat le-constraint set}$
assumes *only-non-strict*: $\text{lec-rel } 'cs \subseteq \{\text{Leq-Rel}\}$
and *fin*: *finite cs*
shows $(\nexists v. \forall c \in cs. v \models_{le} c) \longleftrightarrow$
 $(\exists C \text{ const. } (\forall (r, c) \in \text{set } C. r > 0 \wedge c \in cs)$
 $\wedge (\sum (r, c) \leftarrow C. \text{Leqc } (r *R \text{lec-poly } c) (r *R \text{lec-const } c)) = \text{Leqc } 0 \text{ const}$
 $\wedge \text{const} < 0)$
(is - = ?rhs)
proof -
{
fix c
assume $c \in cs$
with *only-non-strict* **have** $\text{lec-rel } c = \text{Leq-Rel}$ **by** *auto*
then have $\exists p \text{ const. } c = \text{Leqc } p \text{ const}$ **by** (*cases c, auto*)
} note $\text{leqc} = \text{this}$
let $?lhs = \exists C \text{ const rel.}$
 $(\forall (r, c) \in \text{set } C. 0 < r \wedge c \in cs) \wedge$
 $(\sum (r, c) \leftarrow C. \text{Le-Constraint } (\text{lec-rel } c) (r *R \text{lec-poly } c) (r *R \text{lec-const } c))$
 $= \text{Le-Constraint rel } 0 \text{ const} \wedge$
 $(\text{rel} = \text{Leq-Rel} \wedge \text{const} < 0 \vee \text{rel} = \text{Lt-Rel} \wedge \text{const} \leq 0)$
show *?thesis unfolding Motzkin's-transposition-theorem[OF fin]*
proof
assume $?rhs$
then obtain $C \text{ const where } C: \bigwedge r c. (r, c) \in \text{set } C \implies 0 < r \wedge c \in cs$ **and**
 $\text{sum}: (\sum (r, c) \leftarrow C. \text{Leqc } (r *R \text{lec-poly } c) (r *R \text{lec-const } c)) = \text{Leqc } 0 \text{ const}$
and
 $\text{const}: \text{const} < 0$ **by** *blast*
show $?lhs$
proof (*intro exI[of - C] exI[of - const] exI[of - Leq-Rel] conjI*)
show $\forall (r, c) \in \text{set } C. 0 < r \wedge c \in cs$ **using** C **by** *force*
show $(\sum (r, c) \leftarrow C. \text{Le-Constraint } (\text{lec-rel } c) (r *R \text{lec-poly } c) (r *R \text{lec-const } c)) =$
 $\text{Leqc } 0 \text{ const}$ **unfolding** *sum[symmetric]*
by (*rule arg-cong[of - - sum-list], rule map-cong[OF refl], insert C, force*
dest!: leqc)
qed (*insert const, auto*)
next
assume $?lhs$
then obtain $C \text{ const rel where } C: \bigwedge r c. (r, c) \in \text{set } C \implies 0 < r \wedge c \in cs$
and
 $\text{sum}: (\sum (r, c) \leftarrow C. \text{Le-Constraint } (\text{lec-rel } c) (r *R \text{lec-poly } c) (r *R \text{lec-const } c))$
 $= \text{Le-Constraint rel } 0 \text{ const}$ **and**
 $\text{const}: \text{rel} = \text{Leq-Rel} \wedge \text{const} < 0 \vee \text{rel} = \text{Lt-Rel} \wedge \text{const} \leq 0$ **by** *blast*

```

have id: (∑ (r, c)←C. Le-Constraint (lec-rel c) (r *R lec-poly c) (r *R lec-const
c)) =
  (∑ (r, c)←C. Leqc (r *R lec-poly c) (r *R lec-const c)) (is - = ?sum)
  by (rule arg-cong[of - - sum-list], rule map-cong, auto dest!: C leqc)
  have lec-rel ?sum = Leq-Rel unfolding sum-list-lec by (auto simp add:
sum-list-Leq-Rel o-def)
  with sum[unfolded id] have rel: rel = Leq-Rel by auto
  with const have const: const < 0 by auto
  show ?rhs
  by (intro exI[of - C] exI[of - const] conjI const, insert sum id C rel, force+)
qed
qed

```

We also present slightly modified versions

```

lemma sum-list-map-filter-sum: fixes f :: 'a ⇒ '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)
by (induct xs, auto simp: ac-simps)

```

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 (∄ v. ∀ c ∈ cs. v ⊨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)
unfolding Farkas'-Lemma[OF only-non-strict fin]
proof ((standard; elim exE conjE), goal-cases)
case (2 C const)
from finite-distinct-list[OF fin] obtain csl where csl: set csl = cs and dist:
distinct csl
by auto
let ?list = filter (λ c. C c ≠ 0) csl
let ?C = map (λ c. (C c, c)) ?list
show ?case
proof (intro exI[of - ?C] exI[of - const] conjI)
have (∑ (r, c)←?C. Le-Constraint Leq-Rel (r *R lec-poly c) (r *R lec-const
c))
  = (∑ (r, c)←map (λc. (C c, c)) csl. Le-Constraint Leq-Rel (r *R lec-poly c)
(r *R lec-const c))
unfolding map-map
by (rule sum-list-map-filter, auto simp: zero-le-constraint-def)
also have ... = Le-Constraint Leq-Rel 0 const unfolding 2(2)[symmetric]
csl[symmetric]
unfolding sum.distinct-set-conv-list[OF dist] map-map o-def split ..
finally

```

```

    show  $(\sum (r, c) \leftarrow ?C. \text{Le-Constraint Leq-Rel } (r *R \text{ lec-poly } c) (r *R \text{ lec-const } c)) = \text{Le-Constraint Leq-Rel } 0 \text{ const}$ 
      by auto
    show  $\text{const} < 0$  by fact
    show  $\forall (r, c) \in \text{set } ?C. 0 < r \wedge c \in \text{cs}$  using 2(1) unfolding set-map set-filter
    csl by auto
  qed
next
  case (1 C const)
  define CC where  $CC = (\lambda c. \text{sum-list } (\text{map fst } (\text{filter } (\lambda rc. \text{snd } rc = c) C)))$ 
  show  $(\exists C \text{ const. } (\forall c \in \text{cs}. C c \geq 0))$ 
     $\wedge (\sum c \in \text{cs}. \text{Leqc } ((C c) *R \text{ lec-poly } c) ((C c) *R \text{ lec-const } c)) = \text{Leqc } 0$ 
  const
     $\wedge \text{const} < 0$ 
  proof (intro exI[of - CC] exI[of - const] conjI)
    show  $\forall c \in \text{cs}. 0 \leq CC c$  unfolding CC-def using 1(1)
      by (force intro!: sum-list-nonneg)
    show  $\text{const} < 0$  by fact
    from 1 have snd:  $\text{snd} \text{ ' set } C \subseteq \text{cs}$  by auto
    show  $(\sum c \in \text{cs}. \text{Le-Constraint Leq-Rel } (CC c *R \text{ lec-poly } c) (CC c *R \text{ lec-const } c)) = \text{Le-Constraint Leq-Rel } 0 \text{ const}$ 
      unfolding 1(2)[symmetric] using fin snd unfolding CC-def
    proof (induct cs arbitrary: C rule: finite-induct)
      case empty
        hence C:  $C = []$  by auto
        thus ?case by simp
      next
        case *: (insert c cs C)
        let ?D =  $\text{filter } (\text{Not } \circ (\lambda rc. \text{snd } rc = c)) C$ 
        from * have snd ' set ?D  $\subseteq \text{cs}$  by auto
        note IH = *(3)[OF this]
        have id:  $(\sum a \leftarrow ?D. \text{case } a \text{ of } (r, c) \Rightarrow \text{Le-Constraint Leq-Rel } (r *R \text{ lec-poly } c) (r *R \text{ lec-const } c)) =$ 
           $(\sum (r, c) \leftarrow ?D. \text{Le-Constraint Leq-Rel } (r *R \text{ lec-poly } c) (r *R \text{ lec-const } c))$ 
          by (induct C, force+)
        show ?case
          unfolding sum.insert[OF *(1,2)]
          unfolding sum-list-map-filter-sum[of -  $\lambda rc. \text{snd } rc = c$  C, symmetric]
        proof (rule arg-cong2[of - - - (+)], goal-cases)
          case 2
            show ?case unfolding IH[symmetric]
              by (rule sum.cong, insert *(2,1), auto intro!: arg-cong[of - -  $\lambda xs. \text{sum-list } (\text{map } - xs)$ ], (induct C, auto)+)
          next
            case 1
              show ?case
                proof (rule sym, induct C)
                  case (Cons rc C)
                    thus ?case by (cases rc, cases snd rc = c, auto simp: field-simps)
                end
            end
        end
    end
  end

```

```

scaleRat-left-distrib)
  qed (auto simp: zero-le-constraint-def)
  qed
  qed
  qed
  qed

```

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

```

lemma Farkas'-Lemma-indexed: fixes c :: nat  $\Rightarrow$  rat le-constraint
assumes only-non-strict: lec-rel ' c ' Is  $\subseteq$  {Leq-Rel}
and fin: finite Is
shows ( $\nexists$  v.  $\forall$  i  $\in$  Is. v  $\models_{le}$  c i)  $\longleftrightarrow$ 
  ( $\exists$  C const. ( $\forall$  i  $\in$  Is. C i  $\geq$  0)
     $\wedge$  ( $\sum$  i  $\in$  Is. Leqc ((C i) *R lec-poly (c i)) ((C i) *R lec-const (c i))) =
    Leqc 0 const
     $\wedge$  const < 0)
proof -
let ?C = c ' Is
have fin: finite ?C using fin by auto
have ( $\nexists$  v.  $\forall$  i  $\in$  Is. v  $\models_{le}$  c i) = ( $\nexists$  v.  $\forall$  cc  $\in$  ?C. v  $\models_{le}$  cc) by force
also have ... = ( $\exists$  C const. ( $\forall$  i  $\in$  Is. C i  $\geq$  0)
   $\wedge$  ( $\sum$  i  $\in$  Is. Leqc ((C i) *R lec-poly (c i)) ((C i) *R lec-const (c i))) =
  Leqc 0 const
   $\wedge$  const < 0) (is ?l = ?r)
proof
assume ?r
then obtain C const where r: ( $\forall$  i  $\in$  Is. C i  $\geq$  0)
and eq: ( $\sum$  i  $\in$  Is. Leqc ((C i) *R lec-poly (c i)) ((C i) *R lec-const (c
i))) = Leqc 0 const
and const < 0 by auto
from finite-distinct-list[OF <finite Is>]
obtain Isl where isl: set Isl = Is and dist: distinct Isl by auto
let ?CC = filter ( $\lambda$  rc. fst rc  $\neq$  0) (map ( $\lambda$  i. (C i, c i)) Isl)
show ?l unfolding Farkas'-Lemma[OF only-non-strict fin]
proof (intro exI[of - ?CC] exI[of - const] conjI)
show const < 0 by fact
show  $\forall$  (r, ca)  $\in$  set ?CC. 0 < r  $\wedge$  ca  $\in$  ?C using r(1) isl by auto
show ( $\sum$  (r, c)  $\leftarrow$  ?CC. Le-Constraint Leq-Rel (r *R lec-poly c) (r *R lec-const
c)) =
  Le-Constraint Leq-Rel 0 const unfolding eq[symmetric]
by (subst sum-list-map-filter, force simp: zero-le-constraint-def,
  unfold map-map o-def, subst sum-list-distinct-conv-sum-set[OF dist], rule
sum.cong, auto simp: isl)
qed
next
assume ?l
from this[unfolded Farkas'-Lemma-set-sum[OF only-non-strict fin]]
obtain C const where nonneg: ( $\forall$  c  $\in$  ?C. 0  $\leq$  C c)

```

```

and sum: ( $\sum c \in ?C. \text{Le-Constraint Leq-Rel } (C \ c \ *R \ \text{lec-poly } c) (C \ c \ *R \ \text{lec-const } c)$ ) =
  Le-Constraint Leq-Rel 0 const
and const: const < 0
by blast
define I where I = ( $\lambda i. (C \ (c \ i) / \text{rat-of-nat } (\text{card } (Is \cap \{j. c \ i = c \ j\})))$ )
show ?r
proof (intro exI[of - I] exI[of - const] conjI const)
  show  $\forall i \in Is. 0 \leq I \ i$  using nonneg unfolding I-def by auto
  show ( $\sum i \in Is. \text{Le-Constraint Leq-Rel } (I \ i \ *R \ \text{lec-poly } (c \ i)) (I \ i \ *R \ \text{lec-const } (c \ i))$ ) =
    Le-Constraint Leq-Rel 0 const unfolding sum[symmetric]
    unfolding sum.image-gen[OF <finite Is>, of - c]
proof (rule sum.cong[OF refl], goal-cases)
  case (1 cc)
    define II where II = ( $Is \cap \{j. cc = c \ j\}$ )
    from 1 have II  $\neq \{\}$  unfolding II-def by auto
    moreover have finII: finite II using <finite Is> unfolding II-def by auto
    ultimately have card: card II  $\neq 0$  by auto
    let ?C =  $\lambda II. \text{rat-of-nat } (\text{card } II)$ 
    define ii where ii =  $C \ cc / \text{rat-of-nat } (\text{card } II)$ 
    have ( $\sum i \in \{x \in Is. c \ x = cc\}. \text{Le-Constraint Leq-Rel } (I \ i \ *R \ \text{lec-poly } (c \ i)) (I \ i \ *R \ \text{lec-const } (c \ i))$ )
      = ( $\sum i \in II. \text{Le-Constraint Leq-Rel } (ii \ *R \ \text{lec-poly } cc) (ii \ *R \ \text{lec-const } cc)$ )
      unfolding I-def ii-def II-def by (rule sum.cong, auto)
    also have  $\dots = \text{Le-Constraint Leq-Rel } ((?C \ II \ * \ ii) \ *R \ \text{lec-poly } cc) ((?C \ II \ * \ ii) \ *R \ \text{lec-const } cc)$ 
    using finII by (induct II rule: finite-induct, auto simp: zero-le-constraint-def field-simps scaleRat-left-distrib)
    also have  $?C \ II \ * \ ii = C \ cc$  unfolding ii-def using card by auto
    finally show ?case .
  qed
qed
qed
finally show ?thesis .
qed

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* \Rightarrow *linear-poly* **is**
 $\lambda v x.$ if ($x < \text{dim-vec } v$) then $v \$ x$ else 0
by *auto*

definition *val-of-vec* :: *rat vec* \Rightarrow *rat valuation* **where**
val-of-vec $v x = v \$ x$

lemma *valuate-poly-of-vec*: **assumes** $w \in \text{carrier-vec } n$
and $v \in \text{carrier-vec } n$
shows *valuate* (*poly-of-vec* v) (*val-of-vec* w) = $v \cdot w$
using *assms* **by** (*transfer*, *auto simp: val-of-vec-def scalar-prod-def intro: sum.mono-neutral-left*)

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

lemma *constraints-of-mat-vec-solution-main*: **assumes** $A: A \in \text{carrier-mat } nr \ nc$
and $x: x \in \text{carrier-vec } nc$
and $b: b \in \text{carrier-vec } nr$
and *sol*: $A *_v x \leq b$
and $c: c \in \text{constraints-of-mat-vec } A \ b$
shows *val-of-vec* $x \models_{le} c$
proof –
from $c[\text{unfolded } \text{constraints-of-mat-vec-def}] \ A$ **obtain** i **where**
 $i: i < nr$ **and** $c: c = \text{Leqc } (\text{poly-of-vec } (\text{row } A \ i)) \ (b \$ i)$ **by** *auto*
from $i \ A$ **have** $ri: \text{row } A \ i \in \text{carrier-vec } nc$ **by** *auto*
from *sol* $i \ A \ x \ b$ **have** *sol*: $(A *_v x) \$ i \leq b \$ i$ **unfolding** *less-eq-vec-def* **by**
auto
thus *val-of-vec* $x \models_{le} c$ **unfolding** *c satisfiable-le-constraint.simps rel-of.simps*
valuate-poly-of-vec[OF x ri] **using** $A \ x \ i$ **by** *auto*
qed

lemma *vars-poly-of-vec*: $\text{vars } (\text{poly-of-vec } v) \subseteq \{ 0 .. < \text{dim-vec } v \}$
by (*transfer'*, *auto*)

lemma *finite-constraints-of-mat-vec*: *finite* (*constraints-of-mat-vec* $A \ b$)
unfolding *constraints-of-mat-vec-def* **by** *auto*

lemma *lec-rec-constraints-of-mat-vec*: *lec-rel* ' *constraints-of-mat-vec* $A \ b \subseteq \{ \text{Leq-Rel} \}$
unfolding *constraints-of-mat-vec-def* **by** *auto*

lemma *constraints-of-mat-vec-solution-1*:
assumes $A: A \in \text{carrier-mat } nr \ nc$
and $b: b \in \text{carrier-vec } nr$
and *sol*: $\exists x \in \text{carrier-vec } nc. A *_v x \leq b$

shows $\exists v. \forall c \in \text{constraints-of-mat-vec } A \ b. v \models_{le} c$
using *constraints-of-mat-vec-solution-main*[*OF A - b -*] *sol* **by** *blast*

lemma *constraints-of-mat-vec-solution-2*:

assumes *A*: $A \in \text{carrier-mat } nr \ nc$

and *b*: $b \in \text{carrier-vec } nr$

and *sol*: $\exists v. \forall c \in \text{constraints-of-mat-vec } A \ b. v \models_{le} c$

shows $\exists x \in \text{carrier-vec } nc. A *_v x \leq b$

proof –

from *sol* **obtain** *v* **where** *sol*: $v \models_{le} c$ **if** $c \in \text{constraints-of-mat-vec } A \ b$ **for** *c*
by *auto*

define *x* **where** $x = \text{vec } nc \ (\lambda \ i. \ v \ i)$

show *?thesis*

proof (*intro* *beXI*[*of - x*])

show *x*: $x \in \text{carrier-vec } nc$ **unfolding** *x-def* **by** *auto*

have *row A i* $\cdot x \leq b \ \$ \ i$ **if** $i < nr$ **for** *i*

proof –

from *that* **have** *Leqc* (*poly-of-vec* (*row A i*)) ($b \ \$ \ i$) $\in \text{constraints-of-mat-vec } A \ b$

unfolding *constraints-of-mat-vec-def* **using** *A* **by** *auto*

from *sol*[*OF this, simplified*] **have** *valuate* (*poly-of-vec* (*row A i*)) $v \leq b \ \$ \ i$

by *simp*

also **have** *valuate* (*poly-of-vec* (*row A i*)) $v = \text{valuate} \ (\text{poly-of-vec} \ (\text{row } A \ i))$
(*val-of-vec x*)

by (*rule* *valuate-depend*, *insert A that*,

auto simp: x-def val-of-vec-def dest!: set-mp[*OF vars-poly-of-vec*])

also **have** $\dots = \text{row } A \ i \cdot x$

by (*subst* *valuate-poly-of-vec*[*OF x*], *insert that A x, auto*)

finally **show** *?thesis* .

qed

thus $A *_v x \leq b$ **unfolding** *less-eq-vec-def* **using** *x A b* **by** *auto*

qed

qed

lemma *constraints-of-mat-vec-solution*:

assumes *A*: $A \in \text{carrier-mat } nr \ nc$

and *b*: $b \in \text{carrier-vec } nr$

shows $(\exists x \in \text{carrier-vec } nc. A *_v x \leq b) =$

$(\exists v. \forall c \in \text{constraints-of-mat-vec } A \ b. v \models_{le} c)$

using *constraints-of-mat-vec-solution-1*[*OF assms*] *constraints-of-mat-vec-solution-2*[*OF assms*]

by *blast*

lemma *farkas-lemma-matrix*: **fixes** *A* :: *rat mat*

assumes *A*: $A \in \text{carrier-mat } nr \ nc$

and *b*: $b \in \text{carrier-vec } nr$

shows $(\exists x \in \text{carrier-vec } nc. A *_v x \leq b) \longleftrightarrow$

$(\forall y. y \geq 0_v \ nr \longrightarrow \text{mat-of-row } y * A = 0_m \ 1 \ nc \longrightarrow y \cdot b \geq 0)$

proof –

```

define cs where cs = constraints-of-mat-vec A b
have fin: finite {0 ..< nr} by auto
have dim: dim-row A = nr using A by simp
have sum-id: ( $\sum i = 0..<nr. f i$ ) = sum-list (map f [0..<nr]) for f
  by (subst sum-list-distinct-conv-sum-set, auto)
have ( $\exists x \in$  carrier-vec nc.  $A *_{\mathbf{v}}$  x  $\leq$  b) =
  ( $\neg (\nexists v. \forall c \in$  cs.  $v \models_{le}$  c))
  unfolding constraints-of-mat-vec-solution[OF assms] cs-def by simp
also have ... = ( $\neg (\nexists v. \forall i \in \{0..<nr\}. v \models_{le}$  Le-Constraint Leq-Rel (poly-of-vec
(row A i)) (b $ i)))
  unfolding cs-def constraints-of-mat-vec-def dim by auto
also have ... = ( $\nexists C$ .
  ( $\forall i \in \{0..<nr\}. 0 \leq C i$ )  $\wedge$ 
  ( $\sum i = 0..<nr. (C i *_{\mathbf{R}}$  poly-of-vec (row A i))) = 0  $\wedge$ 
  ( $\sum i = 0..<nr. (C i * b$  $ i)) < 0)
  unfolding Farkas'-Lemma-indexed[OF
lec-rec-constraints-of-mat-vec[unfolded constraints-of-mat-vec-def], of A b,
unfolded dim, OF fin] sum-id sum-list-lec le-constraint.simps
sum-list-Leq-Rel map-map o-def unfolding sum-id[symmetric] by simp
also have ... = ( $\forall C. (\forall i \in \{0..<nr\}. 0 \leq C i) \longrightarrow$ 
  ( $\sum i = 0..<nr. (C i *_{\mathbf{R}}$  poly-of-vec (row A i))) = 0  $\longrightarrow$ 
  ( $\sum i = 0..<nr. (C i * b$  $ i))  $\geq$  0)
  using not-less by blast
also have ... = ( $\forall y. y \geq 0_{\mathbf{v}}$  nr  $\longrightarrow$  mat-of-row y * A = 0m 1 nc  $\longrightarrow$  y · b  $\geq$ 
0)
proof ((standard; intro allI impI), goal-cases)
  case *: (1 y)
  define C where C = ( $\lambda i. y$  $ i)
  note main = *(1)[rule-format, of C]
  from *(2) have y: y  $\in$  carrier-vec nr and nonneg:  $\bigwedge i. i \in \{0..<nr\} \implies 0 \leq$ 
C i
  unfolding less-eq-vec-def C-def by auto
  have sum-0: ( $\sum i = 0..<nr. C i *_{\mathbf{R}}$  poly-of-vec (row A i)) = 0 unfolding
C-def
  unfolding zero-coeff-zero coeff-sum
proof
  fix v
  have ( $\sum i = 0..<nr. \text{coeff} (y$  $ i *R poly-of-vec (row A i)) v) =
  ( $\sum i < nr. y$  $ i * coeff (poly-of-vec (row A i)) v) by (rule sum.cong,
auto)
  also have ... = 0
  proof (cases v < nc)
  case False
  have ( $\sum i < nr. y$  $ i * coeff (poly-of-vec (row A i)) v) =
  ( $\sum i < nr. y$  $ i * 0)
  by (rule sum.cong[OF refl], rule arg-cong[of - -  $\lambda x. - * x$ ], insert A False,
transfer, auto)
  also have ... = 0 by simp
  finally show ?thesis by simp

```

```

next
  case True
  have  $(\sum i < nr. y \$ i * coeff (poly-of-vec (row A i)) v) =$ 
     $(\sum i < nr. y \$ i * row A i \$ v)$ 
    by (rule sum.cong[OF refl], rule arg-cong[of - -  $\lambda x. - * x$ ], insert A True,
transfer, auto)
  also have ... = (mat-of-row y * A) $$ (0,v)
    unfolding times-mat-def scalar-prod-def
    using A y True by (auto intro: sum.cong)
  also have ... = 0 unfolding *(3) using True by simp
  finally show ?thesis .
qed
finally show  $(\sum i = 0..<nr. coeff (y \$ i *R poly-of-vec (row A i)) v) = 0$  .
qed
from main[OF nonneg sum-0] have le:  $0 \leq (\sum i = 0..<nr. C i * b \$ i)$  .
thus ?case using y b unfolding scalar-prod-def C-def by auto
next
case *: (2 C)
define y where y = vec nr C
have y:  $y \in carrier-vec nr$  unfolding y-def by auto
note main = *(1)[rule-format, of y]
from *(2) have y0:  $y \geq 0_v nr$  unfolding less-eq-vec-def y-def by auto
have prod0:  $mat-of-row y * A = 0_m 1 nc$ 
proof -
{
  fix j
  assume j:  $j < nc$ 
  from arg-cong[OF *(3), of  $\lambda x. coeff x j$ , unfolded coeff-sum]
  have 0 =  $(\sum i = 0..<nr. C i * coeff (poly-of-vec (row A i)) j)$  by simp
  also have ... =  $(\sum i = 0..<nr. C i * row A i \$ j)$ 
    by (rule sum.cong[OF refl], rule arg-cong[of - -  $\lambda x. - * x$ ], insert A j,
transfer, auto)
  also have ... =  $y \cdot col A j$  unfolding scalar-prod-def y-def using A j
    by (intro sum.cong, auto)
  finally have  $y \cdot col A j = 0$  by simp
}
thus ?thesis by (intro eq-matI, insert A y, auto)
qed
from main[OF y0 prod0] have 0  $\leq y \cdot b$  .
thus ?case unfolding scalar-prod-def y-def using b by auto
qed
finally show ?thesis .
qed

```

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

proof –

define B **where** $B = (-\ 1_m\ nc)\ @_r\ (A\ @_r\ -A)$
define b' **where** $b' = 0_v\ nc\ @_v\ (b\ @_v\ -b)$
define n **where** $n = nc + (nr + nr)$
have $id0: 0_v\ (nc + (nr + nr)) = 0_v\ nc\ @_v\ (0_v\ nr\ @_v\ 0_v\ nr)$ **by** $(intro\ eq\ vecI,$
 $auto)$
have $B: B \in carrier\ mat\ n\ nc$ **unfolding** $B\text{-def}\ n\text{-def}$ **using** A **by** $auto$
have $b': b' \in carrier\ vec\ n$ **unfolding** $b'\text{-def}\ n\text{-def}$ **using** b **by** $auto$
have $(\exists\ x \geq 0_v\ nc.\ A\ *_v\ x = b) = (\exists\ x.\ x \in carrier\ vec\ nc \wedge x \geq 0_v\ nc \wedge A$
 $*_v\ x = b)$
by $(rule\ arg\ cong[of\ -\ -\ Ex],\ intro\ ext,\ insert\ A\ b,\ auto\ simp:\ less\ eq\ vec\ def)$
also have $\dots = (\exists\ x \in carrier\ vec\ nc.\ x \geq 0_v\ nc \wedge A\ *_v\ x = b)$ **by** $blast$
also have $\dots = (\exists\ x \in carrier\ vec\ nc.\ 1_m\ nc\ *_v\ x \geq 0_v\ nc \wedge A\ *_v\ x \leq b \wedge A$
 $*_v\ x \geq b)$
by $(rule\ bex\ cong[OF\ refl],\ insert\ A\ b,\ auto)$
also have $\dots = (\exists\ x \in carrier\ vec\ nc.\ (-\ 1_m\ nc)\ *_v\ x \leq 0_v\ nc \wedge A\ *_v\ x \leq b$
 $\wedge (-\ A)\ *_v\ x \leq -b)$
by $(rule\ bex\ cong[OF\ refl],\ insert\ A\ b,\ auto\ simp:\ less\ eq\ vec\ def)$
also have $\dots = (\exists\ x \in carrier\ vec\ nc.\ B\ *_v\ x \leq b')$
by $(rule\ bex\ cong[OF\ refl],\ insert\ A\ b,\ unfold\ B\text{-def}\ b'\text{-def},$
 $subst\ append\ rows\ le[of\ -\],\ (auto)[4],\ intro\ conj\ cong[OF\ refl],\ subst\ ap\ pend\ rows\ le,\ auto)$
also have $\dots = (\forall\ y \geq 0_v\ n.\ mat\ of\ row\ y\ * B = 0_m\ 1\ nc \longrightarrow y \cdot b' \geq 0)$
by $(rule\ farkas\ lemma\ matrix[OF\ B\ b'])$
also have $\dots = (\forall\ y.\ y \in carrier\ vec\ n \longrightarrow y \geq 0_v\ n \longrightarrow mat\ of\ row\ y\ * B =$
 $0_m\ 1\ nc \longrightarrow y \cdot b' \geq 0)$
by $(intro\ arg\ cong[of\ -\ -\ All],\ intro\ ext,\ auto\ simp:\ less\ eq\ vec\ def)$
also have $\dots = (\forall\ y \in carrier\ vec\ n.\ y \geq 0_v\ n \longrightarrow mat\ of\ row\ y\ * B = 0_m\ 1\ nc$
 $\longrightarrow y \cdot b' \geq 0)$
by $blast$
also have $\dots = (\forall\ y1 \in carrier\ vec\ nc.\ \forall\ y2 \in carrier\ vec\ nr.\ \forall\ y3 \in carrier\ vec$
 $nr.$
 $0_v\ nc\ @_v\ (0_v\ nr\ @_v\ 0_v\ nr) \leq y1\ @_v\ y2\ @_v\ y3 \longrightarrow$
 $mat\ of\ row\ (y1\ @_v\ y2\ @_v\ y3)\ * ((-\ 1_m\ nc)\ @_r\ (A\ @_r\ -A)) = 0_m\ 1$
 nc
 $\longrightarrow 0 \leq (y1\ @_v\ y2\ @_v\ y3) \cdot (0_v\ nc\ @_v\ (b\ @_v\ -b)))$
unfolding $n\text{-def}\ all\ vec\ append\ id0\ b'\text{-def}\ B\text{-def}$ **by** $auto$
also have $\dots = (\forall\ y1 \in carrier\ vec\ nc.\ \forall\ y2 \in carrier\ vec\ nr.\ \forall\ y3 \in carrier\ vec$
 $nr.$
 $0_v\ nc \leq y1 \longrightarrow 0_v\ nr \leq y2 \longrightarrow 0_v\ nr \leq y3 \longrightarrow$
 $(-\ mat\ of\ row\ y1) +$
 $(mat\ of\ row\ y2\ * A - (mat\ of\ row\ y3\ * A)) = 0_m\ 1\ nc$
 $\longrightarrow y2 \cdot b - y3 \cdot b \geq 0)$
by $(intro\ ball\ cong[OF\ refl],\ subst\ append\ vec\ le,\ (auto)[2],\ subst\ append\ vec\ le,$
 $(auto)[2],\ insert\ A\ b,$
 $subst\ scalar\ prod\ append,\ (auto)[4],\ subst\ scalar\ prod\ append,\ (auto)[4],$
 $subst\ mat\ of\ row\ mult\ append\ rows,\ (auto)[4],$
 $subst\ mat\ of\ row\ mult\ append\ rows,\ (auto)[4],$
 $subst\ add\ uminus\ minus\ mat[symmetric],\ auto)$

also have ... = $(\forall y1 \in \text{carrier-vec } nc. \forall y2 \in \text{carrier-vec } nr. \forall y3 \in \text{carrier-vec } nr.$

$$\begin{aligned} &0_v \text{ nc} \leq y1 \longrightarrow 0_v \text{ nr} \leq y2 \longrightarrow 0_v \text{ nr} \leq y3 \longrightarrow \\ &\text{mat-of-row } y1 = \text{mat-of-row } y2 * A - \text{mat-of-row } y3 * A \\ &\longrightarrow y2 \cdot b - y3 \cdot b \geq 0) \end{aligned}$$

proof ((intro ball-cong[OF refl] arg-cong2[of - - - - (\longrightarrow)] refl, standard), goal-cases)
case (1 y1 y2 y3)

from arg-cong[OF 1(4), of $\lambda x. \text{mat-of-row } y1 + x$] **show** ?case **using** 1(1-3)

A

by (subst (asm) assoc-add-mat[symmetric], (auto)[3],
 subst (asm) add-uminus-minus-mat, (auto)[1],
 subst (asm) minus-r-inv-mat, force,
 subst (asm) right-add-zero-mat, force,
 subst (asm) left-add-zero-mat, force, auto)

next

case (2 y1 y2 y3)

show ?case **unfolding** 2(4) **using** 2(1-3) A

by (intro eq-matI, auto)

qed

also have ... = $(\forall y1 \in \text{carrier-vec } nc. \forall y2 \in \text{carrier-vec } nr. \forall y3 \in \text{carrier-vec } nr.$

$$\begin{aligned} &0_v \text{ nc} \leq y1 \longrightarrow 0_v \text{ nr} \leq y2 \longrightarrow 0_v \text{ nr} \leq y3 \longrightarrow \\ &\text{mat-of-row } y1 = \text{mat-of-row } (y2 - y3) * A \\ &\longrightarrow (y2 - y3) \cdot b \geq 0) \end{aligned}$$

by (intro ball-cong[OF refl] imp-cong refl
 arg-cong2[of - - - - (\leq)] arg-cong2[of - - - - (=)],
 subst minus-mult-distrib-mat[symmetric], insert A b, auto
 simp: minus-scalar-prod-distrib mat-of-rows-def
 intro!: arg-cong[of - - $\lambda x. x * \cdot$])

also have ... = $(\forall y1 \in \text{carrier-vec } nc. \forall y2 \in \text{carrier-vec } nr. \forall y3 \in \text{carrier-vec } nr.$

$$\begin{aligned} &0_v \text{ nc} \leq y1 \longrightarrow 0_v \text{ nr} \leq y2 \longrightarrow 0_v \text{ nr} \leq y3 \longrightarrow \\ &y1 = \text{row } (\text{mat-of-row } (y2 - y3) * A) 0 \\ &\longrightarrow (y2 - y3) \cdot b \geq 0) \end{aligned}$$

proof (intro ball-cong[OF refl] arg-cong2[of - - - - (\longrightarrow)] refl, standard, goal-cases)
case (1 y1 y2 y3)

from arg-cong[OF 1(4), of $\lambda x. \text{row } x 0$] 1(1-3) A

show ?case **by** auto

qed (insert A, auto)

also have ... = $(\forall y2 \in \text{carrier-vec } nr. \forall y3 \in \text{carrier-vec } nr.$

$0_v \text{ nc} \leq \text{row } (\text{mat-of-row } (y2 - y3) * A) 0 \longrightarrow 0_v \text{ nr} \leq y2 \longrightarrow 0_v$
 $\text{nr} \leq y3 \longrightarrow$

$$\begin{aligned} &\text{row } (\text{mat-of-row } (y2 - y3) * A) 0 \in \text{carrier-vec } nc \\ &\longrightarrow (y2 - y3) \cdot b \geq 0) \text{ by blast} \end{aligned}$$

also have ... = $(\forall y2 \in \text{carrier-vec } nr. \forall y3 \in \text{carrier-vec } nr.$

$0_v \text{ nc} \leq \text{row } (\text{mat-of-row } (y2 - y3) * A) 0 \longrightarrow 0_v \text{ nr} \leq y2 \longrightarrow 0_v$
 $\text{nr} \leq y3$

$$\longrightarrow (y2 - y3) \cdot b \geq 0)$$

by (intro ball-cong[OF refl] arg-cong2[of - - - - (\longrightarrow)] refl, insert A,

```

      auto simp: row-def)
also have ... = ( $\forall y \in \text{carrier-vec nr. row (mat-of-row } y * A) \ 0 \geq 0_v \ nc \longrightarrow y \cdot b \geq 0$ )
proof ((standard; intro ballI impI), goal-cases)
  case (1 y)
  define y2 where y2 = vec nr ( $\lambda i. \text{if } y \ \$ \ i \geq 0 \ \text{then } y \ \$ \ i \ \text{else } 0$ )
  define y3 where y3 = vec nr ( $\lambda i. \text{if } y \ \$ \ i \geq 0 \ \text{then } 0 \ \text{else } -y \ \$ \ i$ )
  have y: y = y2 - y3 unfolding y2-def y3-def using 1(2)
  by (intro eq-vecI, auto)
  show ?case by (rule 1(1)[rule-format, of y2 y3, folded y, OF - - 1(3)],
    auto simp: y2-def y3-def less-eq-vec-def)
qed auto
also have ... = ( $\forall y \in \text{carrier-vec nr. mat-of-row } y * A \geq 0_m \ 1 \ nc \longrightarrow y \cdot b \geq 0$ )
  by (intro ball-cong arg-cong2[of - - - - ( $\longrightarrow$ )] refl,
    insert A, auto simp: less-eq-vec-def less-eq-mat-def)
finally show ?thesis .
qed
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* \Rightarrow *real* \Rightarrow *real* **where**

[*simp*]: $x *R y = \text{real-of-rat } x * y$

instance **by** *standard* (*auto simp add: field-simps of-rat-mult of-rat-add*)

end

abbreviation *real-satisfies-constraints* :: *real valuation* \Rightarrow *constraint set* \Rightarrow *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* \Rightarrow *real valuation* **where**

of-rat-val $v \ x = \text{of-rat } (v \ x)$

lemma *of-rat-val-eval*: $p \{ \text{of-rat-val } v \} = \text{of-rat } (p \{ v \})$
unfolding *of-rat-val-def linear-poly-sum of-rat-sum*
by (*rule sum.cong, auto simp: of-rat-mult*)

lemma *of-rat-val-constraint*: $\text{of-rat-val } v \models_c c \longleftrightarrow v \models_c c$
by (*cases c, auto simp: of-rat-val-eval of-rat-less of-rat-less-eq*)

lemma *of-rat-val-constraints*: $\text{of-rat-val } v \models_{rcs} cs \longleftrightarrow v \models_{cs} cs$
using *of-rat-val-constraint* **by** *auto*

lemma *sat-scale-rat-real*: **assumes** ($v :: \text{real valuation}$) $\models_c c$
shows $v \models_c (r * R c)$

proof –

have $r < 0 \vee r = 0 \vee r > 0$ **by** *auto*

then show *?thesis* **using** *assms* **by** (*cases c, simp-all add: right-diff-distrib*
valuate-minus valuate-scaleRat scaleRat-leq1 scaleRat-leq2 valuate-zero
of-rat-less of-rat-mult)

qed

fun *of-rat-lec* :: $\text{rat le-constraint} \Rightarrow \text{real le-constraint}$ **where**
of-rat-lec (*Le-Constraint* $r p c$) = *Le-Constraint* $r p (\text{of-rat } c)$

lemma *lec-of-constraint-real*:

assumes *is-le c*

shows ($v \models_{le} \text{of-rat-lec } (\text{lec-of-constraint } c)$) $\longleftrightarrow (v \models_c c)$

using *assms* **by** (*cases c, auto*)

lemma *of-rat-lec-add*: $\text{of-rat-lec } (c + d) = \text{of-rat-lec } c + \text{of-rat-lec } d$
by (*cases c; cases d, auto simp: of-rat-add*)

lemma *of-rat-lec-zero*: $\text{of-rat-lec } 0 = 0$
unfolding *zero-le-constraint-def* **by** *simp*

lemma *of-rat-lec-sum*: $\text{of-rat-lec } (\text{sum-list } c) = \text{sum-list } (\text{map of-rat-lec } c)$
by (*induct c, auto simp: of-rat-lec-zero of-rat-lec-add*)

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

lemma *rat-real-conversion*: **assumes** *finite cs*

shows ($\exists v :: \text{rat valuation. } v \models_{cs} cs$) $\longleftrightarrow (\exists v :: \text{real valuation. } v \models_{rcs} cs)$

proof

show $\exists v. v \models_{cs} cs \implies \exists v. v \models_{rcs} cs$ **using** *of-rat-val-constraint* **by** *auto*

assume $\exists v. v \models_{rcs} cs$

then obtain v **where** $*$: $v \models_{rcs} cs$ **by** *auto*

show $\exists v. v \models_{cs} cs$

proof (*rule ccontr*)

assume $\nexists v. v \models_{cs} cs$

from *farkas-coefficients[OF assms]* **this**

obtain C **where** *farkas-coefficients cs C* **by** *auto*

```

from this[unfolding farkas-coefficients-def]
obtain  $d$  rel where
  isleg:  $(\forall (r,c) \in \text{set } C. c \in \text{cs} \wedge \text{is-le } (r *R c) \wedge r \neq 0)$  and
  leq:  $(\sum (r,c) \leftarrow C. \text{lec-of-constraint } (r *R c)) = \text{Le-Constraint } \text{rel } 0 \ d$  and
  choice:  $\text{rel} = \text{Lt-Rel} \wedge d \leq 0 \vee \text{rel} = \text{Leq-Rel} \wedge d < 0$  by blast
{
  fix  $r \ c$ 
  assume  $c: (r,c) \in \text{set } C$ 
  from  $c * \text{isleg}$  have  $v \models_c c$  by auto
  hence  $v: v \models_c (r *R c)$  by (rule sat-scale-rat-real)
  from  $c \text{ isleg}$  have  $\text{is-le } (r *R c)$  by auto
  from  $\text{lec-of-constraint-real}[OF \text{ this}] \ v$ 
  have  $v \models_{le} \text{of-rat-lec } (\text{lec-of-constraint } (r *R c))$  by blast
} note  $v = \text{this}$ 
have  $\text{Le-Constraint } \text{rel } 0 \ (\text{of-rat } d) = \text{of-rat-lec } (\sum (r,c) \leftarrow C. \text{lec-of-constraint } (r *R c))$ 
  unfolding leq by simp
  also have  $\dots = (\sum (r,c) \leftarrow C. \text{of-rat-lec } (\text{lec-of-constraint } (r *R c)))$  (is - = ?sum)
  unfolding of-rat-lec-sum map-map o-def by (rule arg-cong[of - - sum-list], auto)
  finally have leq:  $\text{Le-Constraint } \text{rel } 0 \ (\text{of-rat } d) = ?\text{sum}$  by simp
  have  $v \models_{le} \text{Le-Constraint } \text{rel } 0 \ (\text{of-rat } d)$  unfolding leq
    by (rule satisfies-sumlist-le-constraints, insert v, auto)
  with choice show False by (auto simp: linear-poly-sum)
qed
qed

```

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

```

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

```

lemma simplex-index-real:

```

  simplex-index cs = Unsat I  $\implies$  set I  $\subseteq$  fst ‘ set cs  $\wedge$   $\neg (\exists v. (\text{set } I, v) \models_{rics} \text{set } cs)$   $\wedge$ 
    (distinct-indices cs  $\longrightarrow (\forall J \subset \text{set } I. (\exists v. (J, v) \models_{ics} \text{set } cs))$ ) — minimal
  unsat core over the reals
  simplex-index cs = Sat v  $\implies \langle v \rangle \models_{cs} (\text{snd } ‘ \text{set } cs)$  — satisfying assingment
  using simplex-index(1)[of cs I] simplex-index(2)[of cs v]
    rat-real-conversion[of Simplex.restrict-to (set I) (set cs)]
  by auto

```

lemma simplex-real:

```

  simplex cs = Unsat I  $\implies \neg (\exists v. v \models_{rcs} \text{set } cs)$  — unsat of original constraints
  over the reals
  simplex cs = Unsat I  $\implies \text{set } I \subseteq \{0..<\text{length } \text{cs}\} \wedge \neg (\exists v. v \models_{rcs} \{\text{cs } ! \ i \mid i. i \in \text{set } I\})$ 
   $\wedge (\forall J \subset \text{set } I. \exists v. v \models_{cs} \{\text{cs } ! \ i \mid i. i \in J\})$  — minimal unsat core over reals

```

$\text{simplex } cs = \text{Sat } v \implies \langle v \rangle \models_{cs} \text{set } cs$ — satisfying assignment over the rationals
proof (*intro simplex(1)[unfolded rat-real-conversion[OF finite-set]]*)
assume *unsat: simplex cs = Inl I*
have *finite {cs ! i | i. i ∈ set I}* **by** *auto*
from *simplex(2)[OF unsat, unfolded rat-real-conversion[OF this]]*
show $\text{set } I \subseteq \{0..<\text{length } cs\} \wedge \neg (\exists v. v \models_{rcs} \{cs ! i \mid i. i \in \text{set } I\})$
 $\wedge (\forall J \subseteq \text{set } I. \exists v. v \models_{cs} \{cs ! i \mid i. i \in J\})$ **by** *auto*
qed (*insert simplex(3), auto*)

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

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

lemma *minimal-unsat-core-real-conv*: $\text{minimal-unsat-core-real } I \text{ ics} = \text{minimal-unsat-core } I \text{ ics}$

proof
show $\text{minimal-unsat-core-real } I \text{ ics} \implies \text{minimal-unsat-core } I \text{ ics}$
unfolding *minimal-unsat-core-real-def minimal-unsat-core-def*
using *of-rat-val-constraint* **by** *simp metis*
next
assume *minimal-unsat-core I ics*
thus *minimal-unsat-core-real I ics*
unfolding *minimal-unsat-core-real-def minimal-unsat-core-def*
using *rat-real-conversion[of Simplex.restrict-to I (set ics)]*
by *auto*
qed

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.