

Linear-Programming

Julian Parsert

September 13, 2023

Abstract

We use the previous formalization of the general simplex algorithm to formulate an algorithm for solving linear programs. We encode the linear programs using only linear constraints. Solving these constraints also solves the original linear program. This algorithm is proven to be sound by applying the weak duality theorem which is also part of this formalization [5].

Contents

1	Related work	1
2	General Theorems used later, that could be moved	2
3	Vectors	3
4	Translations of Jordan Normal Forms Matrix Library to Simplex polynomials	7
4.1	Vectors	7
5	Matrices	10
6	Get different matrices into same space, without interference	13
7	Translate Inequalities to Matrix Form	20
8	Abstract LPs	22

1 Related work

Our work is based on a formalization of the general simplex algorithm described in [3, 6]. However, the general simplex algorithm lacks the ability to optimize a function. Boulmé and Maréchal [2] describe a formalization and implementation of Coq tactics for linear integer programming and linear

arithmetic over rationals. More closely related is the formalization by Al-
lamigeon et al. [1] which formalizes the simplex method and related results.
As part of Flyspeck project Obua and Nipkow [4] created a verification mech-
anism for linear programs using the HOL computing library and external
solvers.

theory *More-Jordan-Normal-Forms*

imports

Jordan-Normal-Form.Matrix-Impl

begin

lemma *set-comprehension-list-comprehension:*

set [f i . i <- [x..<a]] = {f i |i. i ∈ {x..<a}}

<proof>

lemma *in-second-append-list: i ≥ length a ⇒ i < length (a@b) ⇒ (a@b)!i ∈ set
b*

<proof>

2 General Theorems used later, that could be moved

lemma *split-four-block-dual-fst-lst:*

assumes *split-block (four-block-mat A B C D) (dim-row A) (dim-col A) = (U,
X, Y, V)*

shows *U = A V = D*

<proof>

lemma *append-split-vec-distrib-scalar-prod:*

assumes *dim-vec (u @_v w) = dim-vec x*

shows *(u @_v w) · x = u · (vec-first x (dim-vec u)) + w · (vec-last x (dim-vec
w))*

<proof>

lemma *append-dot-product-split:*

assumes *dim-vec (u @_v w) = dim-vec x*

shows *(u @_v w) · x = (∑ i∈{0..< dim-vec u}. u\$i * x\$i) + (∑ i∈{0..< dim-vec
w}. w\$i * x\$(i + dim-vec u))*

<proof>

lemma *assoc-scalar-prod-mult-mat-vec:*

fixes *A :: 'a::comm-semiring-1 mat*

assumes *y ∈ carrier-vec n*

assumes *x ∈ carrier-vec m*

assumes *A ∈ carrier-mat n m*

shows *(A *_v x) · y = (A^T *_v y) · x*

<proof>

3 Vectors

abbreviation $\text{singleton } V \ ([-]_v)$ **where** $\text{singleton } V \ e \equiv (\text{vec } 1 \ (\lambda i. e))$

lemma *elem-in-singleton* [simp]: $[a]_v \ \$ \ 0 = a$
 ⟨proof⟩

lemma *elem-in-singleton-append* [simp]: $(x \ @_v \ [a]_v) \ \$ \ \text{dim-vec } x = a$
 ⟨proof⟩

lemma *vector-cases-append*:
fixes $x :: 'a \ \text{vec}$
shows $x = vNil \ \vee \ (\exists v \ a. \ x = v \ @_v \ [a]_v)$
 ⟨proof⟩

lemma *vec-vec-induct* [case-names *vNil append*, induct type: *vec*]:
assumes $P \ vNil$ **and** $\bigwedge a \ v. \ P \ v \ \Longrightarrow \ P \ (v \ @_v \ [a]_v)$
shows $P \ v$
 ⟨proof⟩

lemma *singleton-append-dotP*:
assumes $\text{dim-vec } z = \text{dim-vec } y + 1$
shows $(y \ @_v \ [x]_v) \cdot z = (\sum_{i \in \{0..<\text{dim-vec } y\}} y \ \$ \ i * z \ \$ \ i) + x * z \ \$ \ \text{dim-vec } y$
 ⟨proof⟩

lemma *map-vec-append*: $\text{map-vec } f \ (a \ @_v \ b) = \text{map-vec } f \ a \ @_v \ \text{map-vec } f \ b$
 ⟨proof⟩

lemma *map-mat-map-vec*:
assumes $i < \text{dim-row } P$
shows $\text{row } (\text{map-mat } f \ P) \ i = \text{map-vec } f \ (\text{row } P \ i)$
 ⟨proof⟩

lemma *append-rows-access1* [simp]:
assumes $i < \text{dim-row } A$
assumes $\text{dim-col } A = \text{dim-col } B$
shows $\text{row } (A \ @_r \ B) \ i = \text{row } A \ i$
 ⟨proof⟩

lemma *append-rows-access2* [simp]:
assumes $i \geq \text{dim-row } A$
assumes $i < \text{dim-row } A + \text{dim-row } B$
assumes $\text{dim-col } A = \text{dim-col } B$
shows $\text{row } (A \ @_r \ B) \ i = \text{row } B \ (i - \text{dim-row } A)$
 ⟨proof⟩

lemma *append-singleton-access* [simp]: $(\text{Matrix.vec } n \ f \ @_v \ [r]_v) \ \$ \ n = r$
 ⟨proof⟩

Move to right place

fun *mat-append-col* **where**

mat-append-col A $b = \text{mat-of-cols } (\text{dim-row } A) (\text{cols } A @ [b])$

fun *mat-append-row* **where**

mat-append-row A $c = \text{mat-of-rows } (\text{dim-col } A) (\text{rows } A @ [c])$

lemma *mat-append-col-dims*:

shows *mat-append-col* A $b \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-col } A + 1)$

<proof>

lemma *mat-append-row-dims*:

shows *mat-append-row* A $c \in \text{carrier-mat } (\text{dim-row } A + 1) (\text{dim-col } A)$

<proof>

lemma *mat-append-col-col*:

assumes $\text{dim-row } A = \text{dim-vec } b$

shows $\text{col } (\text{mat-append-col } A \ b) (\text{dim-col } A) = b$

<proof>

lemma *mat-append-col-vec-index*:

assumes $i < \text{dim-row } A$

and $\text{dim-row } A = \text{dim-vec } b$

shows $(\text{row } (\text{mat-append-col } A \ b) \ i) \$ (\text{dim-col } A) = b \$ i$

<proof>

lemma *mat-append-row-row*:

assumes $\text{dim-col } A = \text{dim-vec } c$

shows $\text{row } (\text{mat-append-row } A \ c) (\text{dim-row } A) = c$

<proof>

lemma *mat-append-row-in-mat*:

assumes $i < \text{dim-row } A$

shows $\text{row } (\text{mat-append-row } A \ r) \ i = \text{row } A \ i$

<proof>

lemma *mat-append-row-vec-index*:

assumes $i < \text{dim-col } A$

and $\text{dim-col } A = \text{dim-vec } b$

shows $\text{vec-index } (\text{col } (\text{mat-append-row } A \ b) \ i) (\text{dim-row } A) = \text{vec-index } b \ i$

<proof>

lemma *mat-append-col-access-in-mat*:

assumes $\text{dim-row } A = \text{dim-vec } b$

and $i < \text{dim-row } A$

and $j < \text{dim-col } A$

shows $(\text{row } (\text{mat-append-col } A \ b) \ i) \$ j = (\text{row } A \ i) \$ j$

<proof>

lemma *constructing-append-col-row*:

assumes $i < \text{dim-row } A$

and $\text{dim-row } A = \text{dim-vec } b$

shows $\text{row } (\text{mat-append-col } A \ b) \ i = \text{row } A \ i \ @_v \ [\text{vec-index } b \ i]_v$

<proof>

definition *one-element-vec* **where** $\text{one-element-vec } n \ e = \text{vec } n \ (\lambda i. \ e)$

lemma *one-element-vec-carrier*: $\text{one-element-vec } n \ e \in \text{carrier-vec } n$

<proof>

lemma *one-element-vec-dim* [*simp*]: $\text{dim-vec } (\text{one-element-vec } n \ (r::\text{rat})) = n$

<proof>

lemma *one-element-vec-access* [*simp*]: $\bigwedge i. \ i < n \implies \text{vec-index } (\text{one-element-vec } n \ e) \ i = e$

<proof>

fun *single-nz-val* **where** $\text{single-nz-val } n \ i \ v = \text{vec } n \ (\lambda j. \ (\text{if } i = j \ \text{then } v \ \text{else } 0))$

lemma *single-nz-val-carrier*: $\text{single-nz-val } n \ i \ v \in \text{carrier-vec } n$

<proof>

lemma *single-nz-val-access1* [*simp*]: $i < n \implies \text{single-nz-val } n \ i \ v \ \$ \ i = v$

<proof>

lemma *single-nz-val-access2* [*simp*]: $i < n \implies j < n \implies i \neq j \implies \text{single-nz-val } n \ i \ v \ \$ \ j = 0$

<proof>

lemma $i < n \implies (v \cdot_v \ \text{unit-vec } n \ i) \ \$ \ i = (v::'a::\{\text{monoid-mult}, \text{times}, \text{zero-neq-one}\})$

<proof>

lemma *single-nz-val-unit-vec*:

fixes $v::'a::\{\text{monoid-mult}, \text{times}, \text{zero-neq-one}, \text{mult-zero}\}$

shows $v \cdot_v \ (\text{unit-vec } n \ i) = \text{single-nz-val } n \ i \ v$

<proof>

lemma *single-nz-valI* [*intro*]:

fixes $v \ i \ \text{val}$

assumes $\bigwedge j. \ j < \text{dim-vec } v \implies j \neq i \implies v \ \$ \ j = 0$

assumes $v \ \$ \ i = \text{val}$

shows $v = \text{single-nz-val } (\text{dim-vec } v) \ i \ \text{val}$

<proof>

lemma *single-nz-val-dotP*:

assumes $i < n$
assumes $\dim\text{-vec } x = n$
shows $\text{single-nz-val } n \ i \ v \cdot x = v * x \ \$ \ i$
 $\langle \text{proof} \rangle$

lemma *single-nz-zero-singleton*: $\text{single-nz-val } 1 \ 0 \ v = [v]_v$
 $\langle \text{proof} \rangle$

lemma *append-one-elem-zero-dotP*:

assumes $\dim\text{-vec } u = m$
and $\dim\text{-vec } x = n$
shows $(\text{one-element-vec } n \ e \ @_v \ (0_v \ m)) \cdot (x \ @_v \ u) = (\sum_{i \in \{0 \ .. < \dim\text{-vec } x\}} e * x \ \$ \ i)$
 $\langle \text{proof} \rangle$

lemma *one-element-vec-dotP*:

assumes $\dim\text{-vec } x = n$
shows $(\text{one-element-vec } n \ e) \cdot x = (\sum_{i \in \{0 \ .. < \dim\text{-vec } x\}} e * x \ \$ \ i)$
 $\langle \text{proof} \rangle$

lemma *singleton-dotP* [*simp*]: $\dim\text{-vec } x = 1 \implies [v]_v \cdot x = v * x \$ 0$
 $\langle \text{proof} \rangle$

lemma *singletons-dotP* [*simp*]: $[v]_v \cdot [w]_v = v * w$
 $\langle \text{proof} \rangle$

lemma *singleton-appends-dotP* [*simp*]: $\dim\text{-vec } x = \dim\text{-vec } y \implies (x \ @_v \ [v]_v) \cdot (y \ @_v \ [w]_v) = x \cdot y + v * w$
 $\langle \text{proof} \rangle$

end

theory *Matrix-LinPoly*

imports

Jordan-Normal-Form.Matrix-Impl

Farkas.Simplex-for-Reals

Farkas.Matrix-Farkas

begin

Add this to linear polynomials in Simplex

lemma *eval-poly-with-sum*: $(v \ \{\!\! \{ X \}\!\! \}) = (\sum_{x \in \text{vars } v} \text{coeff } v \ x * X \ x)$
 $\langle \text{proof} \rangle$

lemma *eval-poly-with-sum-superset*:

assumes *finite* S
assumes $S \supseteq \text{vars } v$
shows $(v \ \{\!\! \{ X \}\!\! \}) = (\sum_{x \in S} \text{coeff } v \ x * X \ x)$
 $\langle \text{proof} \rangle$

Get rid of these synonyms

4 Translations of Jordan Normal Forms Matrix Library to Simplex polynomials

4.1 Vectors

definition *list-to-lpoly* **where**

$list-to-lpoly\ cs = sum-list\ (map2\ (\lambda\ i\ c.\ lp-monom\ c\ i)\ [0..<length\ cs]\ cs)$

lemma *empty-list-0poly*:

shows $list-to-lpoly\ [] = 0$

$\langle proof \rangle$

lemma *sum-list-map-upto-coeff-limit*:

assumes $i \geq length\ L$

shows $coeff\ (list-to-lpoly\ L)\ i = 0$

$\langle proof \rangle$

lemma *rl-lpoly-coeff-nth-non-empty*:

assumes $i < length\ cs$

assumes $cs \neq []$

shows $coeff\ (list-to-lpoly\ cs)\ i = cs!i$

$\langle proof \rangle$

lemma *list-to-lpoly-coeff-nth*:

assumes $i < length\ cs$

shows $coeff\ (list-to-lpoly\ cs)\ i = cs\ !\ i$

$\langle proof \rangle$

lemma *rat-list-outside-zero*:

assumes $length\ cs \leq i$

shows $coeff\ (list-to-lpoly\ cs)\ i = 0$

$\langle proof \rangle$

Transform linear polynomials to rational vectors

fun *dim-poly* **where**

$dim-poly\ p = (if\ (vars\ p) = \{\}\ then\ 0\ else\ Max\ (vars\ p)+1)$

definition *max-dim-poly-list* **where**

$max-dim-poly-list\ lst = Max\ \{Max\ (vars\ p)\ |\ p.\ p \in\ set\ lst\}$

fun *lpoly-to-vec* **where**

$lpoly-to-vec\ p = vec\ (dim-poly\ p)\ (coeff\ p)$

lemma *all-greater-dim-poly-zero[simp]*:

assumes $x \geq \text{dim-poly } p$
shows $\text{coeff } p \ x = 0$
 ⟨proof⟩

lemma *lpoly-to-vec-0-iff-zero-poly* [iff]:
shows $(\text{lpoly-to-vec } p) = 0_v \ 0 \longleftrightarrow p = 0$
 ⟨proof⟩

lemma *dim-poly-dim-vec-equiv*:
 $\text{dim-vec } (\text{lpoly-to-vec } p) = \text{dim-poly } p$
 ⟨proof⟩

lemma *dim-poly-greater-ex-coeff*: $\text{dim-poly } x > d \implies \exists i \geq d. \text{coeff } x \ i \neq 0$
 ⟨proof⟩

lemma *dimpoly-all-zero-limit*:
assumes $\bigwedge i. i \geq d \implies \text{coeff } x \ i = 0$
shows $\text{dim-poly } x \leq d$
 ⟨proof⟩

lemma *construct-poly-from-lower-dim-poly*:
assumes $\text{dim-poly } x = d+1$
obtains $p \ c$ **where** $\text{dim-poly } p \leq d \ x = p + \text{lp-monom } c \ d$
 ⟨proof⟩

lemma *vars-subset-0-dim-poly*:
 $\text{vars } z \subseteq \{0..<\text{dim-poly } z\}$
 ⟨proof⟩

lemma *in-dim-and-not-var-zero*: $x \in \{0..<\text{dim-poly } z\} - \text{vars } z \implies \text{coeff } z \ x = 0$
 ⟨proof⟩

lemma *valuate-with-dim-poly*: $z \ \{ X \} = (\sum i \in \{0..<\text{dim-poly } z\}. \text{coeff } z \ i * X \ i)$
 ⟨proof⟩

lemma *lin-poly-to-vec-coeff-access*:
assumes $x < \text{dim-poly } y$
shows $(\text{lpoly-to-vec } y) \ \$ \ x = \text{coeff } y \ x$
 ⟨proof⟩

lemma *addition-over-lin-poly-to-vec*:
fixes $x \ y$
assumes $a < \text{dim-poly } x$
assumes $\text{dim-poly } x = \text{dim-poly } y$
shows $(\text{lpoly-to-vec } x + \text{lpoly-to-vec } y) \ \$ \ a = \text{coeff } (x + y) \ a$
 ⟨proof⟩

lemma *list-to-lpoly-dim-less*: $\text{length } cs \geq \text{dim-poly } (\text{list-to-lpoly } cs)$
 ⟨proof⟩

Transform rational vectors to linear polynomials

fun *vec-to-lpoly* **where**

vec-to-lpoly *rv* = *list-to-lpoly* (*list-of-vec* *rv*)

lemma *vec-to-lin-poly-coeff-access*:

assumes $x < \text{dim-vec } y$

shows $y \$ x = \text{coeff } (\text{vec-to-lpoly } y) x$

<proof>

lemma *addition-over-vec-to-lin-poly*:

fixes $x y$

assumes $a < \text{dim-vec } x$

assumes $\text{dim-vec } x = \text{dim-vec } y$

shows $(x + y) \$ a = \text{coeff } (\text{vec-to-lpoly } x + \text{vec-to-lpoly } y) a$

<proof>

lemma *outside-list-coeff0*:

assumes $i \geq \text{dim-vec } xs$

shows $\text{coeff } (\text{vec-to-lpoly } xs) i = 0$

<proof>

lemma *vec-to-poly-dim-less*:

$\text{dim-poly } (\text{vec-to-lpoly } x) \leq \text{dim-vec } x$

<proof>

lemma *vec-to-lpoly-from-lpoly-coeff-dual1*:

$\text{coeff } (\text{vec-to-lpoly } (\text{lpoly-to-vec } p)) i = \text{coeff } p i$

<proof>

lemma *vec-to-lpoly-from-lpoly-coeff-dual2*:

assumes $i < \text{dim-vec } (\text{lpoly-to-vec } (\text{vec-to-lpoly } v))$

shows $(\text{lpoly-to-vec } (\text{vec-to-lpoly } v)) \$ i = v \$ i$

<proof>

lemma *vars-subset-dim-vec-to-lpoly-dim*: $\text{vars } (\text{vec-to-lpoly } v) \subseteq \{0..<\text{dim-vec } v\}$

<proof>

lemma *sum-dim-vec-equals-sum-dim-poly*:

shows $(\sum a = 0..<\text{dim-vec } A. \text{coeff } (\text{vec-to-lpoly } A) a * X a) =$

$(\sum a = 0..<\text{dim-poly } (\text{vec-to-lpoly } A). \text{coeff } (\text{vec-to-lpoly } A) a * X a)$

<proof>

lemma *vec-to-lpoly-vNil* [*simp*]: $\text{vec-to-lpoly } vNil = 0$

<proof>

lemma *zero-vector-is-zero-poly*: $\text{coeff } (\text{vec-to-lpoly } (0_v \ n)) i = 0$

<proof>

lemma *coeff-nonzero-dim-vec-non-zero*:

assumes $\text{coeff } (\text{vec-to-lpoly } v) \ i \neq 0$
shows $v \ \$ \ i \neq 0 \ i < \text{dim-vec } v$
 $\langle \text{proof} \rangle$

lemma *lpoly-of-v-equals-v-append0*:
 $\text{vec-to-lpoly } v = \text{vec-to-lpoly } (v \ @_v \ 0_v \ a)$ (**is** $?lhs = ?rhs$)
 $\langle \text{proof} \rangle$

lemma *vec-to-lpoly-eval-dot-prod*:
 $(\text{vec-to-lpoly } v) \ \{ \ x \ \} = v \cdot (\text{vec } (\text{dim-vec } v) \ x)$
 $\langle \text{proof} \rangle$

lemma *dim-poly-of-append-vec*:
 $\text{dim-poly } (\text{vec-to-lpoly } (a \ @_v \ b)) \leq \text{dim-vec } a + \text{dim-vec } b$
 $\langle \text{proof} \rangle$

lemma *vec-coeff-append1*: $i \in \{0..<\text{dim-vec } a\} \implies \text{coeff } (\text{vec-to-lpoly } (a \ @_v \ b)) \ i = a \$ i$
 $\langle \text{proof} \rangle$

lemma *vec-coeff-append2*:
 $i \in \{\text{dim-vec } a..<\text{dim-vec } (a \ @_v \ b)\} \implies \text{coeff } (\text{vec-to-lpoly } (a \ @_v \ b)) \ i = b \$ (i - \text{dim-vec } a)$
 $\langle \text{proof} \rangle$

Maybe Code Equation

lemma *vec-to-lpoly-poly-of-vec-eq*: $\text{vec-to-lpoly } v = \text{poly-of-vec } v$
 $\langle \text{proof} \rangle$

lemma *vars-vec-append-subset*: $\text{vars } (\text{vec-to-lpoly } (0_v \ n \ @_v \ v)) \subseteq \{n..<n + \text{dim-vec } v\}$
 $\langle \text{proof} \rangle$

5 Matrices

fun *matrix-to-lpolies* **where**
 $\text{matrix-to-lpolies } A = \text{map } \text{vec-to-lpoly } (\text{rows } A)$

lemma *matrix-to-lpolies-vec-of-row*:
 $i < \text{dim-row } A \implies \text{matrix-to-lpolies } A \ ! \ i = \text{vec-to-lpoly } (\text{row } A \ i)$
 $\langle \text{proof} \rangle$

lemma *outside-of-col-range-is-0*:
assumes $i < \text{dim-row } A$ **and** $j \geq \text{dim-col } A$
shows $\text{coeff } ((\text{matrix-to-lpolies } A) \ ! \ i) \ j = 0$
 $\langle \text{proof} \rangle$

lemma *polys-greater-col-zero*:
assumes $x \in \text{set } (\text{matrix-to-lpolies } A)$

assumes $j \geq \dim\text{-col } A$
shows $\text{coeff } x \ j = 0$
 $\langle \text{proof} \rangle$

lemma *matrix-to-lp-vec-to-lpoly-row* [simp]:
assumes $i < \dim\text{-row } A$
shows $(\text{matrix-to-lpolies } A)!i = \text{vec-to-lpoly } (\text{row } A \ i)$
 $\langle \text{proof} \rangle$

lemma *matrix-to-lpolies-coeff-access*:
assumes $i < \dim\text{-row } A$ **and** $j < \dim\text{-col } A$
shows $\text{coeff } (\text{matrix-to-lpolies } A \ ! \ i) \ j = A \ \$\$ \ (i,j)$
 $\langle \text{proof} \rangle$

From linear polynomial list to matrix

definition *lin-polies-to-mat* **where**
 $\text{lin-polies-to-mat } lst = \text{mat } (\text{length } lst) \ (\text{max-dim-poly-list } lst) \ (\lambda(x,y).\text{coeff } (lst!x) \ y)$

lemma *lin-polies-to-rat-mat-coeff-index*:
assumes $i < \text{length } L$ **and** $j < (\text{max-dim-poly-list } L)$
shows $\text{coeff } (L \ ! \ i) \ j = (\text{lin-polies-to-mat } L) \ \$\$ \ (i,j)$
 $\langle \text{proof} \rangle$

lemma *vec-to-lpoly-valuate-equiv-dot-prod*:
assumes $\dim\text{-vec } y = \dim\text{-vec } x$
shows $(\text{vec-to-lpoly } y) \ \{\! \{ (\$)x \} \! \} = y \cdot x$
 $\langle \text{proof} \rangle$

lemma *matrix-to-lpolies-valuate-scalarP*:
assumes $i < \dim\text{-row } A$
assumes $\dim\text{-col } A = \dim\text{-vec } x$
shows $(\text{matrix-to-lpolies } A)!i \ \{\! \{ (\$)x \} \! \} = (\text{row } A \ i) \cdot x$
 $\langle \text{proof} \rangle$

lemma *matrix-to-lpolies-lambda-valuate-scalarP*:
assumes $i < \dim\text{-row } A$
assumes $\dim\text{-col } A = \dim\text{-vec } x$
shows $(\text{matrix-to-lpolies } A)!i \ \{\! \{ (\lambda i. (\text{if } i < \dim\text{-vec } x \ \text{then } x\$i \ \text{else } 0)) \} \! \} = (\text{row } A \ i) \cdot x$
 $\langle \text{proof} \rangle$

end
theory *LP-Preliminaries*
imports
More-Jordan-Normal-Forms

Matrix-LinPoly
Jordan-Normal-Form.Matrix-Impl
Farkas.Simplex-for-Reals
HOL-Library.Mapping

begin

fun *vars-from-index-geq-vec* **where**
vars-from-index-geq-vec *index* *b* = [GEQ (*lp-monom* 1 (*i+index*)) (*b*\$*i*). *i* ← [0..*dim-vec* *b*]]

lemma *constraints-set-vars-geq-vec-def*:
set (*vars-from-index-geq-vec* *start* *b*) =
{GEQ (*lp-monom* 1 (*i+start*)) (*b*\$*i*) | *i*. *i* ∈ {0..*dim-vec* *b*}}
⟨*proof*⟩

lemma *vars-from-index-geq-sat*:
assumes ⟨*x*⟩ ⊨_{cs} *set* (*vars-from-index-geq-vec* *start* *b*)
assumes *i* < *dim-vec* *b*
shows ⟨*x*⟩ (*i+start*) ≥ *b*\$*i*
⟨*proof*⟩

fun *mat-x-leq-vec* **where**
mat-x-leq-vec *A* *b* = [LEQ (*matrix-to-lpolies* *A*!*i*) (*b*\$*i*) . *i* <- [0..*dim-vec* *b*]]

lemma *mat-x-leq-vec-sol*:
assumes ⟨*x*⟩ ⊨_{cs} *set* (*mat-x-leq-vec* *A* *b*)
assumes *i* < *dim-vec* *b*
shows ((*matrix-to-lpolies* *A*)!*i*) ⋈⟨*x*⟩ ≤ *b*\$*i*
⟨*proof*⟩

fun *x-mat-eq-vec* **where**
x-mat-eq-vec *b* *A* = [EQ (*matrix-to-lpolies* *A*!*i*) (*b*\$*i*) . *i* <- [0..*dim-vec* *b*]]

lemma *x-mat-eq-vec-sol*:
assumes *x* ⊨_{cs} *set* (*x-mat-eq-vec* *b* *A*)
assumes *i* < *dim-vec* *b*
shows ((*matrix-to-lpolies* *A*)!*i*) ⋈ *x* ⋈ = *b*\$*i*
⟨*proof*⟩

6 Get different matrices into same space, without interference

fun *two-block-non-interfering* **where**

two-block-non-interfering $A\ B = (\text{let } z1 = 0_m (\text{dim-row } A) (\text{dim-col } B);$
 $z2 = 0_m (\text{dim-row } B) (\text{dim-col } A) \text{ in}$
four-block-mat $A\ z1\ z2\ B)$

lemma *split-two-block-non-interfering*:

assumes *split-block* (*two-block-non-interfering* $A\ B$) ($\text{dim-row } A$) ($\text{dim-col } A$) =
 $(Q1, Q2, Q3, Q4)$
shows $Q1 = A\ Q4 = B$
 $\langle \text{proof} \rangle$

lemma *two-block-non-interfering-dims*:

$\text{dim-row } (\text{two-block-non-interfering } A\ B) = \text{dim-row } A + \text{dim-row } B$
 $\text{dim-col } (\text{two-block-non-interfering } A\ B) = \text{dim-col } A + \text{dim-col } B$
 $\langle \text{proof} \rangle$

lemma *two-block-non-interfering-zeros-are-0*:

assumes $i < \text{dim-row } A$
and $j \geq \text{dim-col } A$
and $j < \text{dim-col } (\text{two-block-non-interfering } A\ B)$
shows $(\text{two-block-non-interfering } A\ B)\$(i,j) = 0$ (*two-block-non-interfering* $A\ B)\$(i,j) = 0$
 $\langle \text{proof} \rangle$

lemma *two-block-non-interfering-row-comp1*:

assumes $i < \text{dim-row } A$
shows $\text{row } (\text{two-block-non-interfering } A\ B)\ i = \text{row } A\ i @_v (0_v (\text{dim-col } B))$
 $\langle \text{proof} \rangle$

lemma *two-block-non-interfering-row-comp2*:

assumes $i < \text{dim-row } (\text{two-block-non-interfering } A\ B)$
and $i \geq \text{dim-row } A$
shows $\text{row } (\text{two-block-non-interfering } A\ B)\ i = (0_v (\text{dim-col } A)) @_v \text{row } B\ (i - \text{dim-row } A)$
 $\langle \text{proof} \rangle$

lemma *first-vec-two-block-non-inter-is-first-vec*:

assumes $\text{dim-col } A + \text{dim-col } B = \text{dim-vec } v$
assumes $\text{dim-row } A = n$
shows $\text{vec-first } (\text{two-block-non-interfering } A\ B *_v v)\ n = A *_v (\text{vec-first } v (\text{dim-col } A))$
 $\langle \text{proof} \rangle$

lemma *last-vec-two-block-non-inter-is-last-vec*:

assumes $\text{dim-col } A + \text{dim-col } B = \text{dim-vec } v$
assumes $\text{dim-row } B = n$

shows $\text{vec-last } ((\text{two-block-non-interfering } A \ B) *_{\mathit{v}} \ \mathit{v}) \ n = B *_{\mathit{v}} (\text{vec-last } \ \mathit{v} \ (\text{dim-col } B))$
 ⟨proof⟩

lemma *two-block-non-interfering-mult-decomposition:*

assumes $\text{dim-col } A + \text{dim-col } B = \text{dim-vec } \mathit{v}$

shows $\text{two-block-non-interfering } A \ B *_{\mathit{v}} \ \mathit{v} =$

$A *_{\mathit{v}} \text{vec-first } \mathit{v} \ (\text{dim-col } A) \ @_{\mathit{v}} \ B *_{\mathit{v}} \text{vec-last } \mathit{v} \ (\text{dim-col } B)$

⟨proof⟩

fun *mat-leqb-egc where*

$\text{mat-leqb-egc } A \ b \ c = (\text{let } \text{lst} = \text{matrix-to-lpolies } (\text{two-block-non-interfering } A \ A^T) \ \text{in}$

$\text{[LEQ } (\text{lst}!i) \ (b\$i) \ . \ i < - [0..<\text{dim-vec } b]] \ @$
 $\text{[EQ } (\text{lst}!i) \ ((b@_{\mathit{v}}c)\$i) \ . \ i < - [\text{dim-vec } b \ ..< \ \text{dim-vec } (b@_{\mathit{v}}c)]])$

lemma *mat-leqb-egc-for-LEQ:*

assumes $i < \text{dim-vec } b$

assumes $i < \text{dim-row } A$

shows $(\text{mat-leqb-egc } A \ b \ c)!i = \text{LEQ } ((\text{matrix-to-lpolies } A)!i) \ (b\$i)$

⟨proof⟩

lemma *mat-leqb-egc-for-EQ:*

assumes $\text{dim-vec } b \leq i \ \mathbf{and} \ i < \text{dim-vec } (b@_{\mathit{v}}c)$

assumes $\text{dim-row } A = \text{dim-vec } b \ \mathbf{and} \ \text{dim-col } A \geq \text{dim-vec } c$

shows $(\text{mat-leqb-egc } A \ b \ c)!i =$

$\text{EQ } (\text{vec-to-lpoly } (0_{\mathit{v}} \ (\text{dim-col } A) \ @_{\mathit{v}} \ \text{row } A^T \ (i - \text{dim-vec } b))) \ (c\$ (i - \text{dim-vec } b))$

⟨proof⟩

lemma *mat-leqb-egc-satisfies1:*

assumes $x \models_{cs} \text{set } (\text{mat-leqb-egc } A \ b \ c)$

assumes $i < \text{dim-vec } b$

and $i < \text{dim-row } A$

shows $(\text{matrix-to-lpolies } A)!i \ \{\!|x|\!\} \leq b\i

⟨proof⟩

lemma *mat-leqb-egc-satisfies2:*

assumes $x \models_{cs} \text{set } (\text{mat-leqb-egc } A \ b \ c)$

assumes $\text{dim-vec } b \leq i \ \mathbf{and} \ i < \text{dim-vec } (b@_{\mathit{v}}c)$

and $\text{dim-row } A = \text{dim-vec } b \ \mathbf{and} \ \text{dim-vec } c \leq \text{dim-col } A$

shows $(\text{matrix-to-lpolies } (\text{two-block-non-interfering } A \ A^T) \ ! \ i) \ \{\!|x|\!\} = (b @_{\mathit{v}} c) \$$

i

⟨proof⟩

lemma *mat-leqb-egc-simplex-satisfies2:*

assumes $\text{simplex } (\text{mat-leqb-egc } A \ b \ c) = \text{Sat } x$

assumes $\text{dim-vec } b \leq i \ \mathbf{and} \ i < \text{dim-vec } (b@_{\mathit{v}}c)$

and $\text{dim-row } A = \text{dim-vec } b$ **and** $\text{dim-vec } c \leq \text{dim-col } A$
shows (*matrix-to-lpolies (two-block-non-interfering* $A A^T$) ! i) $\{\langle x \rangle\} = (b @_v c)$
 $\$ i$
 $\langle \text{proof} \rangle$

fun *index-geq-n* **where**
index-geq-n $i n = \text{GEQ } (\text{lp-monom } 1 i) n$

lemma *index-geq-n-simplex*:
assumes $\langle x \rangle \models_c (\text{index-geq-n } i n)$
shows $\langle x \rangle i \geq n$
 $\langle \text{proof} \rangle$

fun *from-index-geq0-vector* **where**
from-index-geq0-vector $i v = [\text{GEQ } (\text{lp-monom } 1 (i+j)) (v\$j) . j < -[0..<\text{dim-vec } v]]$

lemma *from-index-geq-vector-simplex*:
assumes $x \models_{cs} \text{set } (\text{from-index-geq0-vector } i v)$
 $j < \text{dim-vec } v$
shows $x (i + j) \geq v\$j$
 $\langle \text{proof} \rangle$

lemma *from-index-geq0-vector-simplex2*:
assumes $\langle x \rangle \models_{cs} \text{set } (\text{from-index-geq0-vector } i v)$
assumes $i \leq j$ **and** $j < (\text{dim-vec } v) + i$
shows $\langle x \rangle j \geq v\$j - v\$i$
 $\langle \text{proof} \rangle$

definition *x-times-c-geq-y-times-b* **where**
x-times-c-geq-y-times-b $c b = \text{GEQ } (\text{vec-to-lpoly } (c @_v 0_v (\text{dim-vec } b)) - \text{vec-to-lpoly } (0_v (\text{dim-vec } c) @_v b)) 0$

lemma *x-times-c-geq-y-times-b-correct*:
assumes *simplex* $[x\text{-times-c-geq-y-times-b } c b] = \text{Sat } x$
shows $((\text{vec-to-lpoly } (c @_v 0_v (\text{dim-vec } b))) \{\langle x \rangle\}) \geq ((\text{vec-to-lpoly } (0_v (\text{dim-vec } c) @_v b)) \{\langle x \rangle\})$
 $\langle \text{proof} \rangle$

definition *split-i-j-x* **where**
split-i-j-x $i j x = (\text{vec } i \langle x \rangle, \text{vec } (j - i) (\lambda y. \langle x \rangle (y+i)))$

abbreviation *split-n-m-x* **where**

split-n-m-x n m $x \equiv \text{split-}i\text{-}j\text{-}x$ n $(n+m)$ x

lemma *split-vec-dims*:

assumes *split-}i-}j-}x* i j $x = (a, b)$

shows *dim-vec* $a = i$ *dim-vec* $b = (j - i)$

<proof>

lemma *split-n-m-x-abbrev-dims*:

assumes *split-n-m-x* n m $x = (a, b)$

shows *dim-vec* $a = n$ *dim-vec* $b = m$

<proof>

lemma *split-access-fst-1*:

assumes $k < i$

assumes *split-}i-}j-}x* i j $x = (a, b)$

shows $a \ \$ \ k = \langle x \rangle \ k$

<proof>

lemma *split-access-snd-1*:

assumes $i \leq k$ **and** $k < j$

assumes *split-}i-}j-}x* i j $x = (a, b)$

shows $b \ \$ \ (k - i) = \langle x \rangle \ k$

<proof>

lemma *split-access-fst-2*:

assumes $(x, y) = \text{split-}i\text{-}j\text{-}x$ i j Z

assumes $k < \text{dim-vec } x$

shows $x \ \$ \ k = \langle Z \rangle \ k$

<proof>

lemma *split-access-snd-2*:

assumes $(x, y) = \text{split-}i\text{-}j\text{-}x$ i j Z

assumes $k < \text{dim-vec } y$

shows $y \ \$ \ k = \langle Z \rangle \ (k + \text{dim-vec } x)$

<proof>

lemma *from-index-geq0-vector-split-snd*:

assumes $\langle X \rangle \models_{cs} \text{set } (\text{from-index-geq0-vector } d \ v)$

assumes $(x, y) = \text{split-n-m-x}$ d m X

shows $\bigwedge i. i < \text{dim-vec } v \implies i < m \implies y \ \$ \ i \geq v \ \$ \ i$

<proof>

lemma *split-coeff-vec-index-sum*:

assumes $(x, y) = \text{split-}i\text{-}j\text{-}x$ $(\text{dim-vec } (\text{lpoly-to-vec } v)) \ l \ X$

shows $(\sum i = 0..<\text{dim-vec } x. \text{Abstract-Linear-Poly.coeff } v \ i * \langle X \rangle \ i) =$

$(\sum i = 0..<\text{dim-vec } x. \text{lpoly-to-vec } v \ \$ \ i * x \ \$ \ i)$

<proof>

lemma *scalar-prod-valuation-after-split-equiv1*:

assumes $(x, y) = \text{split-}i\text{-}j\text{-}x \ (\text{dim-vec} \ (\text{lpoly-to-vec} \ v)) \ l \ X$

shows $(\text{lpoly-to-vec} \ v) \cdot x = (v \ \{\{X\}\})$

<proof>

definition *mat-times-vec-leq* ($[-*_v-] \leq - [1000, 1000, 100]$)

where

$$[A *_v x] \leq b \iff (\forall i < \text{dim-vec} \ b. (A *_v x)\$i \leq b\$i) \wedge \\ (\text{dim-row} \ A = \text{dim-vec} \ b) \wedge \\ (\text{dim-col} \ A = \text{dim-vec} \ x)$$

definition *vec-times-mat-eq* ($[-*_v-] = - [1000, 1000, 100]$)

where

$$[y *_v A] = c \iff (\forall i < \text{dim-vec} \ c. (A^T *_v y)\$i = c\$i) \wedge \\ (\text{dim-col} \ A^T = \text{dim-vec} \ y) \wedge \\ (\text{dim-row} \ A^T = \text{dim-vec} \ c)$$

definition *vec-times-mat-leq* ($[-*_v-] \leq - [1000, 1000, 100]$)

where

$$[y *_v A] \leq c \iff (\forall i < \text{dim-vec} \ c. (A^T *_v y)\$i \leq c\$i) \wedge \\ (\text{dim-col} \ A^T = \text{dim-vec} \ y) \wedge \\ (\text{dim-row} \ A^T = \text{dim-vec} \ c)$$

lemma *mat-times-vec-leqI[intro]*:

assumes $\text{dim-row} \ A = \text{dim-vec} \ b$

assumes $\text{dim-col} \ A = \text{dim-vec} \ x$

assumes $\bigwedge i. i < \text{dim-vec} \ b \implies (A *_v x)\$i \leq b\$i$

shows $[A *_v x] \leq b$

<proof>

lemma *mat-times-vec-leqD[dest]*:

assumes $[A *_v x] \leq b$

shows $\text{dim-row} \ A = \text{dim-vec} \ b \ \text{dim-col} \ A = \text{dim-vec} \ x \ \bigwedge i. i < \text{dim-vec} \ b \implies (A *_v x)\$i \leq b\$i$

<proof>

lemma *vec-times-mat-eqD[dest]*:

assumes $[y *_v A] = c$

shows $(\forall i < \text{dim-vec} \ c. (A^T *_v y)\$i = c\$i) \ (\text{dim-col} \ A^T = \text{dim-vec} \ y) \ (\text{dim-row} \ A^T = \text{dim-vec} \ c)$

<proof>

lemma *vec-times-mat-leqD[dest]*:

assumes $[y *_v A] \leq c$

shows $(\forall i < \text{dim-vec} \ c. (A^T *_v y)\$i \leq c\$i) \ (\text{dim-col} \ A^T = \text{dim-vec} \ y) \ (\text{dim-row} \ A^T = \text{dim-vec} \ c)$

<proof>

lemma *mat-times-vec-eqI[intro]*:

assumes $\dim\text{-col } A^T = \dim\text{-vec } x$

assumes $\dim\text{-row } A^T = \dim\text{-vec } c$

assumes $\bigwedge i. i < \dim\text{-vec } c \implies (A^T *_v x)\$i = c\$i$

shows $[x *_v A] = c$

<proof>

lemma *mat-leqb-eqc-split-correct1*:

assumes $\dim\text{-vec } b = \dim\text{-row } A$

assumes $\langle X \rangle \models_{cs} \text{set } (\text{mat-leqb-eqc } A \ b \ c)$

assumes $(x, y) = \text{split-i-j-x } (\dim\text{-col } A) \ l \ X$

shows $[A *_v x] \leq b$

<proof>

lemma *mat-leqb-eqc-split-simplex-correct1*:

assumes $\dim\text{-vec } b = \dim\text{-row } A$

assumes $\text{simplex } (\text{mat-leqb-eqc } A \ b \ c) = \text{Sat } X$

assumes $(x, y) = \text{split-i-j-x } (\dim\text{-col } A) \ l \ X$

shows $[A *_v x] \leq b$

<proof>

lemma *sat-mono*:

assumes $\text{set } A \subseteq \text{set } B$

shows $\langle X \rangle \models_{cs} \text{set } B \implies \langle X \rangle \models_{cs} \text{set } A$

<proof>

lemma *mat-leqb-eqc-split-subset-correct1*:

assumes $\dim\text{-vec } b = \dim\text{-row } A$

assumes $\text{set } (\text{mat-leqb-eqc } A \ b \ c) \subseteq \text{set } S$

assumes $\text{simplex } S = \text{Sat } X$

assumes $(x, y) = \text{split-i-j-x } (\dim\text{-col } A) \ l \ X$

shows $[A *_v x] \leq b$

<proof>

lemma *mat-leqb-eqc-split-correct2*:

assumes $\dim\text{-vec } c = \dim\text{-row } A^T$

assumes $\dim\text{-vec } b = \dim\text{-col } A^T$

assumes $\langle X \rangle \models_{cs} \text{set } (\text{mat-leqb-eqc } A \ b \ c)$

assumes $(x, y) = \text{split-n-m-x } (\dim\text{-row } A^T) \ (\dim\text{-col } A^T) \ X$

shows $[y *_v A] = c$

<proof>

lemma *mat-leqb-eqc-split-simplex-correct2*:

assumes $\dim\text{-vec } c = \dim\text{-row } A^T$

assumes $\dim\text{-vec } b = \dim\text{-col } A^T$

assumes $\text{simplex } (\text{mat-leqb-eqc } A \ b \ c) = \text{Sat } X$

assumes $(x, y) = \text{split-n-m-x } (\dim\text{-row } A^T) \ (\dim\text{-col } A^T) \ X$

shows $[y \text{ } _v * A] = c$
 ⟨*proof*⟩

lemma *mat-leqb-eqc-correct*:
assumes $\text{dim-vec } c = \text{dim-row } A^T$
and $\text{dim-vec } b = \text{dim-col } A^T$
assumes $\text{simplex } (\text{mat-leqb-eqc } A \ b \ c) = \text{Sat } X$
assumes $(x, y) = \text{split-n-m-x } (\text{dim-row } A^T) (\text{dim-col } A^T) \ X$
shows $[y \text{ } _v * A] = c \ [A \ *_v \ x] \leq b$
 ⟨*proof*⟩

lemma *eval-lpoly-eq-dot-prod-split1*:
assumes $(x, y) = \text{split-n-m-x } (\text{dim-vec } c) (\text{dim-vec } b) \ X$
shows $(\text{vec-to-lpoly } c) \ \{\langle X \rangle\} = c \cdot x$
 ⟨*proof*⟩

lemma *eval-lpoly-eq-dot-prod-split2*:
assumes $(x, y) = \text{split-n-m-x } (\text{dim-vec } b) (\text{dim-vec } c) \ X$
shows $(\text{vec-to-lpoly } (0_v \ (\text{dim-vec } b) \ @_v \ c)) \ \{\langle X \rangle\} = c \cdot y$
 ⟨*proof*⟩

lemma *x-times-c-geq-y-times-b-split-dotP*:
assumes $\langle X \rangle \models_c \text{x-times-c-geq-y-times-b } c \ b$
assumes $(x, y) = \text{split-n-m-x } (\text{dim-vec } c) (\text{dim-vec } b) \ X$
shows $c \cdot x \geq b \cdot y$
 ⟨*proof*⟩

lemma *mult-right-leq*:
fixes $A :: ('a :: \{\text{comm-semiring-1}, \text{ordered-semiring}\}) \ \text{mat}$
assumes $\text{dim-vec } y = \text{dim-vec } b$
and $\forall i < \text{dim-vec } y. \ y \$i \geq 0$
and $[A \ *_v \ x] \leq b$
shows $(A \ *_v \ x) \cdot y \leq b \cdot y$
 ⟨*proof*⟩

lemma *mult-right-eq*:
assumes $\text{dim-vec } x = \text{dim-vec } c$
and $[y \text{ } _v * A] = c$
shows $(A^T \ *_v \ y) \cdot x = c \cdot x$
 ⟨*proof*⟩

lemma *soundness-mat-x-leq*:
assumes $\text{dim-row } A = \text{dim-vec } b$
assumes $\text{simplex } (\text{mat-x-leq-vec } A \ b) = \text{Sat } X$
shows $\exists x. [A \ *_v \ x] \leq b$
 ⟨*proof*⟩

lemma *completeness-mat-x-leq*:
assumes $\exists x. [A \ *_v \ x] \leq b$

shows $\exists X. \text{simplex} (\text{mat-x-leq-vec } A \ b) = \text{Sat } X$
 $\langle \text{proof} \rangle$

lemma *soundness-mat-x-eq-vec*:
assumes $\text{dim-row } A^T = \text{dim-vec } c$
assumes $\text{simplex} (\text{x-mat-eq-vec } c \ A^T) = \text{Sat } X$
shows $\exists x. [x \ v^* \ A]=c$
 $\langle \text{proof} \rangle$

lemma *completeness-mat-x-eq-vec*:
assumes $\exists x. [x \ v^* \ A]=c$
shows $\exists X. \text{simplex} (\text{x-mat-eq-vec } c \ A^T) = \text{Sat } X$
 $\langle \text{proof} \rangle$

lemma *soundness-mat-leqb-eqc1*:
assumes $\text{dim-row } A = \text{dim-vec } b$
assumes $\text{simplex} (\text{mat-leqb-eqc } A \ b \ c) = \text{Sat } X$
shows $\exists x. [A \ *_v \ x] \leq b$
 $\langle \text{proof} \rangle$

lemma *soundness-mat-leqb-eqc2*:
assumes $\text{dim-row } A^T = \text{dim-vec } c$
assumes $\text{dim-col } A^T = \text{dim-vec } b$
assumes $\text{simplex} (\text{mat-leqb-eqc } A \ b \ c) = \text{Sat } X$
shows $\exists y. [y \ v^* \ A]=c$
 $\langle \text{proof} \rangle$

lemma *completeness-mat-leqb-eqc*:
assumes $\exists x. [A \ *_v \ x] \leq b$
and $\exists y. [y \ v^* \ A]=c$
shows $\exists X. \text{simplex} (\text{mat-leqb-eqc } A \ b \ c) = \text{Sat } X$
 $\langle \text{proof} \rangle$

lemma *sound-and-complte-mat-leqb-eqc [iff]*:
assumes $\text{dim-row } A^T = \text{dim-vec } c$
assumes $\text{dim-col } A^T = \text{dim-vec } b$
shows $(\exists x. [A \ *_v \ x] \leq b) \wedge (\exists y. [y \ v^* \ A]=c) \longleftrightarrow (\exists X. \text{simplex} (\text{mat-leqb-eqc } A \ b \ c) = \text{Sat } X)$
 $\langle \text{proof} \rangle$

7 Translate Inequalities to Matrix Form

fun *nonstrict-constr* **where**
nonstrict-constr (*LEQ* $p \ r$) = *True* |
nonstrict-constr (*GEQ* $p \ r$) = *True* |
nonstrict-constr (*EQ* $p \ r$) = *True* |
nonstrict-constr - = *False*

abbreviation *nonstrict-constrs* $cs \equiv (\forall a \in \text{set } cs. \text{nonstrict-constr } a)$

fun *transf-constraint* **where**
transf-constraint (*LEQ* *p r*) = [*LEQ* *p r*] |
transf-constraint (*GEQ* *p r*) = [*LEQ* (*-p*) (*-r*)] |
transf-constraint (*EQ* *p r*) = [*LEQ* *p r*, *LEQ* (*-p*) (*-r*)] |
transf-constraint - = []

fun *transf-constraints* **where**
transf-constraints [] = [] |
transf-constraints (*x#xs*) = *transf-constraint* *x* @ (*transf-constraints* *xs*)

lemma *trans-constraint-creates-LEQ-only*:
assumes *transf-constraint* *x* ≠ []
shows (∀ *x* ∈ *set* (*transf-constraint* *x*). ∃ *a b*. *x* = *LEQ* *a b*)
⟨*proof*⟩

lemma *trans-constraints-creates-LEQ-only*:
assumes *transf-constraints* *xs* ≠ []
assumes *x* ∈ *set* (*transf-constraints* *xs*)
shows ∃ *p r*. *LEQ* *p r* = *x*
⟨*proof*⟩

lemma *non-strict-constr-no-LT*:
assumes *nonstrict-constrs* *cs*
shows ∀ *x* ∈ *set* *cs*. ¬(∃ *a b*. *LT* *a b* = *x*)
⟨*proof*⟩

lemma *non-strict-constr-no-GT*:
assumes *nonstrict-constrs* *cs*
shows ∀ *x* ∈ *set* *cs*. ¬(∃ *a b*. *GT* *a b* = *x*)
⟨*proof*⟩

lemma *non-strict-constrs-cond*:
assumes ∧*x*. *x* ∈ *set* *cs* ⇒ ¬(∃ *a b*. *LT* *a b* = *x*)
assumes ∧*x*. *x* ∈ *set* *cs* ⇒ ¬(∃ *a b*. *GT* *a b* = *x*)
shows *nonstrict-constrs* *cs*
⟨*proof*⟩

lemma *sat-constr-sat-transf-constrs*:
assumes *v* ⊨_{*c*} *cs*
shows *v* ⊨_{*cs*} *set* (*transf-constraint* *cs*)
⟨*proof*⟩

lemma *sat-constrs-sat-transf-constrs*:
assumes *v* ⊨_{*cs*} *set* *cs*
shows *v* ⊨_{*cs*} *set* (*transf-constraints* *cs*)

<proof>

lemma *sat-transf-constrs-sat-constr*:

assumes *nonstrict-constr cs*

assumes $v \models_{cs} \text{set } (\text{transf-constraint } cs)$

shows $v \models_c cs$

<proof>

lemma *sat-transf-constrs-sat-constrs*:

assumes *nonstrict-constrs cs*

assumes $v \models_{cs} \text{set } (\text{transf-constraints } cs)$

shows $v \models_{cs} \text{set } cs$

<proof>

end

theory *Linear-Programming*

imports

HOL-Library.Code-Target-Int

LP-Preliminaries

Farkas.Simplex-for-Reals

begin

8 Abstract LPs

Primal Problem

definition *sat-primal* $A \ b = \{ x. [A \ *_{\nu} \ x] \leq b \}$

Dual Problem

definition *sat-dual* $A \ c = \{ y. [y \ \nu^* \ A] = c \wedge (\forall i < \text{dim-vec } y. y \ \$ \ i \geq 0) \}$

definition *optimal-set* $f \ S = \{ x \in S. (\forall y \in S. f \ x \ y) \}$

abbreviation *max-lp where*

$\text{max-lp } A \ b \ c \equiv \text{optimal-set } (\lambda x \ y. (y \cdot c) \leq (x \cdot c)) \ (\text{sat-primal } A \ b)$

abbreviation *min-lp where*

$\text{min-lp } A \ b \ c \equiv \text{optimal-set } (\lambda x \ y. (y \cdot c) \geq (x \cdot c)) \ (\text{sat-dual } A \ c)$

lemma *optimal-setI[intro]*:

assumes $x \in S$

assumes $\bigwedge y. y \in S \implies (\lambda x \ y. (y \cdot c) \geq (x \cdot c)) \ x \ y$

shows $x \in \text{optimal-set } (\lambda x \ y. (y \cdot c) \geq (x \cdot c)) \ S$

<proof>

lemma *max-lpI [intro]*:

assumes $[A \ *_{\nu} \ x] \leq b$

assumes $(\bigwedge y. [A \ *_{\nu} \ y] \leq b \implies (\lambda x \ y. (y \cdot c) \geq (x \cdot c)) \ y \ x)$

shows $x \in \text{max-lp } A \ b \ c$
<proof>

lemma *min-lpI* [*intro*]:
assumes $[y \ v^* \ A]=c$
and $(\bigwedge i. i < \text{dim-vec } y \implies y \ \$ \ i \geq 0)$
assumes $(\bigwedge x. x \in \text{sat-dual } A \ c \implies (\lambda x \ y. (y \cdot c) \geq (x \cdot c)) \ y \ x)$
shows $y \in \text{min-lp } A \ b \ c$
<proof>

lemma *sat-primalD* [*dest*]:
assumes $x \in \text{sat-primal } A \ b$
shows $[A \ *_v \ x] \leq b$
<proof>

lemma *sat-primalI* [*intro*]:
assumes $[A \ *_v \ x] \leq b$
shows $x \in \text{sat-primal } A \ b$
<proof>

lemma *sat-dualD* [*dest*]:
assumes $y \in \text{sat-dual } A \ c$
shows $[y \ v^* \ A]=c \ (\forall i < \text{dim-vec } y. y \ \$ \ i \geq 0)$
<proof>

lemma *sat-dualI* [*intro*]:
assumes $[y \ v^* \ A]=c \ (\forall i < \text{dim-vec } y. y \ \$ \ i \geq 0)$
shows $y \in \text{sat-dual } A \ c$
<proof>

lemma *sol-dim-in-sat-primal*: $x \in \text{sat-primal } A \ b \implies \text{dim-vec } x = \text{dim-col } A$
<proof>

lemma *sol-dim-in-max-lp*: $x \in \text{max-lp } A \ b \ c \implies \text{dim-vec } x = \text{dim-col } A$
<proof>

lemma *sol-dim-in-sat-dual*: $x \in \text{sat-dual } A \ c \implies \text{dim-vec } x = \text{dim-row } A$
<proof>

lemma *sol-dim-in-min-lp*: $x \in \text{min-lp } A \ b \ c \implies \text{dim-vec } x = \text{dim-row } A$
<proof>

lemma *min-lp-in-sat-dual*: $x \in \text{min-lp } A \ b \ c \implies x \in \text{sat-dual } A \ c$
<proof>

lemma *max-lp-in-sat-primal*: $x \in \text{max-lp } A \ b \ c \implies x \in \text{sat-primal } A \ b$
<proof>

```

locale abstract-LP =
  fixes  $A :: ('a::\{comm-semiring-1,ordered-semiring,linorder\}) mat$ 
  fixes  $b :: 'a vec$ 
  fixes  $c :: 'a vec$ 
  fixes  $m$ 
  fixes  $n$ 
  assumes  $b \in carrier-vec\ m$ 
  assumes  $c \in carrier-vec\ n$ 
  assumes  $A \in carrier-mat\ m\ n$ 
begin

lemma dim-b-row-A:  $dim-vec\ b = dim-row\ A$ 
   $\langle proof \rangle$ 

lemma dim-b-col-A:  $dim-vec\ c = dim-col\ A$ 
   $\langle proof \rangle$ 

lemma weak-duality-aux:
  fixes  $i\ j$ 
  assumes  $i \in \{c \cdot x \mid x. x \in sat-primal\ A\ b\}$ 
  and  $j \in \{b \cdot y \mid y. y \in sat-dual\ A\ c\}$ 
  shows  $i \leq j$ 
   $\langle proof \rangle$ 

theorem weak-duality-theorem:
  assumes  $x \in max-lp\ A\ b\ c$ 
  assumes  $y \in min-lp\ A\ b\ c$ 
  shows  $x \cdot c \leq y \cdot b$ 
   $\langle proof \rangle$ 

end

fun create-optimal-solutions where
  create-optimal-solutions  $A\ b\ c =$ 
    (case simplex (x-times-c-geq-y-times-b c b #
      mat-leqb-eqc A b c @
      from-index-geq0-vector (dim-vec c) (0_v (dim-vec b)))
    of
      Unsat X  $\Rightarrow$  Unsat X
      | Sat X  $\Rightarrow$  Sat X)

fun optimize-no-cond where optimize-no-cond  $A\ b\ c =$  (case create-optimal-solutions
   $A\ b\ c$  of
    Unsat X  $\Rightarrow$  Unsat X
    | Sat X  $\Rightarrow$  Sat (fst (split-n-m-x (dim-vec c) (dim-vec b) X)))

lemma create-opt-sol-satisfies:
  assumes create-optimal-solutions  $A\ b\ c = Sat\ X$ 
  shows  $\langle X \rangle \models_{cs} set ((x-times-c-geq-y-times-b\ c\ b\ \# mat-leqb-eqc\ A\ b\ c\ @$ 

```


from-index-geq0-vector (dim-vec c) (0_v (dim-vec b)))

<proof>

lemma *create-opt-sol-sat-leq-mat:*
assumes *dim-vec b = dim-row A*
assumes *create-optimal-solutions A b c = Sat X*
and *(x, y) = split-i-j-x (dim-col A) (dim-vec b) X*
shows $[A *_v x] \leq b$
<proof>

lemma *create-opt-sol-sat-eq-mat:*
assumes *dim-vec c = dim-row A^T*
and *dim-vec b = dim-col A^T*
assumes *create-optimal-solutions A b c = Sat X*
and *(x, y) = split-i-j-x (dim-vec c) (dim-vec c + dim-vec b) X*
shows $[y *_v A] = c$
<proof>

lemma *create-opt-sol-satisfies-leq:*
assumes *create-optimal-solutions A b c = Sat X*
assumes *(x, y) = split-n-m-x (dim-vec c) (dim-vec b) X*
shows $x \cdot c \geq y \cdot b$
<proof>

lemma *create-opt-sol-satisfies-geq0:*
assumes *create-optimal-solutions A b c = Sat X*
assumes *(x, y) = split-n-m-x (dim-vec c) (dim-vec b) X*
shows $\bigwedge i. i < \text{dim-vec } y \implies y\$i \geq 0$
<proof>

locale *rat-LP = abstract-LP A b c m n*
for *A :: rat mat*
and *b :: rat vec*
and *c :: rat vec*
and *m :: nat*
and *n :: nat*
begin

lemma *create-opt-sol-in-LP:*
assumes *create-optimal-solutions A b c = Sat X*
assumes *(x, y) = split-n-m-x (dim-vec c) (dim-vec b) X*
shows $[A *_v x] \leq b \wedge [y *_v A] = c \wedge x \cdot c \geq y \cdot b \wedge \bigwedge i. i < \text{dim-vec } y \implies y\$i \geq 0$
<proof>

lemma *create-optim-in-sols:*
assumes *create-optimal-solutions A b c = Sat X*
assumes *(x, y) = split-n-m-x (dim-vec c) (dim-vec b) X*
shows $c \cdot x \in \{c \cdot x \mid x. [A *_v x] \leq b\}$

$b \cdot y \in \{b \cdot y \mid y. [y \ v^* \ A]=c \wedge (\forall i < \dim\text{-vec } y. y\$i \geq 0)\}$
<proof>

lemma *cx-leq-bx-in-creating-opt:*

assumes *create-optimal-solutions* $A \ b \ c = \text{Sat } X$

assumes $(x, y) = \text{split-n-m-x } (\dim\text{-vec } c) (\dim\text{-vec } b) \ X$

shows $c \cdot x \leq b \cdot y$

<proof>

lemma *min-max-for-sol:*

assumes *create-optimal-solutions* $A \ b \ c = \text{Sat } X$

assumes $(x, y) = \text{split-n-m-x } (\dim\text{-vec } c) (\dim\text{-vec } b) \ X$

shows $c \cdot x = b \cdot y$

<proof>

lemma *create-opt-solutions-correct:*

assumes *create-optimal-solutions* $A \ b \ c = \text{Sat } X$

assumes $(x, y) = \text{split-n-m-x } (\dim\text{-vec } c) (\dim\text{-vec } b) \ X$

shows $x \in \text{max-lp } A \ b \ c$

<proof>

lemma *optimize-no-cond-correct:*

assumes *optimize-no-cond* $A \ b \ c = \text{Sat } x$

shows $x \in \text{max-lp } A \ b \ c$

<proof>

lemma *optimize-no-cond-sol-sat:*

assumes *optimize-no-cond* $A \ b \ c = \text{Sat } x$

shows $x \in \text{sat-primal } A \ b$

<proof>

end

fun *maximize* **where**

maximize $A \ b \ c = (\text{if } \dim\text{-vec } b = \dim\text{-row } A \wedge \dim\text{-vec } c = \dim\text{-col } A \text{ then}$

$\text{Some } (\text{optimize-no-cond } A \ b \ c)$

$\text{else None})$

lemma *optimize-sound:*

assumes *maximize* $A \ b \ c = \text{Some } (\text{Sat } x)$

shows $x \in \text{max-lp } A \ b \ c$

<proof>

lemma *maximize-option-elim:*

assumes *maximize* $A \ b \ c = \text{Some } x$

shows $\dim\text{-vec } b = \dim\text{-row } A \ \dim\text{-vec } c = \dim\text{-col } A$

<proof>

lemma *optimize-sol-dimension*:
 assumes *maximize A b c = Some (Sat x)*
 shows $x \in \text{carrier-vec } (\text{dim-col } A)$
 $\langle \text{proof} \rangle$

lemma *optimize-sat*:
 assumes *maximize A b c = Some (Sat x)*
 shows $[A *_v x] \leq b$
 $\langle \text{proof} \rangle$

derive (*eq*) *ceq rat*
derive (*linorder*) *compare rat*
derive (*compare*) *ccompare rat*
derive (*rbt*) *set-impl rat*

derive (*eq*) *ceq atom QDelta*
derive (*linorder*) *compare-order QDelta*
derive *compare-order atom*
derive *ccompare atom QDelta*
derive (*rbt*) *set-impl atom QDelta*

lemma *of-rat-val: simplex cs = (Sat v) \implies of-rat-val $\langle v \rangle \models_{rcs}$ set cs*
 $\langle \text{proof} \rangle$

end

References

- [1] X. Allamigeon and R. D. Katz. A formalization of convex polyhedra based on the simplex method. In M. Ayala-Rincón and C. A. Muñoz, editors, *Interactive Theorem Proving*, pages 28–45, Cham, 2017. Springer International Publishing.
- [2] S. Boulmé and A. Maréchal. A Coq tactic for equality learning in linear arithmetic. In J. Avigad and A. Mahboubi, editors, *Interactive Theorem*

Proving, pages 108–125, Cham, 2018. Springer International Publishing.

- [3] F. Mari, M. Spasi, 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] S. Obua and T. Nipkow. Flyspeck II: the basic linear programs. *Annals of Mathematics and Artificial Intelligence*, 56(3):245–272, Aug 2009.
- [5] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.
- [6] M. Spasić and F. Marić. Formalization of incremental simplex algorithm by stepwise refinement. In D. Giannakopoulou and D. Méry, editors, *FM 2012: Formal Methods*, pages 434–449, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.