

# 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

<b>1</b>	<b>Related work</b>	<b>1</b>
<b>2</b>	<b>General Theorems used later, that could be moved</b>	<b>2</b>
<b>3</b>	<b>Vectors</b>	<b>5</b>
<b>4</b>	<b>Translations of Jordan Normal Forms Matrix Library to Simplex polynomials</b>	<b>14</b>
4.1	Vectors . . . . .	14
<b>5</b>	<b>Matrices</b>	<b>21</b>
<b>6</b>	<b>Get different matrices into same space, without interference</b>	<b>25</b>
<b>7</b>	<b>Translate Inequalities to Matrix Form</b>	<b>47</b>
<b>8</b>	<b>Abstract LPs</b>	<b>49</b>

## 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 mechanism 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}}*

**by** (*simp*) (*fastforce*)

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

**by** (*metis diff-add-inverse diff-less-mono in-set-conv-nth leD length-append nth-append*)

## 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** –

**define** *nr* **where** *nr: nr = dim-row (four-block-mat A B C D)*

**define** *nc* **where** *nc: nc = dim-col (four-block-mat A B C D)*

**define** *nr2* **where** *nr2: nr2 = nr - dim-row A*

**define** *nc2* **where** *nc2: nc2 = nc - dim-col A*

**define** *A1* **where** *A1: A1 = mat (dim-row A) (dim-col A) ((\$\$) (four-block-mat A B C D))*

**define** *A2* **where** *A2: A2 = mat (dim-row A) nc2 (λ(i, j). (four-block-mat A B C D) \$\$ (i, j + dim-col A))*

**define** *A3* **where** *A3: A3 = mat nr2 (dim-col A) (λ(i, j). (four-block-mat A B C D) \$\$ (i + dim-row A, j))*

**define** *A4* **where** *A4: A4 = mat nr2 nc2 (λ(i, j). (four-block-mat A B C D) \$\$ (i + dim-row A, j + dim-col A))*

**have** *g: split-block (four-block-mat A B C D) (dim-row A) (dim-col A) = (A1, A2, A3, A4)*

**using** *split-block-def[of (four-block-mat A B C D) (dim-row A) (dim-col A)]*

**by** (*metis A1 A2 A3 A4 nc nc2 nr nr2*)

**have** *D: D = A4*

**using** *A4* **by** (*auto*) (*standard, (simp add: nr nr2 nc nc2)+*)

**have** *A = A1*

**using** *A1* **by** *auto*

**then have** *split-block (four-block-mat A B C D) (dim-row A) (dim-col A) = (A, A2, A3, D)*

**using** *D g* **by** *blast*

**also show**  $U = A$   
**using** *assms calculation by auto*  
**ultimately show**  $V = D$   
**using** *assms by auto*  
**qed**

**lemma** *append-split-vec-distrib-scalar-prod*:  
**assumes**  $\dim\text{-vec } (u @_v w) = \dim\text{-vec } x$   
**shows**  $(u @_v w) \cdot x = u \cdot (\text{vec-first } x (\dim\text{-vec } u)) + w \cdot (\text{vec-last } x (\dim\text{-vec } w))$   
**proof** –  
**have**  $(u @_v w) \cdot (\text{vec-first } x (\dim\text{-vec } u) @_v \text{vec-last } x (\dim\text{-vec } w)) =$   
 $u \cdot \text{vec-first } x (\dim\text{-vec } u) + w \cdot \text{vec-last } x (\dim\text{-vec } w)$   
**by** (*meson carrier-vec-dim-vec scalar-prod-append vec-first-carrier vec-last-carrier*)  
**then show** *?thesis*  
**by** (*metis assms carrier-vec-dim-vec index-append-vec(2) vec-first-last-append*)  
**qed**

**lemma** *append-dot-product-split*:  
**assumes**  $\dim\text{-vec } (u @_v w) = \dim\text{-vec } x$   
**shows**  $(u @_v w) \cdot x = (\sum i \in \{0..< \dim\text{-vec } u\}. u\$i * x\$i) + (\sum i \in \{0..< \dim\text{-vec } w\}. w\$i * x\$(i + \dim\text{-vec } u))$   
**proof** –  
**define** *ix* **where**  $ix = \text{vec-first } x (\dim\text{-vec } u)$   
**define** *lx* **where**  $lx = \text{vec-last } x (\dim\text{-vec } w)$   
**have**  $*$ :  $(u @_v w) \cdot x = u \cdot ix + w \cdot lx$   
**using** *append-split-vec-distrib-scalar-prod ix-def lx assms by blast*  
**have**  $(u @_v w) \cdot x = (\sum i \in \{0..< \dim\text{-vec } x\}. (u @_v w) \$ i * x \$ i)$   
**using** *scalar-prod-def[of (u @\_v w) x] by simp*  
**also have**  $\dots = (\sum i \in \{0..< \dim\text{-vec } u\}. (u @_v w) \$ i * x \$ i) +$   
 $(\sum i \in \{\dim\text{-vec } u..< \dim\text{-vec } (u @_v w)\}. (u @_v w) \$ i * x \$ i)$   
**using** *assms sum.atLeastLessThan-concat[of 0 dim-vec u dim-vec (u @\_v w)*  
 $(\lambda i. (u @_v w) \$ i * x \$ i), OF le0[of \dim\text{-vec } u]$   
 $le-add1[of \dim\text{-vec } u \dim\text{-vec } w] \text{index-append-vec}(2)[of u w]$  **by** *simp*  
**also have**  $*$ :  $\dots = (\sum i \in \{0..< \dim\text{-vec } u\}. u\$i * x\$i) + w \cdot lx$   
**using** *\* calculation by (auto simp: ix-def scalar-prod-def vec-first-def)*  
**have**  $w \cdot lx = (\sum i \in \{0..< \dim\text{-vec } w\}. w\$i * x\$(i + \dim\text{-vec } u))$  **unfolding** *lx*  
*vec-last-def*  
**unfolding** *scalar-prod-def* **using** *add-diff-cancel-right' index-append-vec(2)[of u w]* **by** (*auto*)  
 $(\text{metis } \langle \dim\text{-vec } (u @_v w) = \dim\text{-vec } u + \dim\text{-vec } w \rangle \text{add.commute add-diff-cancel-right' assms})$   
**then show** *?thesis*  
**using** *\* calculation by auto*  
**qed**

**lemma** *assoc-scalar-prod-mult-mat-vec*:  
**fixes**  $A :: 'a::\text{comm-semiring-1} \text{ mat}$   
**assumes**  $y \in \text{carrier-vec } n$

**assumes**  $x \in \text{carrier-vec } m$   
**assumes**  $A \in \text{carrier-mat } n \ m$   
**shows**  $(A *_v x) \cdot y = (A^T *_v y) \cdot x$   
**proof** –  
**have**  $(A *_v x) \cdot y = (\sum i \in \{0 ..< n\}. (A *_v x) \$ i * y \$ i)$   
**unfolding** *scalar-prod-def* **using** *assms(1) carrier-vecD* **by** *blast*  
**also have**  $\dots = (\sum i \in \{0 ..< n\}. (\text{vec } (\text{dim-row } A) (\lambda i. \text{row } A \ i \cdot x)) \$ i * y$   
 $\$ i)$   
**unfolding** *mult-mat-vec-def* **by** *blast*  
**also have**  $\dots = (\sum i \in \{0 ..< n\}. (\lambda i. \text{row } A \ i \cdot x) \ i * y \$ i)$   
**using** *assms(3)* **by** *auto*  
**also have**  $\dots = (\sum i \in \{0 ..< n\}. (\sum j \in \{0 ..< m\}. (\text{row } A \ i) \$ j * x \$ j) * y \$ i)$   
**unfolding** *scalar-prod-def* **using** *assms(2) carrier-vecD* **by** *blast*  
**also have**  $\dots = (\sum j \in \{0 ..< n\}. (\sum i \in \{0 ..< m\}. (\text{row } A \ j) \$ i * x \$ i * y$   
 $\$ j))$   
**by** (*simp add: sum-distrib-right*)  
**also have**  $\dots = (\sum j \in \{0 ..< n\}. (\sum i \in \{0 ..< m\}. A \$\$ (j,i) * x \$ i * y \$ j))$   
 $\$ j))$   
**unfolding** *row-def* **using** *assms(3)* **by** *auto*  
**also have**  $\dots = (\sum j \in \{0 ..< n\}. (\sum i \in \{0 ..< m\}. A \$\$ (j,i) * y \$ j * x \$ i))$   
**by** (*meson semiring-normalization-rules(16) sum.cong*)  
**also have**  $\dots = (\sum j \in \{0 ..< n\}. (\sum i \in \{0 ..< m\}. (\text{col } A \ i) \$ j * y \$ j * x$   
 $\$ i))$   
**using** *assms(3)* **by** *auto*  
**also have**  $\dots = (\sum i \in \{0 ..< m\}. (\sum j \in \{0 ..< n\}. (\text{col } A \ i) \$ j * y \$ j * x$   
 $\$ i))$   
**using** *Groups-Big.comm-monoid-add-class.sum.swap[of*  
 $(\lambda i \ j. (\text{col } A \ i) \$ j * y \$ j * x \$ i) \{0..<n\} \{0 ..< m\}, \text{symmetric}]$   
**by** *simp*  
**also have**  $\dots = (\sum i \in \{0 ..< m\}. (\sum j \in \{0 ..< n\}. (\text{col } A \ i) \$ j * y \$ j) * x$   
 $\$ i)$   
**by** (*simp add: sum-distrib-right*)  
**also have**  $\dots = (\sum i \in \{0 ..< m\}. (\lambda i. \text{col } A \ i \cdot y) \ i * x \$ i)$   
**unfolding** *scalar-prod-def* **using** *assms(1) carrier-vecD* **by** *blast*  
**also have**  $\dots = (\sum i \in \{0 ..< m\}. (\lambda i. \text{row } A^T \ i \cdot y) \ i * x \$ i)$   
**using** *assms(3)* **by** *auto*  
**also have**  $\dots = (\sum i \in \{0 ..< m\}. (\text{vec } (\text{dim-row } A^T) (\lambda i. \text{row } A^T \ i \cdot y)) \$ i$   
 $* x \$ i)$   
**using** *assms* **by** *auto*  
**also have**  $\dots = (\sum i \in \{0 ..< m\}. (A^T *_v y) \$ i * x \$ i)$   
**using** *assms* **by** *auto*  
**also have**  $\dots = (A^T *_v y) \cdot x$   
**using** *scalar-prod-def[of (A^T \*\_v y) x,symmetric]* **using** *assms(2) carrier-vecD*  
**by** *blast*  
**finally show** *?thesis* .  
**qed**

### 3 Vectors

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

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

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

**lemma** *vector-cases-append*:

**fixes**  $x :: 'a \ \text{vec}$

**shows**  $x = vNil \ \vee \ (\exists v \ a. \ x = v \ @_v \ [a]_v)$

**proof** –

**have**  $x \neq vNil \implies (\exists v \ a. \ x = v \ @_v \ [a]_v)$

**proof** (*rule ccontr*)

**assume**  $a1: x \neq vNil$

**assume**  $na: \neg (\exists v \ a. \ x = v \ @_v \ [a]_v)$

**have**  $\text{dim-vec } x \geq 1$

**using**  $a1 \ \text{eq-vecI}$  **by** *auto*

**define**  $v$  **where**  $v = \text{vec} \ (\text{dim-vec } x - 1) \ (\lambda i. \ x \ \$ \ i)$

**have**  $v': \forall i < \text{dim-vec } v. \ v \ \$ \ i = x \ \$ \ i$

**using**  $v$  **by** *auto*

**define**  $a$  **where**  $a = x \ \$ \ (\text{dim-vec } x - 1)$

**have**  $a': [a]_v \ \$ \ 0 = a$  **by** *simp*

**have**  $ff1: 1 + \text{dim-vec } v = \text{dim-vec } x$

**by** (*metis* (*no-types*)  $\langle 1 \leq \text{dim-vec } x \rangle \ \text{add-diff-cancel-left}' \ \text{dim-vec } \text{le-Suc-ex}$

$v$ )

**have**  $\forall i < \text{dim-vec } x. \ x \ \$ \ i = (v \ @_v \ [a]_v) \ \$ \ i$

**proof** (*standard,standard*)

**fix**  $i :: \text{nat}$

**assume**  $as: i < \text{dim-vec } x$

**have**  $x \ \$ \ \text{dim-vec } v = a$

**by** (*simp*  $\text{add: } a \ v$ )

**then have**  $x \ \$ \ i = (v \ @_v \ [a]_v) \ \$ \ i$

**using**  $ff1$  **as** **by** (*metis* (*no-types*)  $\text{One-nat-def } a' \ \text{add.left-neutral}$

$\text{add-Suc-right} \ \text{add-diff-cancel-left}' \ \text{add-diff-cancel-right}'$

$\text{dim-vec } \text{index-append-vec}(1) \ \text{less-Suc-eq } v'$ )

**then show**  $x \ \$ \ i = (v \ @_v \ [a]_v) \ \$ \ i$

**by** *blast*

**qed**

**then have**  $x = v \ @_v \ [a]_v$

**using**  $a \ a' \ v \ v'$

**by** (*metis*  $\text{dim-vec } \text{eq-vecI} \ ff1 \ \text{index-append-vec}(2) \ \text{semiring-normalization-rules}(24)$ )

**then show** *False* **using**  $na$  **by** *auto*

**qed**

**then show** *?thesis*

**by** *blast*

**qed**

**lemma** *vec-rev-induct* [*case-names* *vNil append*, *induct type: vec*]:

**assumes**  $P \ vNil$  and  $\bigwedge a \ v. P \ v \implies P \ (v \ @_v \ [a]_v)$

**shows**  $P \ v$

**proof** (*induction dim-vec v arbitrary: v*)

**case**  $0$

**then have**  $v = vNil$

**by** *auto*

**then show** *?case*

**using** *assms(1)* **by** *auto*

**next**

**case** (*Suc l*)

**obtain**  $xs \ x$  **where**  $xs\text{-}x: v = xs \ @_v \ [x]_v$

**using** *vector-cases-append[of v] Suc.hyps(2) dim-vec* **by** (*auto*)

**have**  $l = dim\text{-}vec \ xs$

**using** *Suc.hyps(2) xs-x* **by** *auto*

**then have**  $P \ xs$

**using** *Suc.hyps(1)[of xs]* **by** *auto*

**then have**  $P \ (xs \ @_v \ [x]_v)$

**using** *assms(2)[of xs x]* **by** *auto*

**then show** *?case*

**by** (*simp add: xs-x*)

**qed**

**lemma** *singleton-append-dotP*:

**assumes**  $dim\text{-}vec \ z = dim\text{-}vec \ y + 1$

**shows**  $(y \ @_v \ [x]_v) \cdot z = (\sum i \in \{0..<dim\text{-}vec \ y\}. y \ \$ \ i \ * \ z \ \$ \ i) + x \ * \ z \ \$ \ dim\text{-}vec$

$y$

**proof** –

**have**  $(y \ @_v \ [x]_v) \cdot z = (\sum i \in \{0..<dim\text{-}vec \ z\}. (y \ @_v \ [x]_v) \ \$ \ i \ * \ z \ \$ \ i)$

**unfolding** *scalar-prod-def* **by** *blast*

**also have**  $\dots = (\sum i \in \{0..<dim\text{-}vec \ z-1\}. (y \ @_v \ [x]_v) \ \$ \ i \ * \ z \ \$ \ i) +$   
 $(y \ @_v \ [x]_v) \ \$ \ (dim\text{-}vec \ z-1) \ * \ z \ \$ \ (dim\text{-}vec \ z-1)$

**by** (*simp add: assms*)

**also have**  $\dots = (\sum i \in \{0..<dim\text{-}vec \ y\}. (y \ @_v \ [x]_v) \ \$ \ i \ * \ z \ \$ \ i) +$   
 $(y \ @_v \ [x]_v) \ \$ \ (dim\text{-}vec \ y) \ * \ z \ \$ \ (dim\text{-}vec \ y)$

**using** *assms* **by** *auto*

**also have**  $\dots = (\sum i \in \{0..<dim\text{-}vec \ y\}. y \ \$ \ i \ * \ z \ \$ \ i) +$   
 $x \ * \ z \ \$ \ (dim\text{-}vec \ y)$

**by** *simp*

**finally show** *?thesis* .

**qed**

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

**by** (*induction a arbitrary: b*) (*auto*)

**lemma** *map-mat-map-vec*:

**assumes**  $i < dim\text{-}row \ P$

**shows**  $row \ (map\text{-}mat \ f \ P) \ i = map\text{-}vec \ f \ (row \ P \ i)$

```

using assms by auto

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
  show  $\text{dim-vec } (\text{Matrix.row } (A @_r B) i) = \text{dim-vec } (\text{Matrix.row } A i)$ 
    by (simp add: append-rows-def)
  fix ia
  assume  $ia < \text{dim-vec } (\text{row } A i)$ 
  have  $\text{row } (A @_r B) i = (\text{row } A i @_v \text{row } (0_m (\text{dim-row } A) 0) i)$ 
    unfolding append-rows-def using
      carrier-mat-triv[of A] row-four-block-mat(1)[of A dim-row A
      -  $0_m (\text{dim-row } A) 0 0 B \text{dim-row } B 0_m (\text{dim-row } B) 0 i$ , OF - - - - assms(1)]
    by (metis assms(2) carrier-mat-triv zero-carrier-mat)
  also have  $\dots = \text{row } A i @_v vNil$ 
    by (simp add: assms(1))
  also have  $\dots = \text{row } A i$ 
    by auto
  finally show  $\text{row } (A @_r B) i \$ ia = \text{row } A i \$ ia$ 
    by auto
qed

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
  show  $\text{dim-vec } (\text{row } (A @_r B) i) = \text{dim-vec } (\text{row } B (i - \text{dim-row } A))$ 
    by (simp add: append-rows-def assms(3))
  fix ia
  assume  $ia < \text{dim-vec } (\text{row } B (i - \text{dim-row } A))$ 
  have  $\text{row } (A @_r B) i = (\text{row } B (i - \text{dim-row } A) @_v \text{row } (0_m (\text{dim-row } B) 0) (i - \text{dim-row } A))$ 
    unfolding append-rows-def using carrier-mat-triv[of A] row-four-block-mat(2)[of
    A dim-row A
    -  $0_m (\text{dim-row } A) 0 0 B \text{dim-row } B 0_m (\text{dim-row } B) 0 i$ , OF - - - -
    assms(2)]
    by (metis assms(1) assms(3) carrier-mat-triv le-antisym less-imp-le-nat nat-less-le zero-carrier-mat)
  also have  $\dots = \text{row } B (i - \text{dim-row } A) @_v vNil$ 
    by fastforce
  also have  $\dots = \text{row } B (i - \text{dim-row } A)$ 
    by auto
  finally show  $\text{row } (A @_r B) i \$ ia = \text{row } B (i - \text{dim-row } A) \$ ia$ 
    by auto
qed

```

**lemma** *append-singleton-access* [*simp*]:  $(Matrix.vec\ n\ f\ @_v\ [r]_v)\ \$\ n = r$   
**by** *simp*

Move to right place

**fun** *mat-append-col* **where**  
*mat-append-col*  $A\ b = mat-of-cols\ (dim-row\ A)\ (cols\ A\ @\ [b])$

**fun** *mat-append-row* **where**  
*mat-append-row*  $A\ c = mat-of-rows\ (dim-col\ A)\ (rows\ A\ @\ [c])$

**lemma** *mat-append-col-dims*:  
**shows** *mat-append-col*  $A\ b \in carrier-mat\ (dim-row\ A)\ (dim-col\ A + 1)$   
**by** *auto*

**lemma** *mat-append-row-dims*:  
**shows** *mat-append-row*  $A\ c \in carrier-mat\ (dim-row\ A + 1)\ (dim-col\ A)$   
**by** *auto*

**lemma** *mat-append-col-col*:  
**assumes**  $dim-row\ A = dim-vec\ b$   
**shows**  $col\ (mat-append-col\ A\ b)\ (dim-col\ A) = b$   
**proof** (*standard*)  
**let**  $?nA = (mat-of-cols\ (dim-row\ A)\ (cols\ A\ @\ [b]))$   
**show**  $dim-vec\ (col\ (mat-append-col\ A\ b)\ (dim-col\ A)) = dim-vec\ b$   
**by** (*simp add: assms*)  
**fix**  $i$   
**assume**  $i < dim-vec\ b$   
**have**  $col\ (mat-append-col\ A\ b)\ (dim-col\ A)\ \$\ i = vec-index\ (vec\ (dim-row\ ?nA)\ (\lambda\ i.\ ?nA\ \$\$ (i,\ (dim-col\ A))))\ i$   
**by** (*simp add: col-def*)  
**also have**  $\dots = vec-index\ (vec\ (dim-row\ A)\ (\lambda\ i.\ ?nA\ \$\$ (i,\ (dim-col\ A))))\ i$   
**by** *auto*  
**also have**  $\dots = vec-index\ ((cols\ A\ @\ [b])\ !\ dim-col\ A)\ i$   
**by** (*simp add: <i < dim-vec b> assms mat-of-cols-index*)  
**also have**  $\dots = vec-index\ b\ i$   
**by** (*metis cols-length nth-append-length*)  
**finally show**  $col\ (mat-append-col\ A\ b)\ (dim-col\ A)\ \$\ i = b\ \$\ i$ .  
**qed**

**lemma** *mat-append-col-vec-index*:  
**assumes**  $i < dim-row\ A$   
**and**  $dim-row\ A = dim-vec\ b$   
**shows**  $(row\ (mat-append-col\ A\ b)\ i)\ \$\ (dim-col\ A) = b\ \$\ i$   
**using** *mat-append-col-col*  
**by** (*metis (no-types, lifting) One-nat-def add-Suc-right assms(1) assms(2) carrier-matD(2)*)  
*col-def dim-row-mat(1) index-row(1) index-vec lessI mat-append-col.simps*



*mat-append-col-dims mat-of-cols-def semiring-norm(51)*

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

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

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

**proof**

**let**  $?nA = (\text{mat-of-rows } (\dim\text{-col } A) \ (\text{Matrix.rows } A \ @ \ [c]))$

**show**  $\dim\text{-vec } (\text{Matrix.row } (\text{mat-append-row } A \ c) \ (\dim\text{-row } A)) = \dim\text{-vec } c$

**using** *assms* **by** *simp*

**fix**  $i$  **assume**  $i < \dim\text{-vec } c$

**from** *mat-append-row.simps[of A c]*

**have**  $\text{row } (\text{mat-append-row } A \ c) \ (\dim\text{-row } A) \ \$ \ i = \text{vec-index } (\text{row } ?nA \ (\dim\text{-row } A)) \ i$

**by** *auto*

**also have**  $\dots = \text{vec-index } (\text{vec } (\dim\text{-col } ?nA) \ (\lambda \ j. \ ?nA \ \$ \$ \ (\dim\text{-row } A, j))) \ i$

**by** (*simp add: Matrix.row-def*)

**also have**  $\dots = \text{vec-index } ((\text{rows } A \ @ \ [c]) \ ! \ \dim\text{-row } A) \ i$

**by** (*metis (mono-tags, lifting) ‹mat-append-row A c = mat-of-rows (dim-col A) (Matrix.rows A @ [c])›*)

*add-Suc-right append-Nil2 assms calculation carrier-matD(1) col-transpose cols-transpose*

*index-transpose-mat(2) index-transpose-mat(3) length-append length-rows lessI list.size(3)*

*mat-append-col.elims mat-append-col-col mat-append-row-dims nth-append-length*

*transpose-mat-of-rows One-nat-def*

**also have**  $\dots = \text{vec-index } c \ i$

**by** (*metis length-rows nth-append-length*)

**finally show**  $\text{Matrix.row } (\text{mat-append-row } A \ c) \ (\dim\text{-row } A) \ \$ \ i = c \ \$ \ i .$

**qed**

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

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

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

**by** (*auto*) (*metis assms le-imp-less-Suc length-append-singleton*)

*length-rows mat-of-rows-row nat-less-le nth-append nth-rows row-carrier*)

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

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

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

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

**by** (*metis One-nat-def add.right-neutral add-Suc-right assms(1) assms(2) carrier-matD(1)*)

*carrier-matD(2) index-col index-row(1) lessI mat-append-row-dims mat-append-row-row*)

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

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

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

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

**shows**  $(\text{row } (\text{mat-append-col } A \ b) \ i) \ \$ \ j = (\text{row } A \ i) \ \$ \ j$   
**using** *Matrix.row-transpose*[of  $j \ A$ , *OF assms*(3)]  
*Matrix.transpose-transpose*[of  $(\text{mat-append-col } A \ b)$ ] *assms carrier-matD*(1)  
*carrier-matD*(2) *cols-length cols-transpose index-col index-row*(1)[of  $i \ \text{mat-append-col}$   
 $A \ b \ j$ ] *index-transpose-mat*(2)  
*mat-append-col.simps mat-append-col-dims*  
*mat-of-cols-carrier*(3) *mat-of-rows-row*  
*nth-append nth-rows row-carrier trans-less-add1 transpose-mat-of-cols*  
*mat-of-cols-index*  
**by** (*smt cols-nth index-row*(1))

**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**  
**show** 1:  $\text{dim-vec } (\text{Matrix.row } (\text{mat-append-col } A \ b) \ i) = \text{dim-vec } (\text{Matrix.row } A \ i \ @_v \ [b \ \$ \ i]_v)$   
**by** *simp*  
**fix**  $ia$   
**assume**  $a: ia < \text{dim-vec } (\text{Matrix.row } A \ i \ @_v \ [b \ \$ \ i]_v)$   
**consider**  $ia = \text{dim-col } A \mid ia < \text{dim-col } A$   
**using** *a less-SucE* **by** *auto*  
**then show**  $\text{row } (\text{mat-append-col } A \ b) \ i \ \$ \ ia = (\text{Matrix.row } A \ i \ @_v \ [b \ \$ \ i]_v) \ \$ \ ia$   
**proof** (*cases*)  
**case** 1  
**then show** *?thesis*  
**using** *mat-append-col-vec-index*[of  $i \ A \ b$ , *OF assms*] **by** *auto*  
**next**  
**case** 2  
**have**  $\text{row } (\text{mat-append-col } A \ b) \ i \ \$ \ ia = (\text{mat-append-col } A \ b) \ \$ \ (i, ia)$   
**using** *a assms*(1) **by** *auto*  
**then show** *?thesis using mat-append-col-access-in-mat*[of  $A \ b \ i \ ia$ , *OF assms*(2)  
*assms*(1) 2]  
**using** 2 **by** *auto*  
**qed**  
**qed**

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

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

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

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

```

unfolding one-element-vec-def by (auto)

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

lemma single-nz-val-carrier: single-nz-val n i v  $\in$  carrier-vec n
by auto

lemma single-nz-val-access1 [simp]:  $i < n \implies single-nz-val\ n\ i\ v\ \$\ i = v$ 
by auto

lemma single-nz-val-access2 [simp]:  $i < n \implies j < n \implies i \neq j \implies single-nz-val\ n\ i\ v\ \$\ j = 0$ 
by (auto)

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

lemma single-nz-val-unit-vec:
fixes  $v::'a::\{monoid-mult,times,zero-neq-one,mult-zero\}$ 
shows  $v \cdot_v (unit-vec\ n\ i) = single-nz-val\ n\ i\ v$ 
proof
show  $*$ :  $dim-vec\ (v \cdot_v\ unit-vec\ n\ i) = dim-vec\ (single-nz-val\ n\ i\ v)$ 
by (simp)
fix ia
assume  $ia < dim-vec\ (single-nz-val\ n\ i\ v)$ 
then show  $(v \cdot_v\ unit-vec\ n\ i)\ \$\ ia = single-nz-val\ n\ i\ v\ \$\ ia$ 
using  $*$  by (simp add: unit-vec-def)
qed

lemma single-nz-valI [intro]:
fixes v i val
assumes  $\bigwedge j. j < dim-vec\ v \implies j \neq i \implies v\$j = 0$ 
assumes  $v\$i = val$ 
shows  $v = single-nz-val\ (dim-vec\ v)\ i\ val$ 
using assms(1) assms(2) by auto

lemma single-nz-val-dotP:
assumes  $i < n$ 
assumes  $dim-vec\ x = n$ 
shows  $single-nz-val\ n\ i\ v \cdot x = v * x \$ i$ 
proof –
let  $?y = single-nz-val\ n\ i\ v$ 
have  $single-nz-val\ n\ i\ v \cdot x = (\sum\ i \in \{0 ..< dim-vec\ x\}. ?y\ \$\ i * x\ \$\ i)$ 
unfolding scalar-prod-def by auto
also have  $\dots = (\sum\ i \in \{0 ..< dim-vec\ x\} - \{i\}. ?y\ \$\ i * x\ \$\ i) + ?y\ \$\ i * x\ \$\ i$ 
by (metis (no-types, lifting) add.commute assms(1) assms(2) atLeast0LessThan
finite-atLeastLessThan lessThan-iff sum.remove)

```

**also have** ... =  $(\sum_{i \in \{0 \dots \dim\text{-vec } x\} - \{i\}} ?y \$ i * x \$ i) + v * x \$ i$   
**by** (*simp add: assms(1)*)  
**also have** ... =  $v * x \$ i$   
**proof** –  
**have**  $\bigwedge j. j \in \{0 \dots \dim\text{-vec } x\} - \{i\} \implies ?y \$ j * x \$ j = 0$   
**by** (*simp add: assms(2)*)  
**then have**  $(\sum_{i \in \{0 \dots \dim\text{-vec } x\} - \{i\}} ?y \$ i * x \$ i) = 0$  **by** *auto*  
**then show** *?thesis* **by** *auto*  
**qed**  
**finally show** *?thesis* .  
**qed**

**lemma** *single-nz-zero-singleton: single-nz-val 1 0 v = [v]<sub>v</sub>*  
**by** (*auto*)

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

**assumes**  $\dim\text{-vec } u = m$   
**and**  $\dim\text{-vec } x = n$   
**shows**  $(\text{one-element-vec } n \ e \ @_v \ (0_v \ m)) \cdot (x \ @_v \ u) = (\sum_{i \in \{0 \dots \dim\text{-vec } x\}} e * x \$ i)$   
**proof** –  
**let**  $?OEV = \text{one-element-vec } n \ e$   
**have**  $\dim\text{-vec } (?OEV \ @_v \ (0_v \ m)) = \dim\text{-vec } (x \ @_v \ u)$   
**by** (*simp add: assms(1) assms(2) one-element-vec-carrier*)  
**have**  $(\text{one-element-vec } n \ e \ @_v \ 0_v \ m) \cdot (x \ @_v \ u) = \text{one-element-vec } n \ e \cdot x + 0_v \ m \cdot u$   
**using** *scalar-prod-append[of ?OEV - 0\_v m - x u] assms*  
**by** (*meson carrier-vec-dim-vec one-element-vec-carrier zero-carrier-vec*)  
**also have** ... =  $(\sum_{i \in \{0 \dots \dim\text{-vec } x\}} ?OEV \$ i * x \$ i) + (\sum_{i \in \{0 \dots \dim\text{-vec } u\}} (0_v \ m) \$ i * u \$ i)$   
**unfolding** *scalar-prod-def* **by** *blast*  
**also have** ... =  $(\sum_{i \in \{0 \dots \dim\text{-vec } x\}} ?OEV \$ i * x \$ i)$   
**using** *assms(1)* **by** *auto*  
**also have** ... =  $(\sum_{i \in \{0 \dots \dim\text{-vec } x\}} e * x \$ i)$   
**using** *assms(2)* **by** *auto*  
**finally show** *?thesis* .  
**qed**

**lemma** *one-element-vec-dotP:*

**assumes**  $\dim\text{-vec } x = n$   
**shows**  $(\text{one-element-vec } n \ e) \cdot x = (\sum_{i \in \{0 \dots \dim\text{-vec } x\}} e * x \$ i)$   
**by** (*metis (no-types, lifting) assms one-element-vec-access scalar-prod-def sum.ivl-cong*)

**lemma** *singleton-dotP [simp]: dim-vec x = 1  $\implies$  [v]<sub>v</sub> · x = v \* x\$0*

**by** (*metis dim-vec index-vec less-one single-nz-valI single-nz-val-dotP*)

**lemma** *singletons-dotP [simp]: [v]<sub>v</sub> · [w]<sub>v</sub> = v \* w*

**by** (*metis dim-vec index-vec less-one singleton-dotP*)

**lemma** *singleton-appends-dotP* [*simp*]:  $\dim\text{-vec } x = \dim\text{-vec } y \implies (x @_v [v]_v) \cdot (y @_v [w]_v) = x \cdot y + v * w$   
**using** *scalar-prod-append*[of *x dim-vec x [v]\_v 1 y [w]\_v*]  
**by** (*metis carrier-dim-vec singletons-dotP vec-carrier*)

**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*:  $(v \{X\}) = (\sum x \in \text{vars } v. \text{coeff } v \ x * X \ x)$   
**by** (*auto simp: linear-poly-sum intro: sum.cong*)

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

**assumes** *finite S*

**assumes**  $S \supseteq \text{vars } v$

**shows**  $(v \{X\}) = (\sum x \in S. \text{coeff } v \ x * X \ x)$

**proof** –

**define** *D* **where** *D*:  $D = S - \text{vars } v$

**have** *zeros*:  $\forall x \in D. \text{coeff } v \ x = 0$

**using** *D coeff-zero* **by** *auto*

**have** *fnt*: *finite* (*vars v*)

**using** *finite-vars* **by** *auto*

**have**  $(v \{X\}) = (\sum x \in \text{vars } v. \text{coeff } v \ x * X \ x)$

**by** (*auto simp add: linear-poly-sum intro: sum.cong*)

**also have**  $\dots = (\sum x \in \text{vars } v. \text{coeff } v \ x * X \ x) + (\sum x \in D. \text{coeff } v \ x * X \ x)$

**using** *zeros* **by** *auto*

**also have**  $\dots = (\sum x \in \text{vars } v \cup D. \text{coeff } v \ x * X \ x)$

**using** *assms(1) fnt Diff-partition*[of *vars v S, OF assms(2)*]

*sum.subset-diff*[of *vars v S, OF assms(2) assms(1)*]

**by** (*simp add:*  $\langle \bigwedge g. \text{sum } g \ S = \text{sum } g \ (S - \text{vars } v) + \text{sum } g \ (\text{vars } v) \rangle D$ )

**also have**  $\dots = (\sum x \in S. \text{coeff } v \ x * X \ x)$

**using** *D Diff-partition assms(2)* **by** *fastforce*

**finally show** *?thesis* .

**qed**

Get rid of these synonyms

## 4 Translations of Jordan Normal Forms Matrix Library to Simplex polynomials

### 4.1 Vectors

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

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

**lemma** *empty-list-0poly*:

**shows**  $\text{list-to-lpoly } [] = 0$

**unfolding** *list-to-lpoly-def* by *simp*

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

**assumes**  $i \geq \text{length } L$

**shows**  $\text{coeff } (\text{list-to-lpoly } L) i = 0$

**using** *assms* by (*induction*  $L$  rule: *rev-induct*) (*auto simp: list-to-lpoly-def*)

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

**assumes**  $i < \text{length } cs$

**assumes**  $cs \neq []$

**shows**  $\text{coeff } (\text{list-to-lpoly } cs) i = cs!i$

**using** *assms(2)* *assms(1)*

**proof** (*induction*  $cs$  rule: *rev-nonempty-induct*)

**fix**  $x :: \text{rat}$

**assume**  $i < \text{length } [x]$

**have**  $(\text{list-to-lpoly } [x]) = \text{lp-monom } x 0$

**by** (*simp add: list-to-lpoly-def*)

**then show**  $\text{coeff } (\text{list-to-lpoly } [x]) i = [x] ! i$

**using**  $\langle i < \text{length } [x] \rangle$  *list-to-lpoly-def* by *auto*

**next**

**fix**  $x :: \text{rat}$

**fix**  $xs :: \text{rat list}$

**assume**  $xs \neq []$

**assume** *IH*:  $i < \text{length } xs \implies \text{coeff } (\text{list-to-lpoly } xs) i = xs ! i$

**assume**  $i < \text{length } (xs @ [x])$

**consider**  $(le) i < \text{length } xs \mid (eq) i = \text{length } xs$

**using**  $\langle i < \text{length } (xs @ [x]) \rangle$  *less-Suc-eq* by *auto*

**then show**  $\text{coeff } (\text{list-to-lpoly } (xs @ [x])) i = (xs @ [x]) ! i$

**proof** (*cases*)

**case** *le*

**have**  $\text{coeff } (\text{lp-monom } x (\text{length } xs)) i = 0$

**using** *le* by *auto*

**have**  $\text{coeff } (\text{sum-list } (\text{map2 } (\lambda x y. \text{lp-monom } y x)$

$[0..<\text{length } (xs @ [x])] (xs @ [x]))) i = (xs @ [x]) ! i$

**apply** (*simp add: IH le nth-append*)

**using** *IH le list-to-lpoly-def* by *auto*

**then show** *?thesis*

**unfolding** *list-to-lpoly-def* by *simp*

```

next
  case eq
  then have *: coeff (sum-list (map2 ( $\lambda x y. \text{lp-monom } y \ x$ ) [ $0..<\text{length } xs$ ] xs))
i = 0
    using sum-list-map-upto-coeff-limit[of xs i]
    by (simp add: list-to-lpoly-def)
    have **: (sum-list (map2 ( $\lambda x y. \text{lp-monom } y \ x$ ) [ $0..<\text{length } (xs \ @ \ [x])$ ] (xs @
[x]))) =
      sum-list (map ( $\lambda(x,y). \text{lp-monom } y \ x$ ) (zip [ $0..<\text{length } xs$ ] xs)) + lp-monom
x (length xs)
    by simp
    have coeff ((list-to-lpoly xs) + lp-monom x (length xs)) i = x
    unfolding list-to-lpoly-def using * ** by (simp add: eq)
    then show ?thesis
    by (simp add: eq list-to-lpoly-def)
qed
qed

```

```

lemma list-to-lpoly-coeff-nth:
  assumes i < length cs
  shows coeff (list-to-lpoly cs) i = cs ! i
  using gr-implies-not0 rl-lpoly-coeff-nth-non-empty assms by fastforce

```

```

lemma rat-list-outside-zero:
  assumes length cs ≤ i
  shows coeff (list-to-lpoly cs) i = 0
  using sum-list-map-upto-coeff-limit[of cs i, OF assms] by simp

```

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 ∈ 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 ≥ dim-poly p
  shows coeff p x = 0
  using Max-ge[of vars p x, OF finite-vars[of p]] coeff-zero[of p x]
  by (metis add-cancel-left-right assms dim-poly.elims empty-iff leD le-eq-less-or-eq

      trans-less-add1 zero-neq-one-class.zero-neq-one)

```

```

lemma lpoly-to-vec-0-iff-zero-poly [iff]:
  shows (lpoly-to-vec p) = 0v 0 ↔ p = 0

```

```

proof(standard)
  show lpoly-to-vec  $p = 0_v$   $0 \implies p = 0$ 
  proof (rule contrapos-pp)
    assume  $p \neq 0$ 
    then have vars  $p \neq \{\}$ 
      by (simp add: vars-empty-zero)
    then have dim-poly  $p > 0$ 
      by (simp)
    then show lpoly-to-vec  $p \neq 0_v$   $0$ 
      using vec-of-dim-0[of lpoly-to-vec p] by simp
  qed
next
qed (auto simp: vars-empty-zero)

lemma dim-poly-dim-vec-equiv:
  dim-vec (lpoly-to-vec  $p$ ) = dim-poly  $p$ 
  using lpoly-to-vec.simps by auto

lemma dim-poly-greater-ex-coeff: dim-poly  $x > d \implies \exists i \geq d. \text{coeff } x \ i \neq 0$ 
  by (simp split: if-splits) (meson Max-in coeff-zero finite-vars less-Suc-eq-le)

lemma dimpoly-all-zero-limit:
  assumes  $\bigwedge i. i \geq d \implies \text{coeff } x \ i = 0$ 
  shows dim-poly  $x \leq d$ 
proof -
  have  $(\forall i \geq d. \text{coeff } x \ i = 0) \implies \text{dim-poly } x \leq d$ 
  proof (rule contrapos-pp)
    assume  $\neg \text{dim-poly } x \leq d$ 
    then have dim-poly  $x > d$  by linarith
    then have  $\exists i \geq d. \text{coeff } x \ i \neq 0$ 
      using dim-poly-greater-ex-coeff[of d x] by blast
    then show  $\neg (\forall i \geq d. \text{coeff } x \ i = 0)$ 
      by blast
  qed
then show ?thesis
  using assms by blast
qed

lemma construct-poly-from-lower-dim-poly:
  assumes dim-poly  $x = d+1$ 
  obtains  $p \ c$  where dim-poly  $p \leq d$   $x = p + \text{lp-monom } c \ d$ 
proof -
  define  $c'$  where  $c': c' = \text{coeff } x \ d$ 
  have  $f: \forall i > d. \text{coeff } x \ i = 0$ 
    using assms by auto
  have  $*$ :  $x = x - (\text{lp-monom } c' \ d) + (\text{lp-monom } c' \ d)$ 
    by simp
  have coeff  $(x - (\text{lp-monom } c' \ d)) \ d = 0$ 
    using  $c'$  by simp

```



**then have**  $\forall i \geq d. \text{coeff } (x - (\text{lp-monom } c' d)) i = 0$   
**using** *f* **by** *auto*  
**then have** \*\*:  $\text{dim-poly } (x - (\text{lp-monom } c' d)) \leq d$   
**using** *dimpoly-all-zero-limit*[of  $d (x - (\text{lp-monom } c' d))$ ] **by** *auto*  
**define** *p'* **where** *p'*:  $p' = x - (\text{lp-monom } c' d)$   
**have**  $\exists p c. \text{dim-poly } p \leq d \wedge x = p + \text{lp-monom } c d$   
**using** \* \*\* **by** *blast*  
**then show** *?thesis*  
**using** \* *p' c'* **that** **by** *blast*  
**qed**

**lemma** *vars-subset-0-dim-poly*:  
 $\text{vars } z \subseteq \{0..<\text{dim-poly } z\}$   
**by** (*simp add: finite-vars less-Suc-eq-le subsetI*)

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

**lemma** *valuate-with-dim-poly*:  $z \llbracket X \rrbracket = (\sum i \in \{0..<\text{dim-poly } z\}. \text{coeff } z i * X i)$   
**using** *eval-poly-with-sum-superset*[of  $\{0..<\text{dim-poly } z\} z X$ ] **using** *vars-subset-0-dim-poly*  
**by** *blast*

**lemma** *lin-poly-to-vec-coeff-access*:  
**assumes**  $x < \text{dim-poly } y$   
**shows** (*lpoly-to-vec* *y*) \$  $x = \text{coeff } y x$   
**proof** –  
**have**  $x < \text{dim-vec } (\text{lpoly-to-vec } y)$   
**using** *dim-poly-dim-vec-equiv*[of *y*] *assms* **by** *auto*  
**then show** *?thesis*  
**by** (*simp add: coeff-def*)  
**qed**

**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** (*lpoly-to-vec*  $x + \text{lpoly-to-vec } y$ ) \$  $a = \text{coeff } (x + y) a$   
**using** *assms(1) assms(2) lin-poly-to-vec-coeff-access* **by** (*simp add: dim-poly-dim-vec-equiv*)

**lemma** *list-to-lpoly-dim-less*:  $\text{length } cs \geq \text{dim-poly } (\text{list-to-lpoly } cs)$   
**using** *dimpoly-all-zero-limit sum-list-map-upto-coeff-limit* **by** *blast*

Transform rational vectors to linear polynomials

**fun** *vec-to-lpoly* **where**  
 $\text{vec-to-lpoly } rv = \text{list-to-lpoly } (\text{list-of-vec } rv)$

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

by (simp add: assms list-to-lpoly-coeff-nth)

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

fixes  $x\ y$

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

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

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

using *assms(1) assms(2) coeff-plus index-add-vec(1)*

by (metis *vec-to-lin-poly-coeff-access*)

**lemma** *outside-list-coeff0*:

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

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

by (simp add: *assms sum-list-map-upto-coeff-limit*)

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

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

using *list-to-lpoly-dim-less[of list-of-vec x]* by simp

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

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

by (metis *all-greater-dim-poly-zero dim-poly-dim-vec-equiv lin-poly-to-vec-coeff-access not-less outside-list-coeff0 vec-to-lin-poly-coeff-access*)

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

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

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

by (metis *assms dim-poly-dim-vec-equiv less-le-trans lin-poly-to-vec-coeff-access vec-to-lin-poly-coeff-access vec-to-poly-dim-less*)

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

by (meson *ivl-subset le-numeral-extra(3) order.trans vec-to-poly-dim-less vars-subset-0-dim-poly*)

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

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

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

**proof** –

consider (eq)  $\dim\text{-vec}\ A = \dim\text{-poly}\ (\text{vec-to-lpoly}\ A) \mid$

(le)  $\dim\text{-vec}\ A > \dim\text{-poly}\ (\text{vec-to-lpoly}\ A)$

using *vec-to-poly-dim-less[of A]* by fastforce

then show ?thesis

**proof** (cases)

case le

define  $dp$  where  $dp = \dim\text{-poly}\ (\text{vec-to-lpoly}\ A)$

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

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

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

by (metis (no-types, lifting) *dp vec-to-poly-dim-less sum.atLeastLessThan-concat*)

```

zero-le)
  also have ... = ( $\sum a = 0..<dp.$  coeff (vec-to-lpoly A) a * X a)
  using all-greater-dim-poly-zero by (simp add: dp)
  also have ... = ( $\sum a = 0..<dim-poly$  (vec-to-lpoly A).coeff (vec-to-lpoly A) a *
X a)
  using dp by auto
  finally show ?thesis
  by blast
qed (auto)
qed

```

```

lemma vec-to-lpoly-vNil [simp]: vec-to-lpoly vNil = 0
  by (simp add: empty-list-0poly)

```

```

lemma zero-vector-is-zero-poly: coeff (vec-to-lpoly (0_v n)) i = 0
  by (metis index-zero-vec(1) index-zero-vec(2) not-less
outside-list-coeff0 vec-to-lin-poly-coeff-access)

```

```

lemma coeff-nonzero-dim-vec-non-zero:
  assumes coeff (vec-to-lpoly v) i  $\neq$  0
  shows v $ i  $\neq$  0 i < dim-vec v
  apply (metis assms leI outside-list-coeff0 vec-to-lin-poly-coeff-access)
  using assms leI outside-list-coeff0 by blast

```

```

lemma lpoly-of-v-equals-v-append0:
  vec-to-lpoly v = vec-to-lpoly (v @_v 0_v a) (is ?lhs = ?rhs)
proof -
  have  $\forall i.$  coeff ?lhs i = coeff ?rhs i
  proof
    fix i
    consider (le) i < dim-vec v | (ge) i  $\geq$  dim-vec v
    using leI by blast
    then show coeff (vec-to-lpoly v) i = coeff (vec-to-lpoly (v @_v 0_v a)) i
    proof (cases)
      case le
      then show ?thesis using vec-to-lin-poly-coeff-access[of i v] index-append-vec(1)
        by (metis index-append-vec(2) vec-to-lin-poly-coeff-access trans-less-add1)
    next
      case ge
      then have coeff (vec-to-lpoly v) i = 0
        using outside-list-coeff0 by blast
      moreover have coeff (vec-to-lpoly (v @_v 0_v a)) i = 0
      proof (rule ccontr)
        assume na:  $\neg$  coeff (vec-to-lpoly (v @_v 0_v a)) i = 0
        define va where v: va = coeff (vec-to-lpoly (v @_v 0_v a)) i
        have i < dim-vec (v @_v 0_v a)
          using coeff-nonzero-dim-vec-non-zero[of (v @_v 0_v a) i] na by blast
        moreover have (0_v a) $ (i - dim-vec v) = va
          by (metis ge diff-is-0-eq' index-append-vec(1) index-append-vec(2))
      qed
    qed
  qed

```

$\text{not-less-zero } \text{vec-to-lin-poly-coeff-access } v \text{ zero-less-diff calculation}$   
**moreover have**  $va \neq 0$  **using**  $v \text{ na}$  **by** *linarith*  
**ultimately show** *False*  
**using**  $ge$  **by** *auto*  
**qed**  
**then show**  $\text{coeff } (\text{vec-to-lpoly } v) \ i = \text{coeff } (\text{vec-to-lpoly } (v @_v 0_v \ a)) \ i$   
**using** *not-less* **using** *calculation* **by** *linarith*  
**qed**  
**qed**  
**then show** *?thesis*  
**using** *Abstract-Linear-Poly.poly-eqI* **by** *blast*  
**qed**

**lemma** *vec-to-lpoly-eval-dot-prod*:  
 $(\text{vec-to-lpoly } v) \ \{ x \} = v \cdot (\text{vec } (\text{dim-vec } v) \ x)$   
**proof** –  
**have**  $(\text{vec-to-lpoly } v) \ \{ x \} = (\sum_{i \in \{0..<\text{dim-vec } v\}} \text{coeff } (\text{vec-to-lpoly } v) \ i * x \ i)$   
**using** *eval-poly-with-sum-superset*[of  $\{0..<\text{dim-vec } v\}$  *vec-to-lpoly } v x]*  
*vars-subset-dim-vec-to-lpoly-dim* **by** *blast*  
**also have**  $\dots = (\sum_{i \in \{0..<\text{dim-vec } v\}} v \$ i * x \ i)$   
**using** *list-to-lpoly-coeff-nth* **by** *auto*  
**also have**  $\dots = v \cdot (\text{vec } (\text{dim-vec } v) \ x)$   
**unfolding** *scalar-prod-def* **by** *auto*  
**finally show** *?thesis* .  
**qed**

**lemma** *dim-poly-of-append-vec*:  
 $\text{dim-poly } (\text{vec-to-lpoly } (a @_v b)) \leq \text{dim-vec } a + \text{dim-vec } b$   
**using** *vec-to-poly-dim-less*[of  $a @_v b$ ] *index-append-vec(2)*[of  $a \ b$ ] **by** *auto*

**lemma** *vec-coeff-append1*:  $i \in \{0..<\text{dim-vec } a\} \implies \text{coeff } (\text{vec-to-lpoly } (a @_v b)) \ i = a \$ i$   
**by** (*metis atLeastLessThan-iff index-append-vec(1) index-append-vec(2) vec-to-lin-poly-coeff-access trans-less-add1*)

**lemma** *vec-coeff-append2*:  
 $i \in \{\text{dim-vec } a..<\text{dim-vec } (a @_v b)\} \implies \text{coeff } (\text{vec-to-lpoly } (a @_v b)) \ i = b \$ (i - \text{dim-vec } a)$   
**by** (*metis atLeastLessThan-iff index-append-vec(1) index-append-vec(2) leD vec-to-lin-poly-coeff-access*)

Maybe Code Equation

**lemma** *vec-to-lpoly-poly-of-vec-eq*:  $\text{vec-to-lpoly } v = \text{poly-of-vec } v$   
**proof** –  
**have**  $\bigwedge i. i < \text{dim-vec } v \implies \text{coeff } (\text{poly-of-vec } v) \ i = v \ \$ \ i$   
**by** (*simp add: coeff.rep-eq poly-of-vec.rep-eq*)  
**moreover have**  $\bigwedge i. i < \text{dim-vec } v \implies \text{coeff } (\text{vec-to-lpoly } v) \ i = v \ \$ \ i$   
**by** (*simp add: vec-to-lin-poly-coeff-access*)  
**moreover have**  $\bigwedge i. i \geq \text{dim-vec } v \implies \text{coeff } (\text{poly-of-vec } v) \ i = 0$

by (*simp add: coeff.rep-eq poly-of-vec.rep-eq*)  
**moreover have**  $\bigwedge i. i \geq \dim\text{-vec } v \implies \text{coeff } (\text{vec-to-lpoly } v) i = 0$   
 using *outside-list-coeff0* **by blast**  
**ultimately show** *?thesis*  
 by (*metis Abstract-Linear-Poly.poly-eq-iff le-less-linear*)  
**qed**

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

**proof** –

let *?p* = (*vec-to-lpoly* (*0<sub>v</sub>* *n* *@<sub>v</sub>* *v*))  
 have *dim-poly* *?p*  $\leq n + \dim\text{-vec } v$   
 using *dim-poly-of-append-vec*[of *0<sub>v</sub>* *n* *v*] **by auto**  
 have  $\text{vars } (\text{vec-to-lpoly } (0_v \ n \ @_v \ v)) \subseteq \{0..<n+\dim\text{-vec } v\}$   
 using *vars-subset-dim-vec-to-lpoly-dim*[of (*0<sub>v</sub>* *n* *@<sub>v</sub>* *v*)] **by auto**  
**moreover have**  $\forall i < n. \text{coeff } ?p \ i = 0$   
 using *vec-coeff-append1*[of - *0<sub>v</sub>* *n* *v*] **by auto**  
**ultimately show**  $\text{vars } (\text{vec-to-lpoly } (0_v \ n \ @_v \ v)) \subseteq \{n..<n+\dim\text{-vec } v\}$   
 by (*meson atLeastLessThan-iff coeff-zero not-le subsetCE subsetI*)  
**qed**

## 5 Matrices

**fun** *matrix-to-lpolies* **where**

*matrix-to-lpolies* *A* = *map vec-to-lpoly* (*rows* *A*)

**lemma** *matrix-to-lpolies-vec-of-row*:

$i < \dim\text{-row } A \implies \text{matrix-to-lpolies } A ! i = \text{vec-to-lpoly } (\text{row } A \ i)$   
 using *matrix-to-lpolies.simps*[of *A*] **by simp**

**lemma** *outside-of-col-range-is-0*:

**assumes**  $i < \dim\text{-row } A$  **and**  $j \geq \dim\text{-col } A$   
**shows**  $\text{coeff } ((\text{matrix-to-lpolies } A) ! i) \ j = 0$   
 using *outside-list-coeff0*[of *col* *A* *i* *j*]  
**by** (*metis assms(1) assms(2) index-row(2) length-rows matrix-to-lpolies.simps nth-map nth-rows outside-list-coeff0*)

**lemma** *polys-greater-col-zero*:

**assumes**  $x \in \text{set } (\text{matrix-to-lpolies } A)$   
**assumes**  $j \geq \dim\text{-col } A$   
**shows**  $\text{coeff } x \ j = 0$   
**using** *assms(1) assms(2) outside-of-col-range-is-0*[of - *A* *j*]  
*assms(2) matrix-to-lpolies.simps* **by** (*metis in-set-conv-nth length-map length-rows*)

**lemma** *matrix-to-lp-vec-to-lpoly-row* [*simp*]:

**assumes**  $i < \dim\text{-row } A$   
**shows**  $(\text{matrix-to-lpolies } A) ! i = \text{vec-to-lpoly } (\text{row } A \ i)$   
**by** (*simp add: assms*)

**lemma** *matrix-to-lpolies-coeff-access*:  
**assumes**  $i < \dim\text{-row } A$  **and**  $j < \dim\text{-col } A$   
**shows**  $\text{coeff } (\text{matrix-to-lpolies } A ! i) j = A \text{ \$\$ } (i,j)$   
**using** *matrix-to-lp-vec-to-lpoly-row*[of  $i A$ , *OF* *assms(1)*]  
**by** (*metis* *assms(1)* *assms(2)* *index-row(1)* *index-row(2)* *vec-to-lin-poly-coeff-access*)

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**  $\text{coeff } (L ! i) j = (\text{lin-polies-to-mat } L) \text{ \$\$ } (i,j)$   
**unfolding** *lin-polies-to-mat-def* **by** (*simp* *add: assms(1)* *assms(2)*)

**lemma** *vec-to-lpoly-valuate-equiv-dot-prod*:  
**assumes**  $\dim\text{-vec } y = \dim\text{-vec } x$   
**shows**  $(\text{vec-to-lpoly } y) \text{ \}\ (\$)x \text{ \}} = y \cdot x$   
**proof** –  
**let**  $?p = \text{vec-to-lpoly } y$   
**have**  $2: ?p \text{ \}\ (\$)x \text{ \}} = (\sum_{j \in \text{vars } ?p} \text{coeff } ?p j * x \$j)$   
**using** *eval-poly-with-sum*[of  $?p (\$)x$ ] **by** *blast*  
**have**  $\text{vars } ?p \subseteq \{0..<\dim\text{-vec } y\}$   
**using** *vars-subset-dim-vec-to-lpoly-dim* **by** *blast*  
**have**  $2: ?p \text{ \}\ (\$)x \text{ \}} = (\sum_{j \in \text{vars } ?p} \text{coeff } ?p j * x \$j)$   
**using** *eval-poly-with-sum*[of  $?p (\$)x$ ] **by** *blast*  
**also have**  $*$ :  $\dots = (\sum_{i \in \{0..<\dim\text{-poly } ?p\}} \text{coeff } ?p i * x \$i)$   
**using** *valuate-with-dim-poly* **by** (*metis* (*no-types*, *lifting*) *calculation sum.cong*)  
**also have**  $\dots = y \cdot x$   
**proof** –  
**have**  $\bigwedge j. j < \dim\text{-vec } x \implies \text{coeff } (\text{vec-to-lpoly } y) j = y \$ j$   
**using** *assms* *vec-to-lin-poly-coeff-access* **by** *auto*  
**then show** *?thesis*  
**using** *vec-to-lpoly-eval-dot-prod*[of  $y (\$)x$ ]  
**by** (*metis* *assms* *calculation* *dim-vec* *index-vec* *vec-eq-iff*)  
**qed**  
**finally show** *?thesis* **unfolding** *scalar-prod-def* .  
**qed**

**lemma** *matrix-to-lpolies-valuate-scalarP*:  
**assumes**  $i < \dim\text{-row } A$   
**assumes**  $\dim\text{-col } A = \dim\text{-vec } x$   
**shows**  $(\text{matrix-to-lpolies } A ! i) \text{ \}\ (\$)x \text{ \}} = (\text{row } A i) \cdot x$   
**using** *vec-to-lpoly-valuate-equiv-dot-prod*[of *row*  $A i x$ ]  
**by** (*simp* *add: assms(1)* *assms(2)*)

```

lemma matrix-to-lpolies-lambda-valuate-scalarP:
  assumes  $i < \text{dim-row } A$ 
  assumes  $\text{dim-col } A = \text{dim-vec } x$ 
shows  $(\text{matrix-to-lpolies } A!i) \ \{\!| \ (\lambda i. \ (\text{if } i < \text{dim-vec } x \text{ then } x\$i \text{ else } 0)) \ \}\!| = (\text{row } A \ i) \cdot x$ 
proof –
  have  $\bigwedge j. j < \text{dim-vec } x \implies x\$j = (\lambda i. \ (\text{if } i < \text{dim-vec } x \text{ then } x\$i \text{ else } 0)) \ j$ 
    by simp
  let  $?p = (\text{matrix-to-lpolies } A!i)$ 
  have  $\bigwedge j. \text{coeff } (\text{matrix-to-lpolies } A!i) \ j \neq 0 \implies j < \text{dim-vec } x$ 
    using outside-of-col-range-is-0[of i A] assms(1) assms(2) leI by auto
  then have  $\text{subs: vars } ?p \subseteq \{0..<\text{dim-vec } x\}$ 
    using  $\langle \bigwedge j. \text{Abstract-Linear-Poly.coeff } (\text{matrix-to-lpolies } A \ ! \ i) \ j \neq 0 \implies j < \text{dim-vec } x \rangle$  atLeastLessThan-iff coeff-zero by blast
  then have  $*$ :  $\bigwedge j. j \in \text{vars } ?p \implies x\$j = (\lambda i. \ (\text{if } i < \text{dim-vec } x \text{ then } x\$i \text{ else } 0)) \ j$ 
    by (simp add:  $\langle \bigwedge j. \text{Abstract-Linear-Poly.coeff } (\text{matrix-to-lpolies } A \ ! \ i) \ j \neq 0 \implies j < \text{dim-vec } x \rangle$  coeff-zero)
  have  $\text{row } A \ i \cdot x = (?p \ \{\!| \ (\$) \ x \ \}\!|)$ 
    using assms(1) assms(2) matrix-to-lpolies-valuate-scalarP[of i A x] by linarith
  also have  $\dots = (\sum_{j \in \text{vars } ?p} \text{coeff } ?p \ j * x\$j)$ 
    using eval-poly-with-sum by blast
  also have  $\dots = (\sum_{j \in \text{vars } ?p} \text{coeff } ?p \ j * (\lambda i. \ (\text{if } i < \text{dim-vec } x \text{ then } x\$i \text{ else } 0)) \ j)$ 
    by (metis (full-types, opaque-lifting)  $\langle \bigwedge j. \text{Abstract-Linear-Poly.coeff } (\text{matrix-to-lpolies } A \ ! \ i) \ j \neq 0 \implies j < \text{dim-vec } x \rangle$  mult commute mult-zero-right)
  also have  $\dots = (?p \ \{\!| \ (\lambda i. \ (\text{if } i < \text{dim-vec } x \text{ then } x\$i \text{ else } 0)) \ \}\!|)$ 
    using eval-poly-with-sum by presburger
  finally show ?thesis
    by linarith
qed

```

```

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  $index \ b = [\text{GEQ } (\text{lp-monom } 1 \ (i+\text{index})) \ (b\$i). \ i \leftarrow [0..<\text{dim-vec } b]]$ 

```

**lemma** *constraints-set-vars-geq-vec-def*:  
 $set (vars-from-index-geq-vec \ start \ b) =$   
 $\{GEQ (lp-monom \ 1 \ (i+start)) (b\$i) \mid i. i \in \{0..<dim-vec \ b\}\}$   
**using** *set-comprehension-list-comprehension*[of  
 $(\lambda i. GEQ (lp-monom \ 1 \ (i+start)) (b\$i)) \ dim-vec \ b]$  **by** *auto*

**lemma** *vars-from-index-geq-sat*:  
**assumes**  $\langle x \rangle \models_{cs} set (vars-from-index-geq-vec \ start \ b)$   
**assumes**  $i < dim-vec \ b$   
**shows**  $\langle x \rangle (i+start) \geq b\$i$   
**proof** –  
**have**  $e-e: GEQ (lp-monom \ 1 \ (i+start)) (b\$i) \in set (vars-from-index-geq-vec \ start \ b)$   
**using** *constraints-set-vars-geq-vec-def*[of *start* *b*] **using** *assms(2)* **by** *auto*  
**then** **have**  $\langle x \rangle \models_c GEQ (lp-monom \ 1 \ (i+start)) (b\$i)$   
**using** *assms(1)* **by** *blast*  
**then** **have**  $(lp-monom \ 1 \ (i+start)) \ \{\!\{ \langle x \rangle \!\}\} \geq (b\$i)$   
**using** *satisfies-constraint.simps(4)*[of  $\langle x \rangle \ lp-monom \ 1 \ (i + start) \ b\$i]$   
**by** *simp*  
**then** **show** *?thesis*  
**by** *simp*  
**qed**

**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**  $\langle x \rangle \models_{cs} set (mat-x-leq-vec \ A \ b)$   
**assumes**  $i < dim-vec \ b$   
**shows**  $((matrix-to-lpolies \ A)!i) \ \{\!\{ \langle x \rangle \!\}\} \leq b\$i$   
**proof** –  
**have**  $e-e: LEQ ((matrix-to-lpolies \ A)!i) (b\$i) \in set (mat-x-leq-vec \ A \ b)$   
**by** (*simp add: assms(2)*)  
**then** **have**  $\langle x \rangle \models_c LEQ ((matrix-to-lpolies \ A)!i) (b\$i)$   
**using** *assms(1)* **by** *blast*  
**then** **show** *?thesis*  
**using** *satisfies-constraint.simps* **by** *auto*  
**qed**

**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**  $x \models_{cs} \text{set } (x\text{-mat-eq-vec } b \ A)$   
**assumes**  $i < \text{dim-vec } b$   
**shows**  $((\text{matrix-to-lpolies } A)!i) \{ x \} = b\$i$   
**proof** –  
**have**  $e\text{-e}: EQ ((\text{matrix-to-lpolies } A)!i) (b\$i) \in \text{set } (x\text{-mat-eq-vec } b \ A)$   
**by** (*simp add: assms(2)*)  
**then have**  $x \models_c EQ ((\text{matrix-to-lpolies } A)!i) (b\$i)$   
**using** *assms(1)* **by** *blast*  
**then show** *?thesis*  
**using** *satisfies-constraint.simps* **by** *auto*  
**qed**

## 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*  $A \ B$ ) (*dim-row*  $A$ ) (*dim-col*  $A$ ) =  
 $(Q1, Q2, Q3, Q4)$   
**shows**  $Q1 = A \ Q4 = B$   
**using** *split-four-block-dual-fst-1st*[*of*  $A \ - \ B \ Q1 \ Q2 \ Q3 \ Q4$ ]  
*assms* **by** *auto*

**lemma** *two-block-non-interfering-dims*:  
*dim-row* (*two-block-non-interfering*  $A \ B$ ) = *dim-row*  $A$  + *dim-row*  $B$   
*dim-col* (*two-block-non-interfering*  $A \ B$ ) = *dim-col*  $A$  + *dim-col*  $B$   
**by** (*simp*)+

**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*  $A \ B)\$(i,j) = 0$   
**using** *four-block-mat-def* *assms* *two-block-non-interfering-dims*[*of*  $A \ B$ ] **by** *auto*

**lemma** *two-block-non-interfering-row-comp1*:  
**assumes**  $i < \text{dim-row } A$   
**shows** *row* (*two-block-non-interfering*  $A \ B$ )  $i = \text{row } A \ i \ @_v (0_v (\text{dim-col } B))$   
**using** *assms* **by** *auto*

**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**  $\text{row } (\text{two-block-non-interfering } A \ B) \ i = (0_v \ (\text{dim-col } A)) \ @_v \ \text{row } B \ (i - \text{dim-row } A)$   
**using** *assms* **by** (*auto*)

**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**  $\text{vec-first } (\text{two-block-non-interfering } A \ B \ *_v \ v) \ n = A \ *_v \ (\text{vec-first } v \ (\text{dim-col } A))$

**proof**

**fix**  $i$

**assume**  $a: i < \text{dim-vec } (A \ *_v \ \text{vec-first } v \ (\text{dim-col } A))$

**let**  $?tb = \text{two-block-non-interfering } A \ B$

**have**  $i-n: i < n$  **using**  $a$  *assms*(2) **by** *auto*

**have**  $\text{vec-first } (?tb \ *_v \ v) \ n \ \$ \ i = \text{vec-first } (\text{vec } (\text{dim-row } ?tb) \ (\lambda \ i. \ \text{row } ?tb \ i \cdot v)) \ n \ \$ \ i$

**unfolding** *mult-mat-vec-def* **by** *simp*

**also have**  $\dots = (\text{vec } n \ (\lambda \ i. \ \text{row } ?tb \ i \cdot v)) \ \$ \ i$

**unfolding** *vec-first-def* **using** *trans-less-add1*

**by** (*metis*  $a$  *assms*(2) *dim-mult-mat-vec index-vec two-block-non-interfering-dims*(1))

**also have**  $\dots = \text{row } ?tb \ i \cdot v$  **by** (*simp add: i-n*)

**also have**  $\dots = (\text{row } A \ i \ @_v \ 0_v \ (\text{dim-col } B)) \cdot v$

**using** *assms*(2)  $i-n$  *two-block-non-interfering-row-comp1* **by** *fastforce*

**also have**  $\dots = \text{row } A \ i \cdot \text{vec-first } v \ (\text{dim-vec } (\text{row } A \ i)) +$

$0_v \ (\text{dim-col } B) \cdot \text{vec-last } v \ (\text{dim-vec } (0_v \ (\text{dim-col } B)))$

**using** *append-split-vec-distrib-scalar-prod*[*of row A i 0\_v (dim-col B) v*] *assms*(1)

**by** *auto*

**then have**  $\text{vec-first } (\text{two-block-non-interfering } A \ B \ *_v \ v) \ n \ \$ \ i =$

$\text{row } A \ i \cdot \text{vec-first } v \ (\text{dim-vec } (\text{row } A \ i))$

**using** *calculation* **by** *auto*

**then show**  $\text{vec-first } (\text{two-block-non-interfering } A \ B \ *_v \ v) \ n \ \$ \ i =$

$(A \ *_v \ \text{vec-first } v \ (\text{dim-col } A)) \ \$ \ i$

**by** (*simp add: assms*(2)  $i-n$ )

**next**

**have**  $\text{dim-vec } (A \ *_v \ v) = \text{dim-row } A$  **using** *dim-vec-def dim-mult-mat-vec*[*of A v*] **by** *auto*

**then have**  $\text{dim-vec } (\text{vec-first } (\text{two-block-non-interfering } A \ B \ *_v \ v) \ n) = n$  **by** *auto*

**then show**  $\text{dim-vec } (\text{vec-first } (\text{two-block-non-interfering } A \ B \ *_v \ v) \ n) =$

$\text{dim-vec } (A \ *_v \ \text{vec-first } v \ (\text{dim-col } A))$

**by** (*simp add: assms*(2))

**qed**

**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**  $\text{vec-last } ((\text{two-block-non-interfering } A \ B) \ *_v \ v) \ n = B \ *_v \ (\text{vec-last } v \ (\text{dim-col } B))$

```

proof
  fix  $i$ 
  assume  $a: i < \text{dim-vec } (B *_v \text{vec-last } v \text{ (dim-col } B))$ 
  let  $?tb = \text{two-block-non-interfering } A \ B$ 
  let  $?vl = (\text{vec } (\text{dim-row } ?tb) (\lambda i. \text{row } ?tb \ i \cdot v))$ 
  have  $i-n: i < n$  using  $\text{assms}(2)$  using  $a$  by  $\text{auto}$ 
  have  $\text{in}\beta: (\text{dim-row } ?tb) - n + i \geq \text{dim-row } A$ 
    by  $(\text{simp add: assms}(2))$ 
  have  $\text{in}\beta': (\text{dim-row } ?tb) - n + i < \text{dim-row } ?tb$ 
    by  $(\text{simp add: assms}(2) \ i-n \ \text{two-block-non-interfering-dims}(1))$ 
  have  $\text{dim-row } A + n = \text{dim-row } (\text{two-block-non-interfering } A \ B)$ 
    by  $(\text{simp add: assms}(2) \ \text{two-block-non-interfering-dims}(1))$ 
  then have  $\text{dim-a}: \text{dim-row } A = \text{dim-row } (\text{two-block-non-interfering } A \ B) - n$ 
    by  $(\text{metis } (\text{no-types}) \ \text{diff-add-inverse}2)$ 
  have  $\text{vec-last } (?tb *_v v) \ n \ \$ \ i = \text{vec-last } (\text{vec } (\text{dim-row } ?tb) (\lambda i. \text{row } ?tb \ i \cdot v))$ 
 $n \ \$ \ i$ 
    unfolding  $\text{mult-mat-vec-def}$  by  $\text{auto}$ 
  also have  $\dots = ?vl \ \$ \ (\text{dim-vec } ?vl - n + i)$ 
    unfolding  $\text{vec-last-def}$  using  $i-n \ \text{index-vec}$  by  $\text{blast}$ 
  also have  $\dots = \text{row } ?tb \ ((\text{dim-row } ?tb) - n + i) \cdot v$ 
    unfolding  $\text{index-vec}$  by  $(\text{simp add: assms}(2) \ i-n \ \text{two-block-non-interfering-dims}(1))$ 
  also have  $\dots = \text{row } B \ i \cdot \text{vec-last } v \ (\text{dim-vec } (\text{row } B \ i))$ 
proof  $-$ 
  have  $\text{dim-vec } (0_v \ (\text{dim-col } A) \ @_v \ \text{row } B \ i) = \text{dim-vec } v$ 
    by  $(\text{simp add: } \langle \text{dim-col } A + \text{dim-col } B = \text{dim-vec } v \rangle)$ 
  then show  $?thesis$  using  $\text{dim-a} \ \text{assms}(1) \ \text{in}\beta' \ \text{two-block-non-interfering-row-comp}2$ 
     $\text{append-split-vec-distrib-scalar-prod}[\text{of } 0_v \ (\text{dim-col } A) \ \text{row } B \ i \ v]$ 
    by  $(\text{metis } \text{add.commute} \ \text{add.right-neutral} \ \text{diff-add-inverse}$ 
     $\ \text{in}\beta \ \text{index-zero-vec}(2) \ \text{scalar-prod-left-zero} \ \text{vec-first-carrier})$ 
qed
  also have  $\dots = \text{row } B \ i \cdot \text{vec-last } v \ (\text{dim-col } B)$  by  $\text{simp}$ 
  thus  $\text{vec-last } (\text{two-block-non-interfering } A \ B *_v v) \ n \ \$ \ i = (B *_v \text{vec-last } v$ 
 $(\text{dim-col } B)) \ \$ \ i$ 
    using  $\text{assms}(2) \ \text{calculation } i-n$  by  $\text{auto}$ 
qed  $(\text{simp add: assms}(2))$ 

```

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

```

assumes  $\text{dim-col } A + \text{dim-col } B = \text{dim-vec } v$ 
shows  $\text{two-block-non-interfering } A \ B *_v v =$ 
 $A *_v \text{vec-first } v \ (\text{dim-col } A) \ @_v \ B *_v \text{vec-last } v \ (\text{dim-col } B)$ 

```

**proof**  $-$

```

let  $?tb = \text{two-block-non-interfering } A \ B$ 
from  $\text{first-vec-two-block-non-inter-is-first-vec}[\text{of } A \ B \ v \ \text{dim-row } A, \ \text{OF } \text{assms}]$ 
have  $\text{vec-first } (?tb *_v v) \ (\text{dim-row } A) = A *_v \text{vec-first } v \ (\text{dim-col } A)$ 
  by  $\text{blast}$ 
moreover from  $\text{last-vec-two-block-non-inter-is-last-vec}[\text{of } A \ B \ v \ \text{dim-row } B, \ \text{OF } \text{assms}]$ 
have  $\text{vec-last } (?tb *_v v) \ (\text{dim-row } B) = B *_v \text{vec-last } v \ (\text{dim-col } B)$ 
  by  $\text{blast}$ 

```

**ultimately show** *?thesis* **using** *vec-first-last-append*[of *?tb \*<sub>v</sub> v (dim-row A)* (*dim-row B*)]  
*dim-mult-mat-vec*[of *?tb v*] *two-block-non-interfering-dims(1)*[of *A B*]  
**by** (*metis carrier-vec-dim-vec*)  
**qed**

**fun** *mat-leqb-eqc* **where**  
*mat-leqb-eqc A b c* = (*let lst* = *matrix-to-lpolies (two-block-non-interfering A A<sup>T</sup>)* *in*  
 $[LEQ (lst!i) (b\$i) . i <- [0..<dim-vec b]] @$   
 $[EQ (lst!i) ((b@_v c)\$i) . i <- [dim-vec b ..< dim-vec (b@_v c)]]]$ )

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

**assumes**  $i < dim-vec b$

**assumes**  $i < dim-row A$

**shows** (*mat-leqb-eqc A b c*)!i = *LEQ ((matrix-to-lpolies A)!i) (b\\$i)*

**proof** –

**define** *lst* **where** *lst*: *lst* = (*mat-leqb-eqc A b c*)

**define** *l* **where** *l*: *l* = *matrix-to-lpolies (two-block-non-interfering A A<sup>T</sup>)*

**have** *ileqA*:  $i < dim-row A$  **using** *assms* **by** *auto*

**have** *li* = *vec-to-lpoly ((row A i)@<sub>v</sub> 0<sub>v</sub> (dim-row A))*

**unfolding** *l* **using** *two-block-non-interfering-row-comp1*[of *i A A<sup>T</sup>, OF ileqA*]

**by** (*metis ileqA lpoly-of-v-equals-v-append0 matrix-to-lp-vec-to-lpoly-row trans-less-add1 two-block-non-interfering-dims(1)*)

**then have** *leq*: *li* = (*matrix-to-lpolies A*)!i

**using** *lpoly-of-v-equals-v-append0*[of *row A i (dim-row A)*] *l*

**by** (*simp add: ileqA*)

**have** \*: *lst* =  $[LEQ (li) (b\$i) . i <- [0..<dim-vec b]] @$

$[EQ (li) ((b@_v c)\$i) . i <- [dim-vec b ..< dim-vec (b@_v c)]]$

**unfolding** *l lst* **by** (*metis mat-leqb-eqc.simps*)

**have** ( $[LEQ (li) (b\$i) . i <- [0..<dim-vec b]] @$

$[EQ (li) ((b@_v c)\$i) . i <- [dim-vec b ..< dim-vec (b@_v c)]]$ ) ! i =

$[LEQ (li) (b\$i) . i <- [0..<dim-vec b]]!i$

**using** *assms(2)* *lst* **by** (*simp add: assms(1) nth-append*)

**also have** ... = *LEQ (li) (b\\$i)*

**using** *l lst*

**by** (*simp add: assms(1)*)

**finally show** *?thesis*

**using** \* *leq lst* **using** *mat-leqb-eqc.simps*[of *A b c*] **by** *auto*

**qed**

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

**assumes**  $dim-vec b \leq i$  **and**  $i < dim-vec (b@_v c)$

**assumes**  $dim-row A = dim-vec b$  **and**  $dim-col A \geq dim-vec c$

**shows** (*mat-leqb-eqc A b c*)!i =

*EQ (vec-to-lpoly (0<sub>v</sub> (dim-col A) @<sub>v</sub> row A<sup>T</sup> (i-dim-vec b))) (c\$(i-dim-vec b))*

**proof** –

```

define lst where lst: lst = (mat-leqb-ecq A b c)
define l where l: l = matrix-to-lpolies (two-block-non-interfering A AT)
have i-s: i < dim-row (two-block-non-interfering A AT)
using assms by (simp add: assms(2) assms(3) two-block-non-interfering-dims(1))
have l'!: l'i = vec-to-lpoly ((0v (dim-col A)) @v (row AT (i – dim-vec b)))
using l two-block-non-interfering-row-comp2[of i A AT, OF i-s]
assms(1) assms(3) i-s matrix-to-lp-vec-to-lpoly-row by presburger
have ([LEQ (l'i) (b$i) . i <– [0..dim-vec b]]) @
[EQ (l'i) ((b@vc)$i) . i <– [dim-vec b..dim-vec (b@vc)]])!i
=
[EQ (l'i) ((b@vc)$i) . i <– [dim-vec b..dim-vec (b@vc)]]) ! (i – dim-vec
by (simp add: assms(1) leD nth-append)
also have ... = EQ (l'i) ((b@vc)$i)
using assms(1) assms(2) by auto
also have ... = EQ (l'i) (c$(i–dim-vec b))
using assms(1) assms(2) by auto
then show ?thesis
using mat-leqb-ecq.simps by (metis (full-types) calculation l l')
qed

```

**lemma** *mat-leqb-ecq-satisfies1*:

```

assumes x ⊢cs set (mat-leqb-ecq A b c)
assumes i < dim-vec b
and i < dim-row A
shows (matrix-to-lpolies A!i) {x} ≤ b$i
proof –
have e-e: LEQ (matrix-to-lpolies A ! i) (b$i) ∈ set (mat-leqb-ecq A b c)
using mat-leqb-ecq-for-LEQ[of i b A c, OF assms(2) assms(3)]
nth-mem[of i matrix-to-lpolies A] mat-leqb-ecq.simps
by (metis (no-types, lifting) assms(2) diff-zero in-set-conv-nth length-append
length-map
length-upt trans-less-add1)
then have x ⊢c LEQ ((matrix-to-lpolies A)!i) (b$i)
using assms by blast
then show ?thesis
using satisfies-constraint.simps by auto
qed

```

**lemma** *mat-leqb-ecq-satisfies2*:

```

assumes x ⊢cs set (mat-leqb-ecq A b c)
assumes dim-vec b ≤ i and i < dim-vec (b@vc)
and dim-row A = dim-vec b and dim-vec c ≤ dim-col A
shows (matrix-to-lpolies (two-block-non-interfering A AT) ! i) {x} = (b @v c) $
i
proof –
have e-e: EQ (vec-to-lpoly (0v (dim-col A) @v row AT (i – dim-vec b))) (c $ (i
– dim-vec b))

```

```

      ∈ set (mat-leqb-eqc A b c)
    using assms(2) mat-leqb-eqc.simps[of A b c]
      nth-mem[of i (mat-leqb-eqc A b c)]
    using mat-leqb-eqc-for-EQ[of b i c A, OF assms(2) assms(3) assms(4) assms(5)]
    by (metis (mono-tags, lifting) add-diff-cancel-left' assms(3) diff-zero index-append-vec(2)

      length-append length-map length-upt)
  hence sateq:  $x \models_c EQ (vec\text{-to-lpoly } (0_v (dim\text{-col } A) @_v \text{ row } A^T (i - dim\text{-vec } b))) (c \$ (i - dim\text{-vec } b)))$ 
    using assms(1) by blast
  have *:  $i < dim\text{-row } (two\text{-block-non-interfering } A A^T)$ 
    by (metis assms(3) assms(4) assms(5) dual-order.order-iff-strict dual-order.strict-trans

      index-append-vec(2) index-transpose-mat(2) nat-add-left-cancel-less
      two-block-non-interfering-dims(1))
  have **:  $dim\text{-row } A \leq i$ 
    by (simp add: assms(2) assms(4))
  then have  $x \models_c EQ ((matrix\text{-to-lpolies } (two\text{-block-non-interfering } A A^T))!i)$ 
    (( $b @_v c$ )$ $i$ )
    using two-block-non-interfering-row-comp2[of i A AT, OF * **]
    by (metis * sateq assms(3) assms(4) index-append-vec(1) index-append-vec(2)
      leD
      matrix-to-lp-vec-to-lpoly-row)
  then show ?thesis
    using satisfies-constraint.simps(5) by simp
qed

lemma mat-leqb-eqc-simplex-satisfies2:
  assumes simplex (mat-leqb-eqc A b c) = Sat x
  assumes dim-vec b ≤ i and i < dim-vec (b@vc)
  and dim-row A = dim-vec b and dim-vec c ≤ dim-col A
  shows (matrix-to-lpolies (two-block-non-interfering A AT) ! i)  $\{\langle x \rangle\} = (b @_v c)$ 
  $ i
  using mat-leqb-eqc-satisfies2 assms(1) assms(2) assms(3) assms(4) assms(5)
  simplex(3) by blast

fun index-geq-n where
  index-geq-n i n = GEQ (lp-monom 1 i) n

lemma index-geq-n-simplex:
  assumes  $\langle x \rangle \models_c (index\text{-geq-n } i n)$ 
  shows  $\langle x \rangle i \geq n$ 
  using assms by simp

fun from-index-geq0-vector where
  from-index-geq0-vector i v = [GEQ (lp-monom 1 (i+j)) (v$j) . j <- [0..<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**  $x \ (i + j) \geq v\$j$

**proof** –

**have**  $GEQ \ (\text{lp-monom } 1 \ (i+j)) \ (v\$j) \in \text{set } (\text{from-index-geq0-vector } i \ v)$   
**by** (*simp add: assms(2)*)  
**moreover have**  $x \models_c \ GEQ \ (\text{lp-monom } 1 \ (i+j)) \ (v\$j)$   
**using** *calculation(1) assms* **by force**  
**ultimately show** *?thesis* **by simp**

**qed**

**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**  $\langle x \rangle \ j \geq v\$(j - i)$   
**by** (*metis assms(1) assms(2) assms(3) from-index-geq-vector-simplex  
le-add-diff-inverse less-diff-conv2*)

**definition** *x-times-c-geq-y-times-b where*

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

**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**  $((\text{vec-to-lpoly } (c \ @_v \ 0_v \ (\text{dim-vec } b))) \ \{\langle x \rangle\}) \geq$   
 $((\text{vec-to-lpoly } (0_v \ (\text{dim-vec } c) \ @_v \ b)) \ \{\langle x \rangle\})$   
**using** *simplex(3)[OF assms]* **unfolding** *x-times-c-geq-y-times-b-def*  
**by** (*simp add: valuate-minus*)

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

**lemma** *split-vec-dims*:

**assumes** *split-i-j-x*  $i \ j \ x = (a, b)$   
**shows**  $\text{dim-vec } a = i \ \text{dim-vec } b = (j - i)$   
**using** *assms(1) unfolding split-i-j-x-def* **by auto+**

**lemma** *split-n-m-x-abbrev-dims*:  
**assumes** *split-n-m-x*  $n$   $m$   $x = (a, b)$   
**shows** *dim-vec*  $a = n$  *dim-vec*  $b = m$   
**using** *split-vec-dims*  
**using** *assms* **apply** *blast*  
**using** *assms* *split-vec-dims(2)* **by** *fastforce*

**lemma** *split-access-fst-1*:  
**assumes**  $k < i$   
**assumes** *split-i-j-x*  $i$   $j$   $x = (a, b)$   
**shows**  $a \$ k = \langle x \rangle k$   
**by** (*metis* *Pair-inject* *assms(1)* *assms(2)* *index-vec* *split-i-j-x-def*)

**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** –  
**have** *vec*  $(j - i)$   $(\lambda n. \langle x \rangle (n + i)) = b$   
**by** (*metis* (*no-types*) *assms(3)* *prod.sel(2)* *split-i-j-x-def*)  
**then show** *?thesis*  
**using** *assms(1)* *assms(2)* **by** *fastforce*  
**qed**

**lemma** *split-access-fst-2*:  
**assumes**  $(x, y) = \text{split-i-j-x } i \ j \ Z$   
**assumes**  $k < \text{dim-vec } x$   
**shows**  $x \$ k = \langle Z \rangle k$   
**by** (*metis* *assms(1)* *assms(2)* *split-access-fst-1* *split-vec-dims(1)*)

**lemma** *split-access-snd-2*:  
**assumes**  $(x, y) = \text{split-i-j-x } i \ j \ Z$   
**assumes**  $k < \text{dim-vec } y$   
**shows**  $y \$ k = \langle Z \rangle (k + \text{dim-vec } x)$   
**using** *assms* *split-i-j-x-def*[*of*  $i \ j \ Z$ ] **by** *auto*

**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**  $\bigwedge i. i < \text{dim-vec } v \implies i < m \implies y \$ i \geq v \$ i$   
**using** *assms* **unfolding** *split-i-j-x-def*  
**using** *from-index-geq-vector-simplex*[*of*  $d \ v \ \langle X \rangle$  -] *index-vec* **by** (*simp* *add: add commute*)

**lemma** *split-coeff-vec-index-sum*:  
**assumes**  $(x, y) = \text{split-i-j-x } (\text{dim-vec } (\text{lpoly-to-vec } v)) \ l \ X$   
**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** –  
**from** *valuate-with-dim-poly*[*of*  $v \ \langle X \rangle$ , *symmetric*]



**have**  $(\sum i = 0..<dim-vec\ x.\ (lpoly-to-vec\ v)\ \$\ i\ *\ \langle X \rangle\ i) =$   
 $(\sum i = 0..<dim-vec\ x.\ (lpoly-to-vec\ v)\ \$\ i\ *\ x\ \$\ i)$   
**by**  $(metis\ (no-types,\ lifting)\ assms\ split-access-fst-1\ split-vec-dims(1)\ sum.ivl-cong)$   
**then show** *?thesis*  
**by**  $(metis\ (no-types,\ lifting)\ assms\ dim-poly-dim-vec-equiv$   
 $lin-poly-to-vec-coeff-access\ split-vec-dims(1)\ sum.ivl-cong)$   
**qed**

**lemma** *scalar-prod-valuation-after-split-equiv1*:  
**assumes**  $(x,y) = split-i-j-x\ (dim-vec\ (lpoly-to-vec\ v))\ l\ X$   
**shows**  $(lpoly-to-vec\ v) \cdot x = (v\ \{\langle X \rangle\})$   
**proof** –  
**from** *valuate-with-dim-poly[of v <X>, symmetric]*  
**have**  $1: (v\ \{\langle X \rangle\}) = (\sum i = 0..<dim-poly\ v.\ Abstract-Linear-Poly.coeff\ v\ i\ *\$   
 $\langle X \rangle\ i)$  **by** *simp*  
**have**  $(\sum i = 0..<dim-vec\ x.\ (lpoly-to-vec\ v)\ \$\ i\ *\ \langle X \rangle\ i) =$   
 $(\sum i = 0..<dim-vec\ x.\ (lpoly-to-vec\ v)\ \$\ i\ *\ x\ \$\ i)$   
**by**  $(metis\ (no-types,\ lifting)\ assms\ split-access-fst-1\ split-vec-dims(1)\ sum.ivl-cong)$   
**also have**  $\dots = (lpoly-to-vec\ v) \cdot x$   
**unfolding** *scalar-prod-def* **by** *blast*  
**finally show** *?thesis*  
**by**  $(metis\ (no-types,\ lifting)\ 1\ dim-poly-dim-vec-equiv\ lin-poly-to-vec-coeff-access$   
 $split-vec-dims(1)\ sum.ivl-cong\ assms)$   
**qed**

**definition** *mat-times-vec-leq*  $([-*_v]- \leq - [1000,1000,100])$   
**where**  
 $[A\ *_v\ x] \leq b \iff (\forall i < dim-vec\ b.\ (A\ *_v\ x)\ \$\ i \leq b\ \$\ i) \wedge$   
 $(dim-row\ A = dim-vec\ b) \wedge$   
 $(dim-col\ A = dim-vec\ x)$

**definition** *vec-times-mat-eq*  $([-_v*_]- = - [1000,1000,100])$   
**where**  
 $[y\ _v*\ A] = c \iff (\forall i < dim-vec\ c.\ (A^T\ *_v\ y)\ \$\ i = c\ \$\ i) \wedge$   
 $(dim-col\ A^T = dim-vec\ y) \wedge$   
 $(dim-row\ A^T = dim-vec\ c)$

**definition** *vec-times-mat-leq*  $([-_v*_]- \leq - [1000,1000,100])$   
**where**  
 $[y\ _v*\ A] \leq c \iff (\forall i < dim-vec\ c.\ (A^T\ *_v\ y)\ \$\ i \leq c\ \$\ i) \wedge$   
 $(dim-col\ A^T = dim-vec\ y) \wedge$   
 $(dim-row\ A^T = dim-vec\ c)$

**lemma** *mat-times-vec-leqI[intro]*:  
**assumes**  $dim-row\ A = dim-vec\ b$   
**assumes**  $dim-col\ A = dim-vec\ x$   
**assumes**  $\bigwedge i.\ i < dim-vec\ b \implies (A\ *_v\ x)\ \$\ i \leq b\ \$\ i$

**shows**  $[A *_v x] \leq b$   
**unfolding** *mat-times-vec-leq-def* **using** *assms* **by** *auto*

**lemma** *mat-times-vec-leqD[dest]*:  
**assumes**  $[A *_v x] \leq b$   
**shows**  $\dim\text{-row } A = \dim\text{-vec } b \wedge \dim\text{-col } A = \dim\text{-vec } x \wedge i. i < \dim\text{-vec } b \implies (A *_v x)\$i \leq b\$i$   
**using** *assms mat-times-vec-leq-def* **by** *blast+*

**lemma** *vec-times-mat-eqD[dest]*:  
**assumes**  $[y *_v A] = c$   
**shows**  $(\forall i. i < \dim\text{-vec } c. (A^T *_v y)\$i = c\$i) (\dim\text{-col } A^T = \dim\text{-vec } y) (\dim\text{-row } A^T = \dim\text{-vec } c)$   
**using** *assms vec-times-mat-eq-def* **by** *blast+*

**lemma** *vec-times-mat-leqD[dest]*:  
**assumes**  $[y *_v A] \leq c$   
**shows**  $(\forall i. i < \dim\text{-vec } c. (A^T *_v y)\$i \leq c\$i) (\dim\text{-col } A^T = \dim\text{-vec } y) (\dim\text{-row } A^T = \dim\text{-vec } c)$   
**using** *assms vec-times-mat-leq-def* **by** *blast+*

**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**  $[x *_v A] = c$   
**unfolding** *vec-times-mat-eq-def* **using** *assms* **by** *blast*

**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**  $(x, y) = \text{split-}i\text{-}j\text{-}x \ (\dim\text{-col } A) \ l \ X$   
**shows**  $[A *_v x] \leq b$   
**proof** (*standard, goal-cases*)  
**case** 1  
**then show** *?case* **using** *assms(1)[symmetric]* .  
**case** 2  
**then show** *?case* **using** *assms(3)* **unfolding** *split-}i-}j-}x-def*  
**using** *split-vec-dims[of 0 dim-col A X x y]* **by** *auto*  
**case** (3 *i*)  
**with** *mat-leqb-eqc-satisfies1* [of *A b c <X> i*]  
**have** *m*: (*matrix-to-lpolies* *A ! i*)  $\llbracket \langle X \rangle \rrbracket \leq b \$ i$   
**using** *assms(1) assms(2)* **by** *linarith*  
**have** *leq*: *dim-poly* (*vec-to-lpoly* (*row* *A i*))  $\leq \dim\text{-col } A$   
**using** *vec-to-poly-dim-less* [of *row A i*] **by** *simp*  
**have** *i*:  $i < \dim\text{-row } A$   
**using** 3 *assms(1)* **by** *linarith*  
**from** *two-block-non-interfering-row-comp1* [of *i A A^T*]  
**have** *row* (*two-block-non-interfering* *A A^T*)  $i = \text{row } A \ i \ @_v \ 0_v \ (\dim\text{-col } A^T)$

**using**  $3$  *assms(1)* **by** *linarith*  
**have** ( $\text{vec-to-lpoly}$  ( $\text{row } A \ i \ @_v \ 0_v \ (\text{dim-col } A^T)$ ))  $\{\langle X \rangle\} = ((\text{vec-to-lpoly}$  ( $\text{row } A \ i$ ))  $\{\langle X \rangle\})$   
**using** *lpoly-of-v-equals-v-append0* **by** *auto*  
**also have**  $\dots = (\sum a = 0..<\text{dim-poly}$  ( $\text{vec-to-lpoly}$  ( $\text{row } A \ i$ )).  
 $\text{Abstract-Linear-Poly.coeff}$  ( $\text{vec-to-lpoly}$  ( $\text{row } A \ i$ ))  $a * \langle X \rangle \ a)$   
**using** *valuate-with-dim-poly[of vec-to-lpoly (row A i) <X>]* **by** *blast*  
**also have**  $\dots = (\sum a = 0..<\text{dim-col } A. \text{Abstract-Linear-Poly.coeff}$  ( $\text{vec-to-lpoly}$   
( $\text{row } A \ i$ ))  $a * \langle X \rangle \ a)$   
**using** *split-coeff-vec-index-sum[of x y]*  
 $\text{sum-dim-vec-equals-sum-dim-poly}$ [ $\text{of row } A \ i \ \langle X \rangle$ ] **by** *auto*  
**also have**  $\dots = \text{row } A \ i \cdot x$   
**unfolding** *scalar-prod-def* **using**  $\langle \text{dim-col } A = \text{dim-vec } x \rangle \ i$  *assms(3)*  
**using** *matrix-to-lpolies-coeff-access[of i A]* *matrix-to-lp-vec-to-lpoly-row[of i A]*  
 $\text{split-access-fst-1}$ [ $\text{of } - \ (\text{dim-col } A) \ l \ X \ x \ y$ ] **by** *fastforce*  
**finally show** *?case*  
**using**  $m \ i$  *lpoly-of-v-equals-v-append0* **by** *auto*  
**qed**

**lemma** *mat-leqb-eqc-split-simplex-correct1*:  
**assumes**  $\text{dim-vec } b = \text{dim-row } A$   
**assumes**  $\text{simplex} \ (\text{mat-leqb-eqc } A \ b \ c) = \text{Sat } X$   
**assumes**  $(x, y) = \text{split-i-j-x} \ (\text{dim-col } A) \ l \ X$   
**shows**  $[A \ *_v \ x] \leq b$   
**using** *mat-leqb-eqc-split-correct1*[ $\text{of } b \ A \ c \ X \ x \ y$ ] *assms(1)* *assms(2)* *assms(3)*  
*simplex(3)* **by** *blast*

**lemma** *sat-mono*:  
**assumes**  $\text{set } A \subseteq \text{set } B$   
**shows**  $\langle X \rangle \models_{cs} \text{set } B \implies \langle X \rangle \models_{cs} \text{set } A$   
**using** *assms(1)* *assms* **by** *blast*

**lemma** *mat-leqb-eqc-split-subset-correct1*:  
**assumes**  $\text{dim-vec } b = \text{dim-row } A$   
**assumes**  $\text{set} \ (\text{mat-leqb-eqc } A \ b \ c) \subseteq \text{set } S$   
**assumes**  $\text{simplex } S = \text{Sat } X$   
**assumes**  $(x, y) = \text{split-i-j-x} \ (\text{dim-col } A) \ l \ X$   
**shows**  $[A \ *_v \ x] \leq b$   
**using** *sat-mono* *assms(1)* *assms(2)* *assms(3)* *assms(4)*  
*mat-leqb-eqc-split-correct1* *simplex(3)* **by** *blast*

**lemma** *mat-leqb-eqc-split-correct2*:  
**assumes**  $\text{dim-vec } c = \text{dim-row } A^T$   
**assumes**  $\text{dim-vec } b = \text{dim-col } A^T$   
**assumes**  $\langle X \rangle \models_{cs} \text{set} \ (\text{mat-leqb-eqc } A \ b \ c)$   
**assumes**  $(x, y) = \text{split-n-m-x} \ (\text{dim-row } A^T) \ (\text{dim-col } A^T) \ X$   
**shows**  $[y \ *_v \ A] = c$   
**proof** (*standard, goal-cases*)  
**case**  $1$

```

then show ?case
  using assms split-n-m-x-abbrev-dims(2)[OF assms(4)][symmetric] by linarith
case 2
then show ?case using assms(1)[symmetric] .
case (3 i)
define lst where lst: lst = matrix-to-lpolies (two-block-non-interfering A AT)
define db where db: db = dim-vec b
define dc where dc: dc = dim-vec c
have cA: dim-vec c ≤ dim-col A
  by (simp add: assms(1))
have dbi-dim: db+i < dim-vec (b @v c)
  by (simp add: 3 db)
have *: dim-vec b < db+i
  by (simp add: db)
have ([LEQ (lst!i) (b$i) . i <- [0..<dim-vec b]] @
  [EQ (lst!i) ((b@vc)$i) . i <- [dim-vec b ..< dim-vec (b@vc)]]) ! (db + i) =
  EQ (lst!(db+i)) ((b@vc)$ (db+i)) using mat-leqb-egc-for-EQ[of b db+i c A]
  nth-append[of [LEQ (lst!i) (b$i) . i <- [0..<dim-vec b]]
  [EQ (lst!i) ((b@vc)$i) . i <- [dim-vec b ..< dim-vec (b@vc)]])
  by (simp add: 3 db)
have rowA: dim-vec b = dim-row A
  using assms index-transpose-mat(3)[of A] by linarith
have ⟨X⟩ ⊢c EQ (lst!(db+i)) (c$i)
proof -
  have db + i - dim-vec b = i
    using db diff-add-inverse by blast
  then have (lst ! (db + i)) ‖⟨X⟩‖ = c $ i
    by (metis dbi-dim rowA * cA assms(3) index-append-vec(1)
      index-append-vec(2) leD lst mat-leqb-egc-satisfies2)
  then show ?thesis
    using satisfies-constraint.simps(5)[of ⟨X⟩ (lst ! (db + i)) (c $ i)] by simp
qed
then have sat: (lst!(db+i)) ‖⟨X⟩‖ = c$i
  by simp
define V where V: V = vec (db+dc) (λi. ⟨X⟩ i)
have vdim: dim-vec V = dim-vec (b@vc) using V db dc by simp
have *: db + i < dim-row (two-block-non-interfering A AT)
  by (metis dbi-dim assms(1) index-append-vec(2) rowA two-block-non-interfering-dims(1))
have **: dim-row A ≤ db + i
  by (simp add: assms(2) db)
from two-block-non-interfering-row-comp2[of db+i A AT, OF * **]
have eql: row (two-block-non-interfering A AT) (db + i) = 0v (dim-col A) @v
row AT i
  by (simp add: assms(2) db)
with matrix-to-lp-vec-to-lpoly-row[of i AT]
have eqv: lst!(db+i) = vec-to-lpoly (0v (dim-col A) @v row AT i)
  using * lst matrix-to-lp-vec-to-lpoly-row by presburger
then have ∀ j < dim-col A. Abstract-Linear-Poly.coeff (lst!(db+i)) j = 0
  by (metis index-append-vec(1) index-append-vec(2) index-zero-vec(1) index-zero-vec(2))

```

$vec\text{-to-lin-poly-coeff-access trans-less-add1}$   
**moreover have**  $\forall j \geq db+dc. Abstract\text{-Linear-Poly.coeff (lst!(db+i)) j = 0$   
**by** (*metis (mono-tags, lifting) eqv index-transpose-mat(3) index-zero-vec(2) leD*  
 $add.commute assms(1) assms(2) coeff\text{-nonzero-dim-vec-non-zero(2) in-$   
 $dex-append-vec(2)$   
 $index\text{-row}(2) index\text{-transpose-mat}(2) db dc$ )  
**moreover have vars**  $(lst!(db+i)) \subseteq \{dim\text{-col } A.. < db+dc\}$   
**by** (*meson atLeastLessThan-iff calculation(1) calculation(2) coeff-zero not-le*  
 $subsetI$ )  
**ultimately have**  $(lst!(db+i)) \langle \langle X \rangle \rangle = (\sum j \in \{dim\text{-col } A.. < db+dc\}. Abstract\text{-Linear-Poly.coeff$   
 $(lst!(db+i)) j * \langle X \rangle j)$   
**using**  $eval\text{-poly-with-sum-superset[of } \{dim\text{-col } A.. < db+dc\} lst!(db+i) \langle X \rangle]$  **by**  
 $blast$   
**also have**  $... = (\sum j \in \{dim\text{-col } A.. < db+dc\}. Abstract\text{-Linear-Poly.coeff (lst!(db+i))$   
 $j * V\$j)$   
**using**  $V$  **by auto**  
**also have**  $... = (\sum j \in \{dim\text{-col } A.. < db+dc\}. (0_v (dim\text{-col } A) @_v row A^T i) \$j *$   
 $V\$j)$   
**proof** –  
**have**  $\forall j \in \{dim\text{-col } A.. < db+dc\}. Abstract\text{-Linear-Poly.coeff (lst!(db+i)) j = (0_v$   
 $(dim\text{-col } A) @_v row A^T i) \$j$   
**by** (*metis <V ≡ vec (db + dc) <X> vdim assms(1) assms(2) index-transpose-mat(2)*  
 $atLeastLessThan-iff dim-vec eqv index-append-vec(2) index\text{-row}(2)$   
 $vec\text{-to-lin-poly-coeff-access semiring-normalization-rules(24)$   
 $two-block-non-interfering-dims(2)$ )  
**then show** *?thesis*  
**by** (*metis (mono-tags, lifting) sum.cong*)  
**qed**  
**also have**  $... = (\sum j \in \{0.. < dim\text{-col } A\}. (0_v (dim\text{-col } A) @_v row A^T i) \$j * V\$j)$   
 $+$   
 $(\sum j \in \{dim\text{-col } A.. < db+dc\}. (0_v (dim\text{-col } A) @_v row A^T i) \$j * V\$j)$   
**by** (*metis (no-types, lifting) add-cancel-left-left atLeastLessThan-iff mult-eq-0-iff*  
 $class-semiring.add.finprod-all1 index-append-vec(1) index-zero-vec(1)$   
 $index-zero-vec(2) trans-less-add1$ )  
**also have**  $... = (\sum j \in \{0.. < db+dc\}. (0_v (dim\text{-col } A) @_v row A^T i) \$j * V\$j)$   
**by** (*metis (no-types, lifting) add.commute assms(1) dc index-transpose-mat(2)*  
 $le-add1 le-add-same-cancel1 sum.atLeastLessThan-concat$ )  
**also have**  $... = (0_v (dim\text{-col } A) @_v row A^T i) \cdot V$   
**unfolding**  $scalar\text{-prod-def}$  **by** (*simp add: V*)  
**also have**  $... = 0_v (dim\text{-col } A) \cdot vec\text{-first } V (dim\text{-vec } (0_v (dim\text{-col } A))) +$   
 $row A^T i \cdot vec\text{-last } V (dim\text{-vec } (row A^T i))$   
**using**  $append\text{-split-vec-distrib-scalar-prod[of } 0_v (dim\text{-col } A) row A^T i V]$   
**by** (*metis (no-types, lifting) <dim-vec V = dim-vec (b @\_v c)> add.commute*  
 $assms(1)$   
 $assms(2) index-append-vec(2) index\text{-row}(2) index\text{-transpose-mat}(2)$   
 $index\text{-transpose-mat}(3) index-zero-vec(2)$ )  
**also have**  $0_v (dim\text{-col } A) \cdot vec\text{-first } V (dim\text{-vec } (0_v (dim\text{-col } A))) +$

$\text{row } A^T \ i \cdot \text{vec-last } V \ (\text{dim-vec } (\text{row } A^T \ i)) = (\text{row } A^T \ i) \cdot y$

**proof** –  
**have**  $\text{vec-last } V \ (\text{dim-vec } (\text{row } A^T \ i)) = y$   
**proof** (*standard, goal-cases*)  
**case** (1 *i*)  
**then show** ?*case*  
**proof** –  
**have**  $f1: \text{dim-col } A^T = db$   
**by** (*simp add: assms(2) db*)  
**then have**  $\forall v \text{ va. } \text{vec } db \ (\lambda n. \langle X \rangle \ (n + dc)) = v \vee (x, y) \neq (va, v)$   
**by** (*metis Pair-inject add-diff-cancel-left' assms(1) assms(4) dc split-i-j-x-def*)  
**then show** ?*case*  
**unfolding**  $V \ \text{vec-last-def}$   
**using** *split-access-fst-1* [of ( $\text{dim-row } A^T$ ) *i* ( $\text{dim-col } A^T$ )  $X \ x \ y$ ]  
**by** (*metis 1 add.commute add-diff-cancel-left' add-less-cancel-left*  
 $\text{dim-vec } f1 \ \text{index-row}(2) \ \text{index-vec}$ )  
**qed**  
**next**  
**case** 2  
**then show** ?*case*  
**using**  $\langle \text{dim-col } A^T = \text{dim-vec } y \rangle$  **by** *auto*  
**qed**  
**then show** ?*thesis*  
**by** (*simp add: assms(1)*)  
**qed**  
**then show** ?*case* **unfolding** *mult-mat-vec-def* **by** (*metis 3 assms(1) calculation*  
*index-vec sat*)  
**qed**

**lemma** *mat-leqb-eqc-split-simplex-correct2*:  
**assumes**  $\text{dim-vec } c = \text{dim-row } A^T$   
**assumes**  $\text{dim-vec } b = \text{dim-col } A^T$   
**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**  $[y \ v * A] = c$   
**using** *assms(1) assms(2) assms(3) assms(4) mat-leqb-eqc-split-correct2 simplex(3)* **by** *blast*

**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**  $[y \ v * A] = c \ [A \ *_v \ x] \leq b$   
**using** *mat-leqb-eqc-split-simplex-correct1* [of  $b \ A \ c \ X \ x \ y$ ]  
**using** *assms(1) assms(2) assms(3) assms(4) mat-leqb-eqc-split-simplex-correct2*  
**apply** *blast*  
**using** *mat-leqb-eqc-split-correct2* [of  $b \ A \ c \ X \ x \ y$ ]  
**by** (*metis (no-types) Matrix.transpose-transpose assms(2) assms(3) assms(4)*)

*index-transpose-mat*( $\beta$ )  
*mat-leqb-eqc-split-simplex-correct1*[of  $b \ A \ c \ X \ x \ y$ ]

**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** –

**have**  $*$ :  $(\text{vec-to-lpoly } c) \ \{\langle X \rangle\} =$

$(\sum_{i \in \text{vars } (\text{vec-to-lpoly } c)}. \text{Abstract-Linear-Poly.coeff } (\text{vec-to-lpoly } c) \ i \ *$

$\langle X \rangle \ i)$

**using** *linear-poly-sum sum.cong eval-poly-with-sum* **by** *auto*

**also have**  $\dots = (\sum_{i \in \{0..<\text{dim-vec } c\}}. \text{Abstract-Linear-Poly.coeff } (\text{vec-to-lpoly } c) \ i \ * \ \langle X \rangle \ i)$

**using** *vars-subset-dim-vec-to-lpoly-dim*[of  $c$ ] *linear-poly-sum*[of *vec-to-lpoly*  $c$   $\langle X \rangle$ ]

*eval-poly-with-sum-superset*[of  $\{0..<\text{dim-vec } c\}$  *vec-to-lpoly*  $c$   $\langle X \rangle$ ] **by** *auto*

**also have**  $\dots = (\sum_{i \in \{0..<\text{dim-vec } c\}}. c\$i \ * \ x\$i)$

**using** *split-access-fst-1*[of *dim-vec*  $c$  (*dim-vec*  $c$ ) + (*dim-vec*  $b$ )  $X \ x \ y$ ]

*split-access-snd-1*[of *dim-vec*  $c$  - ((*dim-vec*  $c$ ) + (*dim-vec*  $b$ ))  $X \ x \ y$ ]

*vec-to-lin-poly-coeff-access*[of  $c$ ] **using** *assms* **by** *auto*

**also have**  $\dots = c \cdot x$

**unfolding** *scalar-prod-def*

**using** *split-vec-dims(1)*[of *dim-vec*  $c$  (*dim-vec*  $c$ ) + (*dim-vec*  $b$ )  $X \ x \ y$ ] *assms*

**by** *auto*

**finally show** *?thesis* .

**qed**

**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** –

**let**  $?p = (\text{vec-to-lpoly } ((0_v (\text{dim-vec } b) @_v c)))$

**let**  $?v0 = (0_v (\text{dim-vec } b) @_v c)$

**have**  $*$ :  $\forall i < \text{dim-vec } b. \text{Abstract-Linear-Poly.coeff } ?p \ i = 0$

**using** *coeff-nonzero-dim-vec-non-zero(1)* **by** *fastforce*

**have**  $**$ :  $\text{dim-vec } ?v0 = \text{dim-vec } b + \text{dim-vec } c$

**by** *simp*

**have**  $?p \ \{\langle X \rangle\} = (\sum_{i \in \text{vars } ?p}. \text{Abstract-Linear-Poly.coeff } ?p \ i \ * \ \langle X \rangle \ i)$

**using** *eval-poly-with-sum* **by** *blast*

**also have**  $\dots = (\sum_{i \in \{0..<\text{dim-vec } ?v0\}}. \text{Abstract-Linear-Poly.coeff } ?p \ i \ * \ \langle X \rangle \ i)$

**using** *eval-poly-with-sum-superset*[of  $\{0..<\text{dim-vec } ?v0\}$   $?p \ \langle X \rangle$ ] *calculation*

*vars-subset-dim-vec-to-lpoly-dim*[of  $?v0$ ] **by** *force*

**also have**  $\dots = (\sum_{i \in \{0..<\text{dim-vec } b\}}. \text{Abstract-Linear-Poly.coeff } ?p \ i \ * \ \langle X \rangle \ i)$

+

$(\sum_{i \in \{(\text{dim-vec } b)..<\text{dim-vec } ?v0\}}. \text{Abstract-Linear-Poly.coeff } ?p \ i$

$* \ \langle X \rangle \ i)$

**by** (*simp add: sum.atLeastLessThan-concat*)

**also have**  $\dots = (\sum_{i \in \{(\text{dim-vec } b)..<\text{dim-vec } ?v0\}}. \text{Abstract-Linear-Poly.coeff } ?p$

$i * \langle X \rangle i$   
**using** \* **by** *simp*  
**also have** ... =  $(\sum i \in \{0..<dim-vec\ b\}. ?v0 \$ i * \langle X \rangle i)$   
**using** *vec-to-lin-poly-coeff-access* **by** *auto*  
**also have** ... =  $(\sum i \in \{0..<dim-vec\ c\}. ?v0 \$ (i+dim-vec\ b) * \langle X \rangle (i+dim-vec\ b))$   
**using** *index-zero-vec(2)*[of *dim-vec\ b*] *index-append-vec(2)*[of  $0_v$  (*dim-vec\ b*) *c*]  
**\*\* \***  
*sum.shift-bounds-nat-ivl*[of  $(\lambda i. ?v0 \$ i * \langle X \rangle i)$   $0$  *dim-vec\ b* *dim-vec\ c*]  
**by** (*simp add: add commute*)  
**also have** ... =  $(\sum i \in \{0..<dim-vec\ c\}. c \$ i * \langle X \rangle (i+dim-vec\ b))$   
**by** *auto*  
**also have** ... =  $(\sum i \in \{0..<dim-vec\ c\}. c \$ i * y \$ i)$   
**using** *split-access-snd-2*[of *x y* (*dim-vec\ b*) (*dim-vec\ c*) *X*] *assms*  
**by** (*metis (mono-tags, lifting) atLeastLessThan-iff split-access-snd-2*  
*split-n-m-x-abbrev-dims(2) split-vec-dims(1) sum.cong*)  
**also have** ... =  $c \cdot y$   
**by** (*metis assms scalar-prod-def split-n-m-x-abbrev-dims(2)*)  
**finally show** ?thesis .  
**qed**

**lemma** *x-times-c-geq-y-times-b-split-dotP*:  
**assumes**  $\langle X \rangle \models_c x\text{-times-c-geq-y-times-b}\ c\ b$   
**assumes**  $(x, y) = \text{split-n-m-x}\ (dim-vec\ c)\ (dim-vec\ b)\ X$   
**shows**  $c \cdot x \geq b \cdot y$   
**using** *assms lpoly-of-v-equals-v-append0 eval-lpoly-eq-dot-prod-split2*[of *x y c b X*]  
*eval-lpoly-eq-dot-prod-split1*[of *x y c b X*]  
**by** (*auto simp: x-times-c-geq-y-times-b-def evaluate-minus*)

**lemma** *mult-right-leq*:  
**fixes**  $A :: ('a::\{comm-semiring-1, ordered-semiring\})\ mat$   
**assumes**  $dim-vec\ y = dim-vec\ b$   
**and**  $\forall i < dim-vec\ y. y \$ i \geq 0$   
**and**  $[A *_v\ x] \leq b$   
**shows**  $(A *_v\ x) \cdot y \leq b \cdot y$   
**proof** –  
**have**  $(\sum n < dim-vec\ b. (A *_v\ x) \$ n * y \$ n) \leq (\sum n < dim-vec\ b. b \$ n * y \$ n)$   
**by** (*metis (no-types, lifting) assms(1) assms(2) assms(3) lessThan-iff*  
*mat-times-vec-leq-def mult-right-mono sum-mono*)  
**then show** ?thesis  
**by** (*metis (no-types) assms(1) atLeast0LessThan scalar-prod-def*)  
**qed**

**lemma** *mult-right-eq*:  
**assumes**  $dim-vec\ x = dim-vec\ c$   
**and**  $[y *_v\ A] = c$   
**shows**  $(A^T *_v\ y) \cdot x = c \cdot x$   
**unfolding** *scalar-prod-def*  
**using** *atLeastLessThan-iff*[of  $- 0$  *dim-vec\ x*] *vec-times-mat-eq-def*[of *y A c*]  
*sum.cong*[of  $- \lambda i. (A^T *_v\ y) \$ i * x \$ i \lambda i. c \$ i * x \$ i$ ]



by (*metis* (*mono-tags*, *lifting*) *assms*(1) *assms*(2))

**lemma** *soundness-mat-x-leq*:

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

**assumes**  $\text{simplex } (\text{mat-x-leq-vec } A \ b) = \text{Sat } X$

**shows**  $\exists x. [A *_{\nu} x] \leq b$

**proof**

**define**  $x$  **where**  $x = \text{fst } (\text{split-n-m-x } (\dim\text{-col } A) (\dim\text{-row } A) X)$

**have**  $*$ :  $\dim\text{-vec } x = \dim\text{-col } A$  **by** (*simp add: split-i-j-x-def*  $x$ )

**have**  $\forall i < \dim\text{-vec } b. (A *_{\nu} x) \$ i \leq b \$ i$

**proof** (*standard*, *standard*)

**fix**  $i$

**assume**  $i < \dim\text{-vec } b$

**have**  $\text{row } A \ i \cdot x \leq b \$ i$

**using** *mat-x-leq-vec-sol*[*of*  $A \ b \ X \ i$ ]

**by** (*metis*  $\langle i < \dim\text{-vec } b \rangle$  *assms*(1) *assms*(2) *eval-lpoly-eq-dot-prod-split1*

*fst-conv index-row*(2) *matrix-to-lp-vec-to-lpoly-row simplex*(3) *split-i-j-x-def*

$x$ )

**then show**  $(A *_{\nu} x) \$ i \leq b \$ i$

**by** (*simp add:*  $\langle i < \dim\text{-vec } b \rangle$  *assms*(1))

**qed**

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

**using** *mat-times-vec-leqI*[*of*  $A \ b \ x$ , *OF* *assms*(1)  $*$ [*symmetric*]] **by** *auto*

**qed**

**lemma** *completeness-mat-x-leq*:

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

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

**proof** (*rule ccontr*)

**assume**  $1$ :  $\nexists X. \text{simplex } (\text{mat-x-leq-vec } A \ b) = \text{Inr } X$

**have**  $*$ :  $\nexists v. v \models_{cs} \text{set } (\text{mat-x-leq-vec } A \ b)$

**using** *simplex*(1)[*of*  $\text{mat-x-leq-vec } A \ b$ ] **using**  $1$  *sum.exhaust-sel* **by** *blast*

**then have**  $\dim\text{-vec } b = \dim\text{-row } A$  **using** *assms* *mat-times-vec-leqD*(1)[*of*  $A - b$ ]

**by** *auto*

**then obtain**  $x$  **where**  $x: [A *_{\nu} x] \leq b$

**using** *assms* **by** *blast*

**have**  $x\text{-A}$ :  $\dim\text{-vec } x = \dim\text{-col } A$

**using**  $x$  **by** *auto*

**define**  $v$  **where**  $v: v = (\lambda i. (\text{if } i < \dim\text{-vec } x \text{ then } x \$ i \text{ else } 0))$

**have**  $v\text{-d}$ :  $\forall i < \dim\text{-vec } x. x \$ i = v \ i$

**by** (*simp add: v*)

**have**  $v \models_{cs} \text{set } (\text{mat-x-leq-vec } A \ b)$

**proof**

**fix**  $c$

**assume**  $c \in \text{set } (\text{mat-x-leq-vec } A \ b)$

**then obtain**  $i$  **where**  $i: c = \text{LEQ } (\text{matrix-to-lpolies } A \ i) (b \$ i) \ i < \dim\text{-vec } b$

**by** *auto*

**let**  $?p = \text{matrix-to-lpolies } A \ i$

**have**  $2$ :  $?p \llbracket v \rrbracket = (\text{row } A \ i) \cdot x$

```

    using matrix-to-lpolies-lambda-valuate-scalarP[of i A x] v
    by (metis ‹dim-vec b = dim-row A› i(2) x-A)
  also have ... ≤ b $ i
    by (metis i(2) index-mult-mat-vec mat-times-vec-leq-def x)
  finally show v ⊨c c
    using i(1) satisfies-constraint.simps(3)[of v (matrix-to-lpolies A ! i) b $ i]
      2 ‹row A i · x ≤ b $ i› by simp
qed
then show False using * by auto
qed

lemma soundness-mat-x-eq-vec:
  assumes dim-row AT = dim-vec c
  assumes simplex (x-mat-eq-vec c AT) = Sat X
  shows ∃ x. [xv* A]=c
proof
  define x where x: x = fst (split-n-m-x (dim-col AT) (dim-row AT) X)
  have dim-vec x = dim-col AT
  unfolding split-i-j-x-def using split-vec-dims(1)[of (dim-col AT) - X x] fst-conv[of
x]
    by (simp add: split-i-j-x-def x)
  have ∀ i < dim-vec c. (AT *v x) $ i = c $ i
  proof (standard, standard)
    fix i
    assume a: i < dim-vec c
    have *: ‹X› ⊨cs set (x-mat-eq-vec c AT)
      using assms(2) simplex(3) by blast
    then have row AT i · x = c $ i
      using x-mat-eq-vec-sol[of c AT ‹X› i, OF * a] eval-lpoly-eq-dot-prod-split1 fstI
    by (metis a assms(1) index-row(2) matrix-to-lpolies-vec-of-row split-i-j-x-def
x)
    then show (AT *v x) $ i = c $ i
      unfolding mult-mat-vec-def using a assms(1) by auto
  qed
  then show [xv* A]=c
    using mat-times-vec-eqI[of A x c, OF ‹dim-vec x = dim-col AT›[symmetric]
assms(1)] by auto
qed

lemma completeness-mat-x-eq-vec:
  assumes ∃ x. [xv* A]=c
  shows ∃ X. simplex (x-mat-eq-vec c AT) = Sat X
proof (rule ccontr)
  assume 1: † X. simplex (x-mat-eq-vec c AT) = Inr X
  then have *: † v. v ⊨cs set (x-mat-eq-vec c AT)
    using simplex(1)[of x-mat-eq-vec c AT] using sum.exhaust-sel 1 by blast
  then have dim-vec c = dim-col A using assms
    by (metis index-transpose-mat(2) vec-times-mat-eqD(3))

```

```

obtain  $x$  where  $[x \ * \ A] = c$  using  $assms$  by  $auto$ 
then have  $dim\text{-}vec\ x = dim\text{-}col\ A^T$  using  $assms$ 
  by ( $metis\ \langle [x \ * \ A] = c \rangle\ vec\text{-}times\text{-}mat\text{-}eq\text{-}def$ )
define  $v$  where  $v: v = (\lambda i. (if\ i < dim\text{-}vec\ x\ then\ x\ \$i\ else\ 0))$ 
have  $v\text{-}d: \forall i < dim\text{-}vec\ x. x\ \$i = v\ i$ 
  by ( $simp\ add: v$ )
have  $v \models_{cs}\ set\ (x\text{-}mat\text{-}eq\text{-}vec\ c\ A^T)$ 
proof
  fix  $a$ 
  assume  $a \in set\ (x\text{-}mat\text{-}eq\text{-}vec\ c\ A^T)$ 
  then obtain  $i$  where  $i: a = EQ\ (matrix\text{-}to\text{-}lpolies\ A^T!\ i)\ (c\ \$i)\ i < dim\text{-}vec\ c$ 
    by ( $metis\ (no\text{-}types,\ lifting)\ add\text{-}cancel\text{-}right\text{-}left\ diff\text{-}zero\ in\text{-}set\ conv\text{-}nth\ length\text{-}map\ length\text{-}upt\ nth\text{-}map\text{-}upt\ x\text{-}mat\text{-}eq\text{-}vec.\ simps$ )
  let  $?p = matrix\text{-}to\text{-}lpolies\ A^T!\ i$ 
  have  $\mathcal{L}: ?p\ \|\ v\ \|\ = (row\ A^T\ i) \cdot x$ 
    using  $matrix\text{-}to\text{-}lpolies\ lambda\text{-}valuate\ scalarP\ [of\ i\ A^T\ x]\ v$ 
    by ( $metis\ \langle dim\text{-}vec\ c = dim\text{-}col\ A \rangle\ \langle dim\text{-}vec\ x = dim\text{-}col\ A^T \rangle\ i(\mathcal{L})\ index\text{-}transpose\text{-}mat(2)$ )
  also have  $\dots = c\ \$i$ 
    by ( $metis\ \langle [x \ * \ A] = c \rangle\ \langle dim\text{-}vec\ c = dim\text{-}col\ A \rangle\ i(2)\ index\text{-}mult\text{-}mat\text{-}vec\ index\text{-}transpose\text{-}mat(2)\ vec\text{-}times\text{-}mat\text{-}eqD(1)$ )
  finally show  $v \models_c a$ 
    using  $i(1)\ satisfies\text{-}constraint.\ simps(5)\ [of\ v\ (matrix\text{-}to\text{-}lpolies\ A^T!\ i)\ (c\ \$i)]$ 
by  $simp$ 
  qed
  then show  $False$ 
    using  $*$  by  $blast$ 
qed

```

```

lemma  $soundness\text{-}mat\text{-}leqb\text{-}eqc1:$ 
  assumes  $dim\text{-}row\ A = dim\text{-}vec\ b$ 
  assumes  $simplex\ (mat\text{-}leqb\text{-}eqc\ A\ b\ c) = Sat\ X$ 
  shows  $\exists x. [A \ * \ _v\ x] \leq b$ 
proof
  define  $x$  where  $x: x = fst\ (split\text{-}n\text{-}m\text{-}x\ (dim\text{-}col\ A)\ (dim\text{-}row\ A)\ X)$ 
  have  $*$ :  $dim\text{-}vec\ x = dim\text{-}col\ A$  by ( $simp\ add: split\text{-}i\text{-}j\text{-}x\text{-}def\ x$ )
  have  $\forall i < dim\text{-}vec\ b. (A \ * \ _v\ x)\ \$i \leq b\ \$i$ 
proof ( $standard,\ standard$ )
  fix  $i$ 
  assume  $i < dim\text{-}vec\ b$ 
  have  $row\ A\ i \cdot x \leq b\ \$i$ 
    using  $mat\text{-}x\text{-}leq\text{-}vec\ sol\ [of\ A\ b\ X\ i]$ 
    by ( $metis\ \langle i < dim\text{-}vec\ b \rangle\ assms(1)\ assms(2)\ fst\ conv\ split\text{-}i\text{-}j\text{-}x\text{-}def\ x\ index\text{-}mult\text{-}mat\text{-}vec\ mat\text{-}leqb\text{-}eqc\ split\text{-}simplex\ correct1\ mat\text{-}times\text{-}vec\ leqD(3)$ )
  then show  $(A \ * \ _v\ x)\ \$i \leq b\ \$i$ 
    by ( $simp\ add: \langle i < dim\text{-}vec\ b \rangle\ assms(1)$ )
qed
then show  $[A \ * \ _v\ x] \leq b$ 
  using  $mat\text{-}times\text{-}vec\ leqI\ [of\ A\ b\ x,\ OF\ assms(1)\ *[symmetric]]$  by  $auto$ 

```

qed

**lemma** *soundness-mat-leqb-eqc2*:

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

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

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

**shows**  $\exists y. [y \ v^* \ A]=c$

**proof** (*standard, intro mat-times-vec-eqI*)

**define**  $y$  **where**  $x: y = \text{snd } (\text{split-n-m-x } (\dim\text{-col } A) (\dim\text{-row } A) \ X)$

**have**  $*$ :  $\dim\text{-vec } y = \dim\text{-row } A$  **by** (*simp add: split-i-j-x-def x*)

**show**  $\dim\text{-col } A^T = \dim\text{-vec } y$  **by** (*simp add: \**)

**show**  $\dim\text{-row } A^T = \dim\text{-vec } c$  **using** *assms(1)* **by** *blast*

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

**proof** –

**fix**  $i$

**assume**  $a: i < \dim\text{-vec } c$

**have**  $[y \ v^* \ A]=c$

**using** *mat-leqb-eqc-split-correct2[of c A b - - y, OF assms(1)[symmetric] assms(2)[symmetric]]*

**by** (*metis Matrix.transpose-transpose assms(3) index-transpose-mat(2) simplex(3) snd-conv split-i-j-x-def x*)

**then show**  $(A^T \ *_v \ y) \ \$ \ i = c \ \$ \ i$

**by** (*metis a vec-times-mat-eq-def*)

qed

qed

**lemma** *completeness-mat-leqb-eqc*:

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

**and**  $\exists y. [y \ v^* \ A]=c$

**shows**  $\exists X. \text{simplex } (\text{mat-leqb-eqc } A \ b \ c) = \text{Sat } X$

**proof** (*rule ccontr*)

**assume**  $1: \nexists X. \text{simplex } (\text{mat-leqb-eqc } A \ b \ c) = \text{Sat } X$

**have**  $*$ :  $\nexists v. v \models_{cs} \text{set } (\text{mat-leqb-eqc } A \ b \ c)$

**using** *simplex(1)[of mat-leqb-eqc A b c]* **using**  $1$  *sum.exhaust-sel* **by** *blast*

**then have**  $\dim\text{-vec } b = \dim\text{-row } A$

**using** *assms mat-times-vec-leqD(1)[of A - b]* **by** *presburger*

**then obtain**  $x \ y$  **where**  $x: [A \ *_v \ x] \leq b$   $[y \ v^* \ A]=c$

**using** *assms* **by** *blast*

**have**  $x\text{-A}: \dim\text{-vec } x = \dim\text{-col } A$

**using**  $x$  **by** *auto*

**have**  $yr: \dim\text{-vec } y = \dim\text{-row } A$

**using** *vec-times-mat-eq-def x(2)* **by** *force*

**define**  $v$  **where**  $v: v = (\lambda i. (\text{if } i < \dim\text{-vec } (x@_v y) \text{ then } (x@_v y)\$i \text{ else } 0))$

**have**  $v\text{-d}: \forall i < \dim\text{-vec } (x@_v y). (x@_v y)\$i = v \ i$

**by** (*simp add: v*)

**have**  $i\text{-in}: \forall i \in \{0..< \dim\text{-vec } y\}. y\$i = v \ (i + \dim\text{-vec } x)$

**by** (*simp add: v*)

**have**  $v \models_{cs} \text{set } (\text{mat-leqb-eqc } A \ b \ c)$

**proof**

```

fix e
assume asm: e ∈ set (mat-leqb-egc A b c)
define lst where lst: lst = matrix-to-lpolies (two-block-non-interfering A AT)
let ?L = [LEQ (lst!i) (b$i) . i <- [0..vc)$i) . i <- [dim-vec b ..< dim-vec (b@vc)]]
have L: mat-leqb-egc A b c = ?L
  by (metis (full-types) lst mat-leqb-egc.simps)
then obtain i where i: e = ?L!i i ∈ {0..vc)
  using i(2) by auto
consider (leqb) i ∈ {0..c e
proof (cases)
  case leqb
  have il: i < dim-vec b
    using atLeastLessThan-iff leqb by blast
  have iA: i < dim-row A
    using ⟨dim-vec b = dim-row A⟩ ⟨i < dim-vec b⟩ by linarith
  then have *: e = LEQ (lst!i) (b$i)
    by (simp add: i(1) nth-append il)
  then have ... = LEQ ((matrix-to-lpolies A)!i) (b$i)
    using mat-leqb-egc-for-LEQ[of i b A c, OF il ⟨i < dim-row A⟩] L i(1) by
simp
  then have eqmp: lst!i = ((matrix-to-lpolies A)!i)
    by blast
  have sset: vars (lst!i) ⊆ {0..

```

```

    by (simp add: *)
next
case eqc
have igeq:  $i \geq \dim\text{-vec } b$ 
  using atLeastLessThan-iff eqc by blast
have *:  $i < \text{length } ?L$ 
  using atLeastLessThan-iff eqc by blast
have  $e = ?L!i$ 
  using L  $i(1)$  by presburger
have  $?L!i \in \text{set } [EQ (l!i) ((b@_v c)\$i). i <- [\dim\text{-vec } b..< \dim\text{-vec } (b@_v c)]]$ 
  using in-second-append-list length-map
  by (metis (no-types, lifting) igeq * length-upt minus-nat.diff-0)
  then have  $?L!i = [EQ (l!i) ((b@_v c)\$i). i <- [\dim\text{-vec } b..< \dim\text{-vec } (b@_v c)]]!(i - \dim\text{-vec } b)$ 
    by (metis (no-types, lifting)  $\langle \dim\text{-vec } b \leq i \rangle$  diff-zero leD
      length-map length-upt nth-append)
  then have  $?L!i = EQ (l!i) ((b@_v c)\$i)$ 
    using add-diff-inverse-nat diff-less-mono
    by (metis (no-types, lifting)  $\langle \dim\text{-vec } b \leq i \rangle$  * ldimbc leD nth-map-upt)
  then have  $e = EQ (l!i) ((b@_v c)\$i)$ 
    using  $i(1)$  by blast
with mat-leqb-eqc-for-EQ[of  $b$   $i$   $c$   $A$ , OF igeq]
have lsta:  $(l!i) = (\text{vec-to-lpoly } (0_v (\dim\text{-col } A) @_v \text{row } A^T (i - \dim\text{-vec } b)))$ 
  by (metis (no-types, lifting)  $\langle \dim\text{-vec } b = \dim\text{-row } A \rangle$  * ldimbc assms(2))
igeq
  index-append-vec(2) lst matrix-to-lpolies-vec-of-row vec-times-mat-eq-def
  two-block-non-interfering-dims(1) two-block-non-interfering-row-comp2 )
let  $?p = (\text{vec-to-lpoly } (0_v (\dim\text{-col } A) @_v \text{row } A^T (i - \dim\text{-vec } b)))$ 
have  $\dim\text{-poly } ?p \leq \dim\text{-col } A + \dim\text{-row } A$ 
  using  $\dim\text{-poly-of-append-vec}$ [of  $0_v (\dim\text{-col } A) \text{row } A^T (i - \dim\text{-vec } b)$ ]
  index-zero-vec(2)[of  $\dim\text{-col } A$ ]
  by (metis  $\langle \dim\text{-vec } (0_v (\dim\text{-col } A)) = \dim\text{-col } A \rangle$  index-row(2) in-
dex-transpose-mat(3))
have  $\forall i < \dim\text{-col } A. \text{Abstract-Linear-Poly.coeff } ?p \ i = 0$ 
  using  $\text{vec-coeff-append1}$ [of  $0_v (\dim\text{-col } A) \text{row } A^T (i - \dim\text{-vec } b)$ ]
  by (metis atLeastLessThan-iff index-zero-vec(1) index-zero-vec(2) zero-le)
  then have  $\dim\text{-vec } (0_v (\dim\text{-col } A) @_v \text{row } A^T (i - \dim\text{-vec } b)) = \dim\text{-col } A + \dim\text{-row } A$ 
    by (metis index-append-vec(2) index-row(2) index-transpose-mat(3) in-
dex-zero-vec(2))
  then have  $\text{allcr}: \forall j \in \{0..< \dim\text{-row } A\}. \text{Abstract-Linear-Poly.coeff } ?p \ (j + \dim\text{-col } A) = (\text{row } A^T (i - \dim\text{-vec } b))\$j$ 
    by (metis add-diff-cancel-right' atLeastLessThan-iff diff-add-inverse in-
dex-zero-vec(2)
      le-add-same-cancel2 less-diff-conv vec-coeff-append2)
  have  $\text{vs}: \text{vars } ?p \subseteq \{\dim\text{-col } A..< \dim\text{-col } A + \dim\text{-row } A\}$ 
  using  $\text{vars-vec-append-subset}$  by (metis index-row(2) index-transpose-mat(3))
  have  $?p \ \{\ v \ \} = (\sum_{j \in \text{vars } ?p} \text{Abstract-Linear-Poly.coeff } ?p \ j * v \ j)$ 
    using eval-poly-with-sum by blast

```

```

also have ... = ( $\sum j \in \{ \dim\text{-col } A .. < \dim\text{-col } A + \dim\text{-row } A \}$ . Abstract-Linear-Poly.coeff
?p j * v j)
  by (metis (mono-tags, lifting) DiffD2 vs coeff-zero finite-atLeastLessThan
mult-not-zero sum.mono-neutral-left)
also have ... = ( $\sum j \in \{ 0 .. < \dim\text{-row } A \}$ . Abstract-Linear-Poly.coeff ?p (j +  $\dim\text{-col}$ 
A) * v (j +  $\dim\text{-col}$  A))
  using sum.shift-bounds-nat-ivl[of  $\lambda j$ . Abstract-Linear-Poly.coeff ?p j * v j 0
 $\dim\text{-col } A$   $\dim\text{-row } A$ ]
  by (metis (no-types, lifting) add commute add-cancel-left-left)
also have ... = ( $\sum j \in \{ 0 .. < \dim\text{-row } A \}$ . Abstract-Linear-Poly.coeff ?p (j +  $\dim\text{-col}$ 
A) * y$j)
  using v i-in yr by (metis (no-types, lifting) sum.cong x-A)
also have ... = ( $\sum j \in \{ 0 .. < \dim\text{-row } A \}$ . (row  $A^T$  (i - dim-vec b))$j * y$j)
  using allcr by (metis (no-types, lifting) sum.cong)
also have ... = (row  $A^T$  (i - dim-vec b)) · y
  by (metis <dim-vec y = dim-row A> scalar-prod-def)
also have ... = (b@vc)$i
  using vec-times-mat-eqD[OF x(2)] * igeq by auto
finally show ?thesis
using e lsta satisfies-constraint.simps(5)[of - (lst ! i) ((b @v c) $ i)] by simp
qed
qed
then show False using * by blast
qed

```

```

lemma sound-and-complte-mat-leqb-eq [iff]:
  assumes  $\dim\text{-row } A^T = \dim\text{-vec } c$ 
  assumes  $\dim\text{-col } A^T = \dim\text{-vec } b$ 
  shows ( $\exists x$ .  $[A *_{\mathbb{V}} x] \leq b$ )  $\wedge$  ( $\exists y$ .  $[y *_{\mathbb{V}} A] = c$ )  $\longleftrightarrow$  ( $\exists X$ . simplex (mat-leqb-eq A
b c) = Sat X)
  by (metis assms(1) assms(2) completeness-mat-leqb-eq index-transpose-mat(3)
soundness-mat-leqb-eq1 soundness-mat-leqb-eq2)

```

## 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  $\equiv$  ( $\forall a \in \text{set } cs$ . *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  $\neq$  []  
**shows**  $(\forall x \in \text{set } (\text{transf-constraint } x). \exists a b. x = \text{LEQ } a b)$   
**using** *assms* **by** (*cases* x, *auto*+)

**lemma** *trans-constraints-creates-LEQ-only*:  
**assumes** *transf-constraints* xs  $\neq$  []  
**assumes**  $x \in \text{set } (\text{transf-constraints } xs)$   
**shows**  $\exists p r. \text{LEQ } p r = x$   
**using** *assms* **apply**(*induction* xs)  
**using** *trans-constraint-creates-LEQ-only* **apply**(*auto*)  
**apply** *fastforce*  
**apply** (*metis in-set-simps*(3) *trans-constraint-creates-LEQ-only*)  
**by** *fastforce*

**lemma** *non-strict-constr-no-LT*:  
**assumes** *nonstrict-constrs* cs  
**shows**  $\forall x \in \text{set } cs. \neg(\exists a b. \text{LT } a b = x)$   
**using** *assms* *nonstrict-constr.simps*(4) **by** *blast*

**lemma** *non-strict-constr-no-GT*:  
**assumes** *nonstrict-constrs* cs  
**shows**  $\forall x \in \text{set } cs. \neg(\exists a b. \text{GT } a b = x)$   
**using** *assms* *nonstrict-constr.simps*(5) **by** *blast*

**lemma** *non-strict-consts-cond*:  
**assumes**  $\bigwedge x. x \in \text{set } cs \implies \neg(\exists a b. \text{LT } a b = x)$   
**assumes**  $\bigwedge x. x \in \text{set } cs \implies \neg(\exists a b. \text{GT } a b = x)$   
**shows** *nonstrict-constrs* cs  
**by** (*metis assms*(1–2) *nonstrict-constr.elims*(3))

**lemma** *sat-constr-sat-transf-constrs*:  
**assumes**  $v \models_c cs$   
**shows**  $v \models_{cs} \text{set } (\text{transf-constraint } cs)$   
**using** *assms* **by** (*cases* cs) (*simp* *add: valuate-uminus valuate-minus*)+

**lemma** *sat-constrs-sat-transf-constrs*:  
**assumes**  $v \models_{cs} \text{set } cs$   
**shows**  $v \models_{cs} \text{set } (\text{transf-constraints } cs)$   
**using** *assms* **by**(*induction* cs, *simp*) (*metis UnE list.set-intros*(1)  
*list.set-intros*(2) *sat-constr-sat-transf-constrs set-append transf-constraints.simps*(2))



```

lemma sat-transf-constrs-sat-constr:
  assumes nonstrict-constr cs
  assumes  $v \models_{cs} \text{set } (\text{transf-constraint } cs)$ 
  shows  $v \models_c cs$ 
  using assms by (cases cs) (simp add: valuate-uminus valuate-minus)+

lemma sat-transf-constrs-sat-constrs:
  assumes nonstrict-constrs cs
  assumes  $v \models_{cs} \text{set } (\text{transf-constraints } cs)$ 
  shows  $v \models_{cs} \text{set } cs$ 
  using assms by (induction cs, auto) (simp add: sat-transf-constrs-sat-constr)

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*

*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*

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

**unfolding** *optimal-set-def* **using** *assms*

**by** *blast*

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

**using** *optimal-setI*[of  $x \{ x. [A *_{\nu} x] \leq b \} c$ ]  
**unfolding** *optimal-set-def optimal-setI*  
**by** (*simp add: assms(1) assms(2) sat-primal-def*)

**lemma** *min-lpI* [*intro*]:  
**assumes**  $[y \ \nu * A] = c$   
**and**  $(\bigwedge i. i < \dim\text{-vec } y \implies y \ \$ \ i \geq 0)$   
**assumes**  $(\bigwedge x. x \in \text{sat-dual } A \ c \implies (\lambda x \ y. (y \cdot c) \geq (x \cdot c)) \ y \ x)$   
**shows**  $y \in \text{min-lp } A \ b \ c$   
**using** *optimal-setI*[of  $y \ \text{sat-dual } A \ c \ c$ ]  
**unfolding** *optimal-set-def optimal-setI sat-dual-def*  
**by** (*simp add: assms(1) assms(2) assms(3) sat-dual-def*)

**lemma** *sat-primalD* [*dest*]:  
**assumes**  $x \in \text{sat-primal } A \ b$   
**shows**  $[A *_{\nu} x] \leq b$   
**using** *assms sat-primal-def* **by force**

**lemma** *sat-primalI* [*intro*]:  
**assumes**  $[A *_{\nu} x] \leq b$   
**shows**  $x \in \text{sat-primal } A \ b$   
**using** *assms sat-primal-def* **by force**

**lemma** *sat-dualD* [*dest*]:  
**assumes**  $y \in \text{sat-dual } A \ c$   
**shows**  $[y \ \nu * A] = c \ (\forall i < \dim\text{-vec } y. y \ \$ \ i \geq 0)$   
**using** *assms sat-dual-def* **apply force**  
**using** *assms sat-dual-def* **by force**

**lemma** *sat-dualI* [*intro*]:  
**assumes**  $[y \ \nu * A] = c \ (\forall i < \dim\text{-vec } y. y \ \$ \ i \geq 0)$   
**shows**  $y \in \text{sat-dual } A \ c$   
**using** *assms sat-dual-def* **by auto**

**lemma** *sol-dim-in-sat-primal*:  $x \in \text{sat-primal } A \ b \implies \dim\text{-vec } x = \dim\text{-col } A$   
**unfolding** *mat-times-vec-leq-def* **by** (*simp add: mat-times-vec-leq-def sat-primal-def*)

**lemma** *sol-dim-in-max-lp*:  $x \in \text{max-lp } A \ b \ c \implies \dim\text{-vec } x = \dim\text{-col } A$   
**unfolding** *optimal-set-def* **using** *sol-dim-in-sat-primal*[of  $x \ A \ b$ ] **by blast**

**lemma** *sol-dim-in-sat-dual*:  $x \in \text{sat-dual } A \ c \implies \dim\text{-vec } x = \dim\text{-row } A$   
**unfolding** *mat-times-vec-leq-def* **by** (*simp add: sat-dual-def vec-times-mat-eq-def*)

**lemma** *sol-dim-in-min-lp*:  $x \in \text{min-lp } A \ b \ c \implies \dim\text{-vec } x = \dim\text{-row } A$   
**unfolding** *optimal-set-def* **using** *sol-dim-in-sat-dual*[of  $x \ A$ ] **by blast**

**lemma** *min-lp-in-sat-dual*:  $x \in \text{min-lp } A \ b \ c \implies x \in \text{sat-dual } A \ c$   
**unfolding** *optimal-set-def* **using** *sol-dim-in-sat-dual*[of  $x \ A$ ] **by blast**

**lemma** *max-lp-in-sat-primal*:  $x \in \text{max-lp } A \ b \ c \implies x \in \text{sat-primal } A \ b$   
**unfolding** *optimal-set-def* **using** *sol-dim-in-sat-dual*[of  $x \ A$ ] **by** *blast*

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

**lemma** *dim-b-row-A*:  $\text{dim-vec } b = \text{dim-row } A$   
**using** *abstract-LP-axioms abstract-LP-def carrier-matD(1) carrier-vecD*  
**by** *metis*

**lemma** *dim-b-col-A*:  $\text{dim-vec } c = \text{dim-col } A$   
**using** *abstract-LP-axioms abstract-LP-def carrier-matD(2) carrier-vecD*  
**by** *metis*

**lemma** *weak-duality-aux*:  
**fixes**  $i \ j$   
**assumes**  $i \in \{c \cdot x \mid x. x \in \text{sat-primal } A \ b\}$   
**and**  $j \in \{b \cdot y \mid y. y \in \text{sat-dual } A \ c\}$   
**shows**  $i \leq j$   
**proof** –  
**obtain**  $x$  **where**  $x: i = c \cdot x \ [A \ *_v \ x] \leq b$   
**using** *assms* **by** *blast*  
**obtain**  $y$  **where**  $y: j = b \cdot y \ [y \ *_v \ A] = c \ (\forall i < \text{dim-vec } y. 0 \leq y \ \$ \ i)$   
**using** *assms* **by** *blast*  
**have**  $d1: \text{dim-vec } x = n$  **using** *mat-times-vec-leq-def*[of  $A \ x \ b$ ]  $x$   
**by** (*metis abstract-LP-axioms abstract-LP-def carrier-matD(2)*)  
**have**  $d2: \text{dim-vec } y = m$   
**by** (*metis abstract-LP-axioms abstract-LP-def carrier-matD(1) index-transpose-mat(3)*)  
*vec-times-mat-eq-def y(2)*)  
**have**  $i = c \cdot x$  **using**  $x$  **by** *auto*  
**also have**  $\dots = (A^T \ *_v \ y) \cdot x$   
**using** *mult-right-eq carrier-vecD y abstract-LP-def*  
**by** (*metis abstract-LP-axioms calculation d1*)  
**also have**  $\dots = (A \ *_v \ x) \cdot y$   
**using** *assoc-scalar-prod-mult-mat-vec*[*symmetric, of y m x n A*] *abstract-LP-axioms*  
*abstract-LP-def d1 d2*  
*carrier-vec-dim-vec* **by** *blast*  
**also have**  $\dots \leq b \cdot y$   
**using** *mult-right-leq*  
**by** (*metis index-transpose-mat(3) mat-times-vec-leq-def vec-times-mat-eq-def*)

```

x(2) y(2) y(3))
  also have ... = j using y by simp
  finally show i ≤ j .
qed

theorem weak-duality-theorem:
  assumes x ∈ max-lp A b c
  assumes y ∈ min-lp A b c
  shows x · c ≤ y · b
proof -
  define i where i: i = x · c
  define j where j: j = y · b
  have dx: dim-vec x = n
    using sol-dim-in-max-lp[of x c A b, OF assms(1)] abstract-LP-axioms ab-
  stract-LP-def
    carrier-matD(2) by blast
  have dy: dim-vec y = m
    using sol-dim-in-min-lp[of y c A, OF assms(2)] abstract-LP-axioms abstract-LP-def
    carrier-matD(1) by blast
  have *: i ∈ {c · x | x. [A *v x] ≤ b} using assms(1) unfolding optimal-set-def dx
  sat-primal-def
    using abstract-LP-axioms abstract-LP-def carrier-vec-dim-vec comm-scalar-prod
  dx i by blast
  have **: j ∈ {b · y | y. [y *v A] = c ∧ (∀ i < dim-vec y. y $ i ≥ 0)}
    using assms(2) unfolding optimal-set-def using abstract-LP-axioms abstract-LP-def
    carrier-vec-dim-vec comm-scalar-prod dy j by blast
  from weak-duality-aux[of i j] have i ≤ j unfolding sat-primal-def sat-dual-def
    using * ** by blast
  then show ?thesis using i j by auto
qed

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-egc A b c @
      from-index-geq0-vector (dim-vec c) (0v (dim-vec b)))
    of
      Unsat X ⇒ Unsat X
      | Sat X ⇒ Sat X)

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

```

**lemma** *create-opt-sol-satisfies*:

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

**shows**  $\langle X \rangle \models_{cs} \text{set } ((x\text{-times-}c\text{-geq-}y\text{-times-}b\ c\ b \# \text{mat-leqb-eqc } A\ b\ c \text{ @}$   
 $\text{from-index-geq0-vector } (\text{dim-vec } c) (0_v (\text{dim-vec } b))))$

**proof** –

**have** *simplex*  $(x\text{-times-}c\text{-geq-}y\text{-times-}b\ c\ b \# \text{mat-leqb-eqc } A\ b\ c \text{ @}$   
 $\text{from-index-geq0-vector } (\text{dim-vec } c) (0_v (\text{dim-vec } b))) = \text{Sat } X$

**proof** (*rule ccontr*)

**assume** *simplex*  $(x\text{-times-}c\text{-geq-}y\text{-times-}b\ c\ b \# \text{mat-leqb-eqc } A\ b\ c \text{ @}$   
 $\text{from-index-geq0-vector } (\text{dim-vec } c) (0_v (\text{dim-vec } b))) \neq \text{Inr } X$

**then have**  $\exists e. \text{simplex } (x\text{-times-}c\text{-geq-}y\text{-times-}b\ c\ b \# \text{mat-leqb-eqc } A\ b\ c \text{ @}$   
 $\text{from-index-geq0-vector } (\text{dim-vec } c) (0_v (\text{dim-vec } b))) = \text{Unsat } e$

**by** (*metis assms create-optimal-solutions.simps sum.case(2) sumE*)

**then have**  $\exists e. \text{create-optimal-solutions } A\ b\ c = \text{Unsat } e$

**using** *assms option.split by force*

**then show** *False* **using** *assms(1) assms by auto*

**qed**

**then show** *?thesis* **using** *simplex(3) by blast*

**qed**

**lemma** *create-opt-sol-sat-leq-mat*:

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

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

**and**  $(x, y) = \text{split-}i\text{-}j\text{-}x (\text{dim-col } A) (\text{dim-vec } b) X$

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

**proof** –

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

**using** *create-opt-sol-satisfies[of A b c X] sat-mono[of (mat-leqb-eqc A b c) - X]*

**using** *assms(2) by (metis append-Cons append-assoc in-set-conv-decomp)*

**then show** *?thesis* **using** *mat-leqb-eqc-split-correct1[of b A c X x y, OF assms(1)*

*\*] assms*

**by** *blast*

**qed**

**lemma** *create-opt-sol-sat-eq-mat*:

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

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

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

**and**  $(x, y) = \text{split-}i\text{-}j\text{-}x (\text{dim-vec } c) (\text{dim-vec } c + \text{dim-vec } b) X$

**shows**  $[y\ _v\ * A] = c$

**proof** –

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

**using** *create-opt-sol-satisfies[of A b c X] sat-mono[of (mat-leqb-eqc A b c) - X]*

*assms(2) assms by (metis UnCI list.set-intros(2) set-append)*

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

**using** *assms(1) by linarith*

**moreover have**  $\text{dim-col } A^T = \text{dim-vec } b$

**by** (*simp add: assms(2)*)

**ultimately show** *?thesis*

**using** *assms* **by** (*metis mat-leqb-eqc-split-correct2*[of *c A b X x y*, *OF assms(1) assms(2) \**]

$\langle \text{dim-vec } b = \text{dim-col } A^T \rangle \langle \text{dim-vec } c = \text{dim-row } A^T \rangle$ )

**qed**

**lemma** *create-opt-sol-satisfies-leq*:

**assumes** *create-optimal-solutions A b c = Sat X*

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

**shows**  $x \cdot c \geq y \cdot b$

**using** *create-opt-sol-satisfies*[of *A b c X*]

**by** (*metis assms(1) assms(2) carrier-vec-dim-vec comm-scalar-prod list.set-intros(1)*

*split-n-m-x-abbrev-dims(2) split-vec-dims(1) x-times-c-geq-y-times-b-split-dotP*)

**lemma** *create-opt-sol-satisfies-geq0*:

**assumes** *create-optimal-solutions A b c = Sat X*

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

**shows**  $\bigwedge i. i < \text{dim-vec } y \implies y\$i \geq 0$

**proof** –

**fix** *i*

**assume**  $i < \text{dim-vec } y$

**have**  $*$ :  $\langle X \rangle \models_{cs} \text{set } (\text{from-index-geq0-vector } (\text{dim-vec } c) (0_v (\text{dim-vec } b)))$

**using** *assms(1) create-opt-sol-satisfies* **by** (*metis UnCI append-Cons set-append*)

**have**  $**$ :  $i < \text{dim-vec } b$

**by** (*metis*  $\langle i < \text{dim-vec } y \rangle$  *assms(2) split-n-m-x-abbrev-dims(2)*)

**then show**  $0 \leq y \$ i$

**using** *from-index-geq0-vector-split-snd*[of *dim-vec c 0\_v (dim-vec b) X x y dim-vec b i*, *OF \* assms(2)*] **by** *simp*

**qed**

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

**shows**  $[A *_v x] \leq b [y *_v A] = c \ x \cdot c \geq y \cdot b \ \bigwedge i. i < \text{dim-vec } y \implies y\$i \geq 0$

**apply** (*metis Pair-inject assms(1) assms(2) create-opt-sol-sat-leq-mat dim-b-col-A dim-b-row-A split-i-j-x-def*)

**using** *assms(1) assms(2) create-opt-sol-sat-eq-mat dim-b-col-A dim-b-row-A*

**apply** (*metis index-transpose-mat(2) index-transpose-mat(3)*)

**using** *assms(1) assms(2) create-opt-sol-satisfies-leq* **apply** *blast*

**using** *assms(1) assms(2) create-opt-sol-satisfies-geq0* **by** *blast*

**lemma** *create-optim-in-sols*:

**assumes** *create-optimal-solutions*  $A \ b \ c = \text{Sat } X$   
**assumes**  $(x, y) = \text{split-n-m-x } (\text{dim-vec } c) (\text{dim-vec } b) \ X$   
**shows**  $c \cdot x \in \{c \cdot x \mid x. [A \ *_{\nu} \ x] \leq b\}$   
 $b \cdot y \in \{b \cdot y \mid y. [y \ *_{\nu} \ A] = c \wedge (\forall i < \text{dim-vec } y. y\ \$i \geq 0)\}$   
**using** *assms(1) assms(2) create-opt-sol-in-LP(1) apply blast*  
**using** *assms(1) assms(2) create-opt-sol-in-LP(2) create-opt-sol-in-LP(4) by blast*

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

**assumes** *create-optimal-solutions*  $A \ b \ c = \text{Sat } X$   
**assumes**  $(x, y) = \text{split-n-m-x } (\text{dim-vec } c) (\text{dim-vec } b) \ X$   
**shows**  $c \cdot x \leq b \cdot y$   
**using** *weak-duality-aux[of c · x b · y] create-optim-in-sols[of X x y, OF assms]*  
**by** *auto*

**lemma** *min-max-for-sol*:

**assumes** *create-optimal-solutions*  $A \ b \ c = \text{Sat } X$   
**assumes**  $(x, y) = \text{split-n-m-x } (\text{dim-vec } c) (\text{dim-vec } b) \ X$   
**shows**  $c \cdot x = b \cdot y$   
**using** *create-opt-sol-in-LP(3)[of X x y, OF assms] cx-leq-bx-in-creating-opt[OF assms]*  
 $\text{comm-scalar-prod}[of \ c \ \text{dim-vec } \ c \ x] \ \text{comm-scalar-prod}[of \ b \ \text{dim-vec } \ b \ y]$   
**by** *(metis add-diff-cancel-left' antisym assms(2) carrier-vec-dim-vec split-vec-dims(1) split-vec-dims(2))*

**lemma** *create-opt-solutions-correct*:

**assumes** *create-optimal-solutions*  $A \ b \ c = \text{Sat } X$   
**assumes**  $(x, y) = \text{split-n-m-x } (\text{dim-vec } c) (\text{dim-vec } b) \ X$   
**shows**  $x \in \text{max-lp } A \ b \ c$   
**proof**  
**show**  $[A \ *_{\nu} \ x] \leq b$   
**using** *assms(1) assms(2) create-opt-sol-in-LP(1) by blast*  
**fix**  $z$   
**assume**  $a: [A \ *_{\nu} \ z] \leq b$   
**have**  $1: c \cdot z \in \{c \cdot x \mid x. x \in \text{sat-primal } A \ b\}$   
**using** *sat-primalI[of A z b, OF a] by blast*  
**have**  $2: b \cdot y \in \{b \cdot y \mid y. y \in \text{sat-dual } A \ c\}$   
**using** *sat-dualI*  
**by** *(metis (mono-tags, lifting) assms(1) assms(2) create-opt-sol-in-LP(2) mem-Collect-eq rat-LP.create-opt-sol-in-LP(4) rat-LP-axioms)*  
**then have**  $c \cdot z \leq b \cdot y$   
**using** *weak-duality-aux[of c · z b · y, OF 1 2] sat-primalI[of A z b, OF a] by blast*  
**also have**  $\dots = x \cdot c$   
**by** *(metis assms(1) assms(2) carrier-vec-dim-vec comm-scalar-prod min-max-for-sol split-n-m-x-abbrev-dims(1))*  
**finally show**  $z \cdot c \leq x \cdot c$

by (metis a carrier-vec-dim-vec comm-scalar-prod dim-b-col-A mat-times-vec-leqD(2))  
qed

**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** –

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

by (metis Inr-Inl-False assms old.sum.exhaust old.sum.simps(5) *optimize-no-cond.simps*)

**have**  $x = (\text{fst } (\text{split-n-m-x } (\text{dim-vec } c) (\text{dim-vec } b) X))$

using  $X$  assms by (metis old.sum.inject(2) old.sum.simps(6) *optimize-no-cond.simps*)

**then show** *?thesis*

using *create-opt-solutions-correct*[of  $X\ x$ ] by (metis  $X$  *fst-conv* old.prod.exhaust)

qed

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

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

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

using *max-lp-in-sat-primal*[OF *optimize-no-cond-correct*[OF assms]] by auto

end

**fun** *maximize* **where**

*maximize*  $A\ b\ c = (\text{if } \text{dim-vec } b = \text{dim-row } A \wedge \text{dim-vec } c = \text{dim-col } A \text{ then}$   
*Some* (*optimize-no-cond*  $A\ b\ c$ )  
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** –

**have**  $*$ :  $\text{dim-vec } b = \text{dim-row } A \wedge \text{dim-vec } c = \text{dim-col } A$

by (metis assms *maximize.simps* option.distinct(1))

**then interpret** *rat*: *rat-LP*  $A\ b\ c$  *dim-vec*  $b$  *dim-vec*  $c$

by (metis *abstract-LP-def* *carrier-mat-triv* *carrier-vec-dim-vec* *rat-LP.intro*)

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

using assms  $*$  by auto

**then show** *?thesis*

by (*simp* add: *rat.optimize-no-cond-correct*)

qed

**lemma** *maximize-option-elim*:

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

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

by (metis assms *maximize.simps* option.distinct(1))+

**lemma** *optimize-sol-dimension*:



**assumes** *maximize*  $A\ b\ c = \text{Some}\ (\text{Sat}\ x)$   
**shows**  $x \in \text{carrier-vec}\ (\text{dim-col}\ A)$   
**using** *assms carrier-dim-vec max-lp-in-sat-primal optimize-sound sol-dim-in-sat-primal*  
**by** *blast*

**lemma** *optimize-sat*:

**assumes** *maximize*  $A\ b\ c = \text{Some}\ (\text{Sat}\ x)$   
**shows**  $[A\ *_v\ x] \leq b$   
**using** *assms maximize-option-elim[OF assms]*  
*max-lp-in-sat-primal[OF optimize-sound[of A b c x, OF assms]]* **by** *blast*

**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*  
**using** *of-rat-val-constraint simplex-real(3)* **by** *blast*

**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.