Linear-Programming

Julian Parsert

September 23, 2019

Abstract

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

Contents

1 Related work

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 Allamigeon 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-vec* $(u \ @_v \ w) =$ *dim-vec* x
shows $(u \ @_v \ w) \cdot x = u \cdot$ (*vec-first* x (*dim-vec* u)) $+ \ w \cdot$ (*vec-last* x (*dim-vec* w))
proof −
have $(u \ @_v \ w) \cdot$ (*vec-first* x (*dim-vec* u) $@_v$ *vec-last* x (*dim-vec* w)) =
$u \cdot$ *vec-first* x (*dim-vec* u) $+ \ w \cdot$ *vec-last* x (*dim-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-vec* $(u \ @_v \ w) =$ *dim-vec* x
shows $(u \ @_v \ w) \cdot x = (\sum i \in \{0..<$ *dim-vec* $u\}. \ u\$i * x\$i) + (\sum i \in \{0..<$ *dim-vec* $w\}. \ w\$i * x\$(i +$ *dim-vec* $u))$
proof −
define ix **where** $ix =$ *vec-first* x (*dim-vec* u)
define lx **where** lx: $lx =$ *vec-last* x (*dim-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 $... = (\sum \ i \in \{0 \ ..< \ dim\text{-}vec \ u\}. \ (u \ @_v \ w) \ \$ \ i * x \ \$ \ i) +$
$(\sum \ i \in \{dim\text{-}vec \ u \ ..< \ dim\text{-}vec \ (u \ @_v \ w)\}. \ (u \ @_v \ w) \ \$ \ i * x \ \$ \ i)$
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-vec u*]]
le-add1[*of dim-vec u dim-vec w*] *index-append-vec*(2)[*of u w*] **by** *simp*
also have $*$: $... = (\sum i \in \{0..<$*dim-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-vec* $w\}. \ w\$i * x\$(i +$ *dim-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*)
(*metis* ‹*dim-vec* $(u \ @_v \ w) =$ *dim-vec* $u +$ *dim-vec* w› *add.commute add-diff-cancel-right' assms*)
then show *?thesis*
using $*$ *calculation* **by** *auto*
qed

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

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

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

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

lemma *vector-cases-append*:
fixes *x* :: *'a vec*
shows *x* = *vNil* $\vee$ ($\exists$ *v a*. *x* = *v* $@_v$ $[a]_v$)
proof −
have *x* $\neq$ *vNil* $\Longrightarrow$ ($\exists$ *v a*. *x* = *v* $@_v$ $[a]_v$)
proof (*rule ccontr*)
assume *a1*: *x* $\neq$ *vNil*
assume *na*: ¬ ($\exists$ *v a*. *x* = *v* $@_v$ $[a]_v$)
have *dim-vec x* $\geq$ *1*
using *a1 eq-vecI* **by** *auto*
define *v* **where** *v*: *v* = *vec* (*dim-vec x* − *1*) ($\lambda i.$ *x* $ *i*)
have *v'*: $\forall$ *i* < *dim-vec v*. *v* $ *i* = *x* $ *i*
using *v* **by** *auto*
define *a* **where** *a*: *a* = *x* $ (*dim-vec x* − *1*)
have *a'*: $[a]_v$ $ *0* = *a* **by** *simp*
have *ff1*: *1* + *dim-vec v* = *dim-vec x*
by (*metis* (*no-types*) ‹*1* $\leq$ *dim-vec x*› *add-diff-cancel-left' dim-vec le-Suc-ex v*)
have $\forall$ *i* < *dim-vec x*. *x*$*i* = (*v* $@_v$ $[a]_v$)$*i*
proof (*standard*,*standard*)
fix *i* :: *nat*
assume *as*: *i* < *dim-vec x*
have *x* $ *dim-vec v* = *a*
by (*simp add*: *a v*)
then have *x* $ *i* = (*v* $@_v$ $[a]_v$) $ *i*
using *ff1 as* **by** (*metis* (*no-types*) *One-nat-def a' add.left-neutral add-Suc-right add-diff-cancel-left' add-diff-cancel-right' dim-vec index-append-vec*(*1*) *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 dim-vec eq-vecI ff1 index-append-vec*(*2*) *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** $\bigwedge a\ v.\ P\ v \Longrightarrow P\ (v\ @_v\ [a]_v)$
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-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 ... $= (\sum i{\in}\{0..{<}dim\text{-}vec\ z{-}1\}.\ (y\ @_v\ [x]_v)\ \$\ i * z\ \$\ i) +$
$(y\ @_v\ [x]_v)\$(dim\text{-}vec\ z{-}1) * z\$(dim\text{-}vec\ z{-}1)$
by (*simp add*: *assms*)
also have ... $= (\sum i{\in}\{0..{<}dim\text{-}vec\ y\}.\ (y\ @_v\ [x]_v)\ \$\ i * z\ \$\ i) +$
$(y\ @_v\ [x]_v)\$(dim\text{-}vec\ y) * z\$(dim\text{-}vec\ y)$
using *assms* **by** *auto*
also have ... $= (\sum i{\in}\{0..{<}dim\text{-}vec\ y\}.\ y\ \$\ i * z\ \$\ i) +$
$x * z\$(dim\text{-}vec\ y)$
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 $i < \textit{dim-row}\ A$
assumes $\textit{dim-col}\ A = \textit{dim-col}\ B$
shows $\textit{row}\ (A\ @_r\ B)\ i = \textit{row}\ A\ i$
proof
show $\textit{dim-vec}\ (\textit{Matrix.row}\ (A\ @_r\ B)\ i) = \textit{dim-vec}\ (\textit{Matrix.row}\ A\ i)$
by (*simp add*: *append-rows-def*)
fix *ia*
assume $ia < \textit{dim-vec}\ (\textit{row}\ A\ i)$
have $\textit{row}\ (A\ @_r\ B)\ i = (\textit{row}\ A\ i\ @_v\ \textit{row}\ (0_m\ (\textit{dim-row}\ A)\ 0)\ i)$
unfolding *append-rows-def* **using**
carrier-mat-triv[*of A*] *row-four-block-mat*(*1*)[*of A dim-row A*
- 0_m (*dim-row A*) *0 0 B dim-row B* 0_m (*dim-row B*) *0 i*, *OF - - - - assms*(*1*)]
by (*metis assms*(*2*) *carrier-mat-triv zero-carrier-mat*)
also have ... $= \textit{row}\ A\ i\ @_v\ \textit{vNil}$
by (*simp add*: *assms*(*1*))
also have ... $= \textit{row}\ A\ i$
by *auto*
finally show $\textit{row}\ (A\ @_r\ B)\ i\ \$\ ia = \textit{row}\ A\ i\ \$\ ia$
by *auto*
qed

lemma *append-rows-access2* [*simp*]:
assumes $i \geq \textit{dim-row}\ A$
assumes $i < \textit{dim-row}\ A + \textit{dim-row}\ B$
assumes $\textit{dim-col}\ A = \textit{dim-col}\ B$
shows $\textit{row}\ (A\ @_r\ B)\ i = \textit{row}\ B\ (i - \textit{dim-row}\ A)$
proof
show $\textit{dim-vec}\ (\textit{row}\ (A\ @_r\ B)\ i) = \textit{dim-vec}\ (\textit{row}\ B\ (i - \textit{dim-row}\ A))$
by (*simp add*: *append-rows-def assms*(*3*))
fix *ia*
assume $ia < \textit{dim-vec}\ (\textit{row}\ B\ (i - \textit{dim-row}\ A))$
have $\textit{row}\ (A\ @_r\ B)\ i = (\textit{row}\ B\ (i - \textit{dim-row}\ A)\ @_v\ \textit{row}\ (0_m\ (\textit{dim-row}\ B)\ 0)\ (i - \textit{dim-row}\ A))$
unfolding *append-rows-def* **using** *carrier-mat-triv*[*of A*] *row-four-block-mat*(*2*)[*of A dim-row A*
- 0_m (*dim-row A*) *0 0 B dim-row B* 0_m (*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 ... $= \textit{row}\ B\ (i - \textit{dim-row}\ A)\ @_v\ \textit{vNil}$
by *fastforce*
also have ... $= \textit{row}\ B\ (i - \textit{dim-row}\ A)$
by *auto*
finally show $\textit{row}\ (A\ @_r\ B)\ i\ \$\ ia = \textit{row}\ B\ (i - \textit{dim-row}\ A)\ \$\ ia$
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* ∈ *carrier-mat* (*dim-row A*) (*dim-col A* + *1*)
by *auto*

lemma *mat-append-row-dims*:
shows *mat-append-row A c* ∈ *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*)
(λ *i*. *?nA* $$ (*i*, (*dim-col A*)))) *i*
by (*simp add*: *col-def*)
also have ... = *vec-index* (*vec* (*dim-row A*) (λ *i*. *?nA* $$ (*i*, (*dim-col A*)))) *i*
by *auto*
also have ... = *vec-index* ((*cols A* @ [*b*]) ! *dim-col A*) *i*
by (*simp add*: ‹*i* < *dim-vec b*› *assms mat-of-cols-index*)
also have ... = *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-col A = dim-vec c*
shows *row (mat-append-row A c) (dim-row A) = c*
proof
let *?nA = (mat-of-rows (dim-col A) (Matrix.rows A @ [c]))*
show *dim-vec (Matrix.row (mat-append-row A c) (dim-row A)) = dim-vec c*
using *assms* **by** *simp*
fix *i* **assume** *i < dim-vec c*
from *mat-append-row.simps[of A c]*
have *row (mat-append-row A c) (dim-row A) $ i = vec-index (row ?nA (dim-row A)) i*
by *auto*
also have *... = vec-index (vec (dim-col ?nA) (λ j. ?nA $$ (dim-row A,j))) i*
by *(simp add: Matrix.row-def)*
also have *... = vec-index ((rows A @ [c]) ! dim-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 *... = vec-index c i*
by *(metis length-rows nth-append-length)*
finally show *Matrix.row (mat-append-row A c) (dim-row A) $ i = c $ i* **.**
qed

lemma *mat-append-row-in-mat*:
assumes *i < dim-row A*
shows *row (mat-append-row A r) i = 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-col A*
and *dim-col A = dim-vec b*
shows *vec-index (col (mat-append-row A b) i) (dim-row A) = 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-row A = dim-vec b*
and *i < dim-row A*
and *j < dim-col A*

shows (*row* (*mat-append-col A b*) *i*) $ *j* = (*row A i*) $ *j*
using *Matrix.row-transpose*[*of j A*, *OF assms*(*3*)]
Matrix.transpose-transpose[*of* (*mat-append-col A b*)] *assms carrier-matD*(*1*)
carrier-matD(*2*) *cols-length cols-transpose index-col index-row*(*1*)[*of i 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* < *dim-row A*
and *dim-row A* = *dim-vec b*
shows *row* (*mat-append-col A b*) *i* = *row A i* @$_v$ [*vec-index b i*]$_v$
proof
show *1*: *dim-vec* (*Matrix.row* (*mat-append-col A b*) *i*) = *dim-vec* (*Matrix.row A i* @$_v$ [*b* $ *i*]$_v$)
by *simp*
fix *ia*
assume *a*: *ia* < *dim-vec* (*Matrix.row A i* @$_v$ [*b* $ *i*]$_v$)
consider *ia* = *dim-col A* | *ia* < *dim-col A*
using *a less-SucE* **by** *auto*
then show *row* (*mat-append-col A b*) *i* $ *ia* = (*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 *row* (*mat-append-col A b*) *i* $ *ia* = (*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** *one-element-vec n e* = *vec n* (λ*i*. *e*)

lemma *one-element-vec-carrier*: *one-element-vec n e* ∈ *carrier-vec n*
unfolding *one-element-vec-def* **by** *auto*

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

lemma *one-element-vec-access* [*simp*]: ⋀*i*. *i* < *n* ⟹ *vec-index* (*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* (λ*j*. (*if i* = *j then v else 0*))

lemma *single-nz-val-carrier*: *single-nz-val n i v* ∈ *carrier-vec n*
by *auto*

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

lemma *single-nz-val-access2* [*simp*]: *i* < *n* ⟹ *j* < *n* ⟹ *i* ≠ *j*⟹ *single-nz-val n i v* $ *j* = *0*
by (*auto*)

lemma *i* < *n* ⟹ (*v* ·$_v$ *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* ·$_v$ (*unit-vec n i*) = *single-nz-val n i v*
proof
show ∗: *dim-vec* (*v* ·$_v$ *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* ·$_v$ *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 ⋀*j*. *j* < *dim-vec v* ⟹ *j* ≠ *i* ⟹ *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* · *x* = *v* ∗ *x* $ *i*
proof −
let *?y* = *single-nz-val n i v*
have *single-nz-val n i v* · *x* = (∑ *i*∈{*0* ..< *dim-vec x*}. *?y* $ *i* ∗ *x* $ *i*)
unfolding *scalar-prod-def* **by** *auto*
also have ... = (∑ *i*∈{*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 ... = $(\sum i{\in}\{0\ ..{<}\ \textit{dim-vec}\ x\}{-}\{i\}.\ ?y\ \$\ i * x\ \$\ i) + v * x\ \$\ i$
by (*simp add*: *assms*(*1*))
also have ... = $v * x\ \$\ i$
proof −
have $\bigwedge j.\ j \in \{0\ ..{<}\ \textit{dim-vec}\ x\}{-}\{i\} \Longrightarrow ?y\ \$\ j * x\ \$\ j = 0$
by (*simp add*: *assms*(*2*))
then have $(\sum i{\in}\{0\ ..{<}\ \textit{dim-vec}\ x\}{-}\{i\}.\ ?y\ \$\ i * x\ \$\ i) = 0$ **by** *auto*
then show *?thesis* **by** *auto*
qed
finally show *?thesis* **.**
qed

lemma *single-nz-zero-singleton*: *single-nz-val 1 0 v* = $[v]_v$
by (*auto*)

lemma *append-one-elem-zero-dotP*:
assumes *dim-vec u* = *m*
and *dim-vec x* = *n*
shows (*one-element-vec n e* $@_v$ (0_v *m*)) $\cdot$ (*x* $@_v$ *u*) = ($\sum i{\in}\{0\ ..{<}\ \textit{dim-vec}\ x\}$. *e* $*$ *x* $ *i*)
proof −
let *?OEV* = *one-element-vec n e*
have *dim-vec* (*?OEV* $@_v$ (0_v *m*)) = *dim-vec* (*x* $@_v$ *u*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *one-element-vec-carrier*)
have (*one-element-vec n e* $@_v$ 0_v *m*) $\cdot$ (*x* $@_v$ *u*) = *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 ... = ($\sum i{\in}\{0..{<}\textit{dim-vec}\ x\}$. *?OEV* $ *i* $*$ *x* $ *i*) + ($\sum i{\in}\{0..{<}\textit{dim-vec}\ u\}$. ($0_v$ *m*)$*i* $*$ *u*$*i*)
unfolding *scalar-prod-def* **by** *blast*
also have ... = ($\sum i{\in}\{0..{<}\textit{dim-vec}\ x\}$. *?OEV* $ *i* $*$ *x* $ *i*)
using *assms*(*1*) **by** *auto*
also have ... = ($\sum i{\in}\{0..{<}\textit{dim-vec}\ x\}$. *e* $*$ *x* $ *i*)
using *assms*(*2*) **by** *auto*
finally show *?thesis* **.**
qed

lemma *one-element-vec-dotP*:
assumes *dim-vec x* = *n*
shows (*one-element-vec n e*) $\cdot$ *x* = ($\sum i{\in}\{0\ ..{<}\ \textit{dim-vec}\ x\}$. *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* $\Longrightarrow$ $[v]_v \cdot x = v * x\$0$
by (*metis dim-vec index-vec less-one single-nz-valI single-nz-val-dotP*)

lemma *singletons-dotP* [*simp*]: $[v]_v \cdot [w]_v = v * w$

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

lemma *singleton-appends-dotP* [*simp*]: *dim-vec x* = *dim-vec y* ⟹ (*x* @$_v$ [*v*]$_v$) · (*y* @$_v$ [*w*]$_v$) = *x* · *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* ‴}) = (∑ *x*∈ *vars v*. *coeff v x* ∗ *X x*)
using *linear-poly-sum sum.cong* **by** *fastforce*

lemma *eval-poly-with-sum-superset*:
assumes *finite S*
assumes *S* ⊇ *vars v*
shows (*v* {‴*X*‴}) = (∑ *x*∈*S*. *coeff v x* ∗ *X x*)
proof −
define *D* **where** *D*: *D* = *S* − *vars v*
have *zeros*: ∀ *x* ∈ *D*. *coeff v x* = *0*
using *D coeff-zero* **by** *auto*
have *fnt*: *finite* (*vars v*)
using *finite-vars* **by** *auto*
have (*v* {‴*X*‴}) = (∑ *x*∈ *vars v*. *coeff v x* ∗ *X x*)
using *linear-poly-sum sum.cong* **by** *fastforce*
also have ... = (∑ *x*∈ *vars v*. *coeff v x* ∗ *X x*) + (∑ *x*∈*D*. *coeff v x* ∗ *X x*)
using *zeros* **by** *auto*
also have ... = (∑ *x*∈ *vars v* ∪ *D*. *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*: ‹⋀*g*. *sum g S* = *sum g* (*S* − *vars v*) + *sum g* (*vars v*)› *D*)
also have ... = (∑ *x*∈*S*. *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**
  list-to-lpoly cs = *sum-list* (*map2* (λ *i c*. *lp-monom c i*) [*0*..<*length cs*] *cs*)

lemma *empty-list-0poly*:
  shows *list-to-lpoly* [] = *0*
  unfolding *list-to-lpoly-def* **by** *simp*

lemma *sum-list-map-upto-coeff-limit*:
  assumes *i* ≥ *length L*
  shows *coeff* (*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* < *length cs*
  assumes *cs* ≠ []
  shows *coeff* (*list-to-lpoly cs*) *i* = *cs*!*i*
  using *assms*(*2*) *assms*(*1*)
proof (*induction cs rule*: *rev-nonempty-induct*)
  fix *x* ::*rat*
  assume *i* < *length* [*x*]
  have (*list-to-lpoly* [*x*]) = *lp-monom x 0*
    by (*simp add*: *list-to-lpoly-def*)
  then show *coeff* (*list-to-lpoly* [*x*]) *i* = [*x*] ! *i*
    using ‹*i* < *length* [*x*]› *list-to-lpoly-def* **by** *auto*
next
  fix *x* :: *rat*
  fix *xs* :: *rat list*
  assume *xs* ≠ []
  assume *IH*: *i* < *length xs* ⟹ *coeff* (*list-to-lpoly xs*) *i* = *xs* ! *i*
  assume *i* < *length* (*xs* @ [*x*])
  consider (*le*) *i* < *length xs* | (*eq*) *i* = *length xs*
    using ‹*i* < *length* (*xs* @ [*x*])› *less-Suc-eq* **by** *auto*
  then show *coeff* (*list-to-lpoly* (*xs* @ [*x*])) *i* = (*xs* @ [*x*]) ! *i*
  proof (*cases*)
    case *le*
    have *coeff* (*lp-monom x* (*length xs*)) *i* = *0*
      using *le* **by** *auto*
    have *coeff* (*sum-list* (*map2* (λ*x y*. *lp-monom y x*)
        [*0*..<*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 (λx y. lp-monom y x) [0..<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 (λ x y. lp-monom y x) [0..<length (xs @ [x])] (xs @
[x]))) =
        sum-list (map (λ(x,y). lp-monom y x) (zip [0..<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) = 0_v 0 ⟷ p = 0
```

```
proof(standard)
  show lpoly-to-vec p = 0_v 0 ⟹ p = 0
  proof (rule contrapos-pp)
    assume p ≠ 0
    then have vars p ≠ {}
      by (simp add: vars-empty-zero)
    then have dim-poly p > 0
      by (simp)
    then show lpoly-to-vec p ≠ 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 ⟹ ∃ i≥d. coeff x i ≠ 0
   by (simp split: if-splits) (meson Max-in coeff-zero finite-vars less-Suc-eq-le)

lemma dimpoly-all-zero-limit:
  assumes ⋀i. i ≥ d ⟹ coeff x i = 0
  shows dim-poly x ≤ d
proof −
  have (∀ i≥ d. coeff x i = 0) ⟹ dim-poly x ≤ d
  proof (rule contrapos-pp)
    assume ¬ dim-poly x ≤ d
    then have dim-poly x > d by linarith
    then have ∃ i ≥ d. coeff x i ≠ 0
      using dim-poly-greater-ex-coeff[of d x] by blast
    then show ¬ (∀ i≥d. 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 ≤ d x = p + lp-monom c d
proof −
  define c' where c': c' = coeff x d
  have f: ∀ i>d. coeff x i = 0
    using assms by auto
  have ∗: x = x − (lp-monom c' d) + (lp-monom c' d)
    by simp
  have coeff (x − (lp-monom c' d)) d = 0
    using c' by simp
```

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

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

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

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

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

lemma *addition-over-lin-poly-to-vec*:
fixes $x\ y$
assumes $a < dim\text{-}poly\ x$
assumes $dim\text{-}poly\ x = dim\text{-}poly\ y$
shows $(lpoly\text{-}to\text{-}vec\ x + lpoly\text{-}to\text{-}vec\ y)\ \$\ a = coeff\ (x + y)\ a$
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*: $length\ cs \geq dim\text{-}poly\ (list\text{-}to\text{-}lpoly\ cs)$
using *dimpoly-all-zero-limit sum-list-map-upto-coeff-limit* **by** *blast*

Transform rational vectors to linear polynomials

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

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

shows $y \$ x = coeff\ (vec\text{-}to\text{-}lpoly\ y)\ x$
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 $(x + y) \$ a = coeff\ (vec\text{-}to\text{-}lpoly\ x + vec\text{-}to\text{-}lpoly\ y)\ a$
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 $coeff\ (vec\text{-}to\text{-}lpoly\ xs)\ i = 0$
by (*simp add*: *assms sum-list-map-upto-coeff-limit*)

lemma *vec-to-poly-dim-less*:
$dim\text{-}poly\ (vec\text{-}to\text{-}lpoly\ x) \leq dim\text{-}vec\ x$
using *list-to-lpoly-dim-less*[*of list-of-vec x*] **by** *simp*

lemma *vec-to-lpoly-from-lpoly-coeff-dual1*:
$coeff\ (vec\text{-}to\text{-}lpoly\ (lpoly\text{-}to\text{-}vec\ p))\ i = coeff\ p\ i$
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 $i < dim\text{-}vec\ (lpoly\text{-}to\text{-}vec\ (vec\text{-}to\text{-}lpoly\ v))$
shows $(lpoly\text{-}to\text{-}vec\ (vec\text{-}to\text{-}lpoly\ v)) \$ i = v \$ i$
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*: $vars\ (vec\text{-}to\text{-}lpoly\ v) \subseteq \{0..<dim\text{-}vec\ v\}$
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 $(\sum a = 0..<dim\text{-}vec\ A.\ coeff\ (vec\text{-}to\text{-}lpoly\ \ A)\ a * X\ a) =$
$(\sum a = 0..<dim\text{-}poly\ (vec\text{-}to\text{-}lpoly\ A).\ coeff\ (vec\text{-}to\text{-}lpoly\ \ A)\ a * X\ a)$
proof −
consider (*eq*) $dim\text{-}vec\ A = dim\text{-}poly\ (vec\text{-}to\text{-}lpoly\ A)$ |
(*le*) $dim\text{-}vec\ A > dim\text{-}poly\ (vec\text{-}to\text{-}lpoly\ A)$
using *vec-to-poly-dim-less*[*of A*] **by** *fastforce*
then show *?thesis*
proof (*cases*)
case *le*
define *dp* **where** *dp*: $dp = dim\text{-}poly\ (vec\text{-}to\text{-}lpoly\ A)$
have $(\sum a = 0..<dim\text{-}vec\ A.\ coeff\ (vec\text{-}to\text{-}lpoly\ A)\ a * X\ a) =$
$(\sum a = 0..<dp.\ coeff\ (vec\text{-}to\text{-}lpoly\ A)\ a * X\ a) +$
$(\sum a = dp..<dim\text{-}vec\ A.\ coeff\ (vec\text{-}to\text{-}lpoly\ A)\ a * X\ a)$

```
    by (metis (no-types, lifting) dp vec-to-poly-dim-less sum.atLeastLessThan-concat
zero-le)
    also have ... = (∑ a = 0..<dp. coeff (vec-to-lpoly A) a * X a)
      using all-greater-dim-poly-zero by (simp add: dp)
    also have ... = (∑ 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 ≠ 0
  shows v $ i ≠ 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 ∀ i. coeff ?lhs i = coeff ?rhs i
  proof
    fix i
    consider (le) i < dim-vec v | (ge) i ≥ 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: ¬ 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)
          not-less-zero vec-to-lin-poly-coeff-access v zero-less-diff calculation)
    moreover have va ≠ 0 using v na by linarith
    ultimately show False
      using ge by auto
  qed
  then show coeff (vec-to-lpoly v) i = coeff (vec-to-lpoly (v @v 0v 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:
(vec-to-lpoly v) {| x |} = v · (vec (dim-vec v) x)
proof −
 have (vec-to-lpoly v) {| x |} = (∑ i∈{0..<dim-vec v}. coeff (vec-to-lpoly v) i * x
i)
  using eval-poly-with-sum-superset[of {0..<dim-vec v} vec-to-lpoly v x]
   vars-subset-dim-vec-to-lpoly-dim by blast
 also have ... = (∑ i∈{0..<dim-vec v}. v$i * x i)
  using list-to-lpoly-coeff-nth by auto
 also have ... =  v · (vec (dim-vec v) x)
  unfolding scalar-prod-def by auto
 finally show ?thesis .
qed

lemma dim-poly-of-append-vec:
 dim-poly (vec-to-lpoly (a@v b)) ≤ dim-vec a + 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 ∈ {0..<dim-vec a} ⟹ coeff (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 ∈ {dim-vec a..<dim-vec (a@v b)} ⟹ coeff (vec-to-lpoly (a@v b)) i = b$(i−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: vec-to-lpoly v = poly-of-vec v
proof −
 have ⋀i. i < dim-vec v ⟹ coeff (poly-of-vec v) i = v $ i
  by (simp add: coeff.rep-eq poly-of-vec.rep-eq)
 moreover have ⋀i. i < dim-vec v ⟹ coeff (vec-to-lpoly v) i = v $ i
  by (simp add: vec-to-lin-poly-coeff-access)
```

moreover have $\bigwedge i.\ i \geq$ *dim-vec* $v \Longrightarrow$ *coeff* (*poly-of-vec* v) $i = 0$
by (*simp add*: *coeff.rep-eq poly-of-vec.rep-eq*)
moreover have $\bigwedge i.\ i \geq$ *dim-vec* $v \Longrightarrow$ *coeff* (*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*: *vars* (*vec-to-lpoly* (0_v n @$_v$ v)) $\subseteq$ {n..<n+*dim-vec* v}
proof −
let *?p* = (*vec-to-lpoly* (0_v n @$_v$ v))
have *dim-poly ?p* $\leq$ n+*dim-vec* v
using *dim-poly-of-append-vec*[*of* 0_v n v] **by** *auto*
have *vars* (*vec-to-lpoly* (0_v n @$_v$ v)) $\subseteq$ {0..<n+*dim-vec* v}
using *vars-subset-dim-vec-to-lpoly-dim*[*of* (0_v n @$_v$ v)] **by** *auto*
moreover have $\forall\, i < n.$ *coeff ?p* $i = 0$
using *vec-coeff-append1*[*of* - 0_v n v] **by** *auto*
ultimately show *vars* (*vec-to-lpoly* (0_v n @$_v$ v)) $\subseteq$ {n..<n+*dim-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-row* $A \Longrightarrow$ *matrix-to-lpolies* A ! i = *vec-to-lpoly* (*row* A i)
using *matrix-to-lpolies.simps*[*of* A] **by** *simp*

lemma *outside-of-col-range-is-0*:
assumes $i <$ *dim-row* A **and** $j \geq$ *dim-col* A
shows *coeff* ((*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 $x \in$ *set* (*matrix-to-lpolies* A)
assumes $j \geq$ *dim-col* A
shows *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-row* A
shows (*matrix-to-lpolies* A)!i = *vec-to-lpoly* (*row* A i)
by (*simp add*: *assms*)

lemma *matrix-to-lpolies-coeff-access*:
assumes $i <$ *dim-row* A **and** $j <$ *dim-col* A
shows *coeff* (*matrix-to-lpolies* A ! i) $j = A$ $$ (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**
lin-polies-to-mat lst = *mat* (*length lst*) (*max-dim-poly-list lst*) (λ(x,y).*coeff* (*lst*!x) y)

lemma *lin-polies-to-rat-mat-coeff-index*:
assumes $i <$ *length* L **and** $j <$ (*max-dim-poly-list* L)
shows *coeff* (L ! i) j = (*lin-polies-to-mat* L) $$ (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-vec* y = *dim-vec* x
shows (*vec-to-lpoly* y) {| ($)$x$ |} = $y \cdot x$
proof −
let *?p* = *vec-to-lpoly* y
have *2*: *?p*{| ($)$x$ |} = (∑j∈*vars?p*. *coeff ?p* j ∗ $x$$$j$)
using *eval-poly-with-sum*[*of ?p* ($)$x$] **by** *blast*
have *vars ?p* ⊆ {*0*..<*dim-vec* y}
using *vars-subset-dim-vec-to-lpoly-dim* **by** *blast*
have *2*: *?p*{| ($)$x$ |} = (∑j∈*vars?p*. *coeff ?p* j ∗ $x$$$j$)
using *eval-poly-with-sum*[*of ?p* ($)$x$] **by** *blast*
also have ∗: ... = (∑i∈{*0*..<*dim-poly ?p*}. *coeff ?p* i ∗ $x$$$i$)
using *valuate-with-dim-poly* **by** (*metis* (*no-types*, *lifting*) *calculation sum.cong*)

also have ... = $y \cdot x$
proof −
have ⋀j. $j <$ *dim-vec* x ⟹ *coeff* (*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-row* A
assumes *dim-col* A = *dim-vec* x
shows (*matrix-to-lpolies* A!i) {| ($)$x$ |} = (*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 $i < dim\text{-}row\ A$
assumes $dim\text{-}col\ A = dim\text{-}vec\ x$
shows $(matrix\text{-}to\text{-}lpolies\ A!i)\ \{\!|\ (\lambda i.\ (if\ i < dim\text{-}vec\ x\ then\ x\$i\ else\ 0))\ |\!\} = (row\ A\ i) \cdot x$
proof −
have $\bigwedge j.\ j < dim\text{-}vec\ x \Longrightarrow x\$j = (\lambda i.\ (if\ i < dim\text{-}vec\ x\ then\ x\$i\ else\ 0))\ j$
by *simp*
let $?p = (matrix\text{-}to\text{-}lpolies\ A!i)$
have $\bigwedge j.\ coeff\ (matrix\text{-}to\text{-}lpolies\ A!i)\ j \neq 0 \Longrightarrow j < dim\text{-}vec\ x$
using *outside-of-col-range-is-0*[*of i A*] *assms*(*1*) *assms*(*2*) *leI* **by** *auto*
then have *subs*: $vars\ ?p \subseteq \{0..<dim\text{-}vec\ x\}$
using ‹$\bigwedge j.\ Abstract\text{-}Linear\text{-}Poly.coeff\ (matrix\text{-}to\text{-}lpolies\ A\ !\ i)\ j \neq 0 \Longrightarrow j < dim\text{-}vec\ x$› *atLeastLessThan-iff coeff-zero* **by** *blast*
then have ∗: $\bigwedge j.\ j \in vars\ ?p \Longrightarrow x\$j = (\lambda i.\ (if\ i < dim\text{-}vec\ x\ then\ x\$i\ else\ 0))\ j$
by (*simp add*: ‹$\bigwedge j.\ Abstract\text{-}Linear\text{-}Poly.coeff\ (matrix\text{-}to\text{-}lpolies\ A\ !\ i)\ j \neq 0 \Longrightarrow j < dim\text{-}vec\ x$› *coeff-zero*)
have $row\ A\ i \cdot x = (?p\ \{\!|\ (\$)\ x\ |\!\})$
using *assms*(*1*) *assms*(*2*) *matrix-to-lpolies-valuate-scalarP*[*of i A x*] **by** *linarith*
also have $... = (\sum j \in vars\ ?p.\ coeff\ ?p\ j * x\$j)$
using *eval-poly-with-sum* **by** *blast*
also have $... = (\sum j \in vars\ ?p.\ coeff\ ?p\ j * (\lambda i.\ (if\ i < dim\text{-}vec\ x\ then\ x\$i\ else\ 0))\ j)$
by (*metis* (*full-types*, *hide-lams*) ‹$\bigwedge j.\ Abstract\text{-}Linear\text{-}Poly.coeff\ (matrix\text{-}to\text{-}lpolies\ A\ !\ i)\ j \neq 0 \Longrightarrow j < dim\text{-}vec\ x$› *mult.commute mult-zero-right*)
also have $... = (?p\ \{\!|\ (\lambda i.\ (if\ i < dim\text{-}vec\ x\ then\ x\$i\ else\ 0))\ |\!\})$
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 = [*GEQ* (*lp-monom 1* (*i*+*index*)) (*b*$*i*). *i* ←

$[0..<\textit{dim-vec}\ b]]$

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

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

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

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

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

lemma *x-mat-eq-vec-sol*:
assumes $x \models_{cs} set\ (x\text{-}mat\text{-}eq\text{-}vec\ b\ A)$
assumes $i < dim\text{-}vec\ b$
shows $((matrix\text{-}to\text{-}lpolies\ A)!i)\ \{\!|\ x\ |\!\}\ = b\$i$
proof −
have *e-e*: $EQ\ ((matrix\text{-}to\text{-}lpolies\ A)!i)\ (b\$i) \in set\ (x\text{-}mat\text{-}eq\text{-}vec\ b\ A)$
by (*simp add*: *assms*(*2*))
then have $x \models_c EQ\ ((matrix\text{-}to\text{-}lpolies\ A)!i)\ (b\$i)$
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 = (*let z1* = 0_m (*dim-row A*) (*dim-col B*);
z2 = 0_m (*dim-row B*) (*dim-col A*) *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-lst*[*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* < *dim-row A*
and *j* ≥ *dim-col A*
and *j* < *dim-col* (*two-block-non-interfering A B*)
shows (*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* <*dim-row A*
shows *row* (*two-block-non-interfering A B*) *i* = *row A i* @$_v$ (0_v (*dim-col B*))
using *assms* **by** *auto*

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

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

lemma *first-vec-two-block-non-inter-is-first-vec*:
assumes *dim-col A* + *dim-col B* = *dim-vec v*
assumes *dim-row A* = *n*
shows *vec-first* (*two-block-non-interfering A B* $*_v$ *v*) *n* = *A* $*_v$ (*vec-first v* (*dim-col A*))
proof
fix *i*
assume *a*: *i* < *dim-vec* (*A* $*_v$ *vec-first v* (*dim-col A*))
let *?tb* = *two-block-non-interfering A B*
have *i-n*: *i* < *n* **using** *a assms*(*2*) **by** *auto*
have *vec-first* (*?tb* $*_v$ *v*) *n* $ *i* = *vec-first* (*vec* (*dim-row ?tb*) (λ *i*. *row ?tb i* · *v*)) *n* $ *i*
unfolding *mult-mat-vec-def* **by** *simp*
also have ... = (*vec n* (λ *i*. *row ?tb i* · *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 ... = *row ?tb i* · *v* **by** (*simp add*: *i-n*)
also have ... = (*row A i* $@_v$ 0_v (*dim-col B*)) · *v*
using *assms*(*2*) *i-n two-block-non-interfering-row-comp1* **by** *fastforce*
also have ... = *row A i* · *vec-first v* (*dim-vec* (*row A i*)) +
0_v (*dim-col B*) · *vec-last v* (*dim-vec* (0_v (*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 *vec-first* (*two-block-non-interfering A B* $*_v$ *v*) *n* $ *i* =
row A i · *vec-first v* (*dim-vec* (*row A i*))
using *calculation* **by** *auto*
then show *vec-first* (*two-block-non-interfering A B* $*_v$ *v*) *n* $ *i* =
(*A* $*_v$ *vec-first v* (*dim-col A*)) $ *i*
by (*simp add*: *assms*(*2*) *i-n*)
next
have *dim-vec* (*A* $*_v$ *v*) = *dim-row A* **using** *dim-vec-def dim-mult-mat-vec*[*of A v*] **by** *auto*
then have *dim-vec* (*vec-first* (*two-block-non-interfering A B* $*_v$ *v*) *n*) = *n* **by** *auto*
then show *dim-vec* (*vec-first* (*two-block-non-interfering A B* $*_v$ *v*) *n*) =
dim-vec (*A* $*_v$ *vec-first v* (*dim-col A*))
by (*simp add*: *assms*(*2*))
qed

lemma *last-vec-two-block-non-inter-is-last-vec*:
assumes *dim-col A* + *dim-col B* = *dim-vec v*
assumes *dim-row B* = *n*
shows *vec-last* ((*two-block-non-interfering A B*) $*_v$ *v*) *n* = *B* $*_v$ (*vec-last v*

```
(dim-col B))
proof
  fix i
  assume a: i < dim-vec (B *v vec-last v (dim-col B))
  let ?tb = two-block-non-interfering A B
  let ?vl = (vec (dim-row ?tb) (λ i. row ?tb i · v))
  have i-n: i < n using  assms(2) using a by auto
  have in3: (dim-row ?tb) − n + i ≥ dim-row A
    by (simp add: assms(2))
  have in3': (dim-row ?tb) − n + i < dim-row ?tb
    by (simp add: assms(2) i-n two-block-non-interfering-dims(1))
  have dim-row A + n = dim-row (two-block-non-interfering A B)
    by (simp add: assms(2) two-block-non-interfering-dims(1))
  then have dim-a: dim-row A = dim-row (two-block-non-interfering A B) − n
    by (metis (no-types) diff-add-inverse2)
  have vec-last (?tb *v v) n $ i = vec-last (vec (dim-row ?tb) (λ i. row ?tb i · v))
n $ i
    unfolding mult-mat-vec-def by auto
  also have ... = ?vl $ (dim-vec ?vl − n + i)
    unfolding vec-last-def using i-n index-vec by blast
  also have ... = row ?tb ((dim-row ?tb) − n + i) · v
   unfolding index-vec by (simp add: assms(2) i-n two-block-non-interfering-dims(1))
  also have ... = row B i · vec-last v (dim-vec (row B i))
  proof −
    have dim-vec (0v (dim-col A) @v row B i) = dim-vec v
      by (simp add: ‹dim-col A + dim-col B = dim-vec v›)
   then show ?thesis using dim-a assms(1) in3' two-block-non-interfering-row-comp2
       append-split-vec-distrib-scalar-prod[of 0v (dim-col A) row B i v]
     by (metis add.commute add.right-neutral diff-add-inverse
        in3 index-zero-vec(2) scalar-prod-left-zero  vec-first-carrier)
  qed
  also have ... = row B i · vec-last v (dim-col B) by simp
   thus vec-last (two-block-non-interfering A B *v v) n $ i = (B *v vec-last v
(dim-col B)) $ i
    using assms(2) calculation i-n by auto
qed (simp add: assms(2))

lemma two-block-non-interfering-mult-decomposition:
  assumes dim-col A + dim-col B = dim-vec v
  shows two-block-non-interfering A B *v v =
        A *v vec-first v (dim-col A) @v B *v vec-last v (dim-col B)
proof −
  let ?tb = two-block-non-interfering A B
  from first-vec-two-block-non-inter-is-first-vec[of A B v dim-row A, OF assms]
  have vec-first (?tb *v v) (dim-row A) = A *v vec-first v (dim-col A)
    by blast
  moreover from last-vec-two-block-non-inter-is-last-vec[of A B v dim-row B, OF
assms]
  have vec-last (?tb *v v) (dim-row B) = B *v vec-last v (dim-col B)
```

by *blast*
ultimately show *?thesis* **using** *vec-first-last-append*[*of ?tb* $*_v$ *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^T) *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*$*i*)
proof −
define *lst* **where** *lst*: *lst* = (*mat-leqb-eqc A b c*)
define *l* **where** *l*: *l* = *matrix-to-lpolies* (*two-block-non-interfering A* A^T)
have *ileqA*: *i* < *dim-row A* **using** *assms* **by** *auto*
have *l*!*i* = *vec-to-lpoly* ((*row A i*)@$_v$ 0_v (*dim-row A*))
unfolding *l* **using** *two-block-non-interfering-row-comp1*[*of i A* A^T, *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*: *l*!*i* = (*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* (*l*!*i*) (*b*$*i*) . *i* <− [*0*..<*dim-vec b*]] @
[*EQ* (*l*!*i*) ((*b*@$_v$*c*)$*i*) . *i* <− [*dim-vec b* ..< *dim-vec* (*b*@$_v$*c*)]]
unfolding *l lst* **by** (*metis mat-leqb-eqc.simps*)
have ([*LEQ* (*l*!*i*) (*b*$*i*). *i* <− [*0*..<*dim-vec b*]] @
[*EQ* (*l*!*i*) ((*b*@$_v$*c*)$*i*). *i* <− [*dim-vec b* ..< *dim-vec* (*b*@$_v$*c*)]]) ! *i* =
[*LEQ* (*l*!*i*) (*b*$*i*). *i* <− [*0*..<*dim-vec b*]]!*i*
using *assms*(*2*) *lst* **by** (*simp add*: *assms*(*1*) *nth-append*)
also have ... = *LEQ* (*l*!*i*) (*b*$*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* ≤ *i* **and** *i* < *dim-vec* (*b*@$_v$*c*)
assumes *dim-row A* = *dim-vec b* **and** *dim-col A* ≥ *dim-vec c*
shows (*mat-leqb-eqc A b c*)!*i* =

```
    EQ (vec-to-lpoly (0_v (dim-col A) @_v row A^T (i−dim-vec b))) (c$(i−dim-vec
b))
proof −
  define lst where lst: lst = (mat-leqb-eqc A b c)
  define l where l: l = matrix-to-lpolies (two-block-non-interfering A A^T)
  have i-s: i < dim-row (two-block-non-interfering A A^T)
   using assms by (simp add: assms(2) assms(3) two-block-non-interfering-dims(1))
  have l':l!i = vec-to-lpoly ((0_v (dim-col A)) @_v (row A^T (i − dim-vec b)))
    using l two-block-non-interfering-row-comp2[of i A A^T, OF i-s]
      assms(1) assms(3) i-s matrix-to-lp-vec-to-lpoly-row by presburger
  have ([LEQ (l!i) (b$i) . i <− [0..<dim-vec b]] @
                    [EQ (l!i) ((b@_v c)$i) . i <− [dim-vec b ..< dim-vec (b@_v c)]])!i
=
       [EQ (l!i) ((b@_v c)$i) . i <− [dim-vec b..< dim-vec (b@_v c)]] ! (i − dim-vec
b)
    by (simp add: assms(1) leD nth-append)
  also have ... = EQ (l!i) ((b@_v c)$i)
    using assms(1) assms(2) by auto
  also have ... = EQ (l!i) (c$(i−dim-vec b))
    using assms(1) assms(2) by auto
  then show ?thesis
    using mat-leqb-eqc.simps by (metis (full-types) calculation l l')
qed

lemma mat-leqb-eqc-satisfies1:
  assumes x ⊨_cs set (mat-leqb-eqc 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-eqc A b c)
    using mat-leqb-eqc-for-LEQ[of i b A c, OF assms(2) assms(3)]
      nth-mem[of i matrix-to-lpolies A] mat-leqb-eqc.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-eqc-satisfies2:
  assumes x ⊨_cs set (mat-leqb-eqc A b c)
  assumes dim-vec b ≤ i and i < dim-vec (b@_v c)
    and dim-row A = dim-vec b and dim-vec c ≤ dim-col A
  shows (matrix-to-lpolies (two-block-non-interfering A A^T) ! i) {|x|} = (b @_v c) $
i
proof −
```

have *e-e*: *EQ* (*vec-to-lpoly* (*0*$_v$ (*dim-col A*) @$_v$ *row* A^T (*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* ⊨$_c$ *EQ* (*vec-to-lpoly* (*0*$_v$ (*dim-col A*) @$_v$ *row* A^T (*i* − *dim-vec b*))) (*c* $ (*i* − *dim-vec b*))
using *assms*(*1*) **by** *blast*
have ∗: *i* < *dim-row* (*two-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-row A* ≤ *i*
by (*simp add*: *assms*(*2*) *assms*(*4*))
then have *x* ⊨$_c$ *EQ* ((*matrix-to-lpolies* (*two-block-non-interfering A* A^T))!*i*) ((*b*@$_v$*c*)$*i*)
using *two-block-non-interfering-row-comp2*[*of i A* A^T, *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*@$_v$*c*)
and *dim-row A* = *dim-vec b* **and** *dim-vec c* ≤ *dim-col A*
shows (*matrix-to-lpolies* (*two-block-non-interfering A* A^T) ! *i*) {|⟨*x*⟩|} = (*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 ⟨*x*⟩ ⊨$_c$ (*index-geq-n i n*)
shows ⟨*x*⟩ *i* ≥ *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* ⊨$_{cs}$ *set* (*from-index-geq0-vector i v*)
j < *dim-vec v*
shows *x* (*i* + *j*) ≥ *v*$*j*
proof −
have *GEQ* (*lp-monom 1* (*i*+*j*)) (*v*$*j*)∈ *set* (*from-index-geq0-vector i v*)
by (*simp add*: *assms*(*2*))
moreover have *x* ⊨$_{c}$ *GEQ* (*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 ⟨*x*⟩ ⊨$_{cs}$ *set* (*from-index-geq0-vector i v*)
assumes *i* ≤ *j* **and** *j* < (*dim-vec v*) + *i*
shows ⟨*x*⟩ *j* ≥ *v*$(*j* − *i*)
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *from-index-geq-vector-simplex*
le-add-diff-inverse less-diff-conv2)

fun *x-times-c-geq-y-times-b* **where**
x-times-c-geq-y-times-b c b = *GEQPP* (*vec-to-lpoly* (*c* @$_v$ *0*$_v$ (*dim-vec b*)))
(*vec-to-lpoly* (*0*$_v$ (*dim-vec c*) @$_v$ *b*))

lemma *x-times-c-geq-y-times-b-correct*:
assumes *simplex* [*x-times-c-geq-y-times-b c b*] = *Sat x*
shows ((*vec-to-lpoly* (*c* @$_v$ *0*$_v$ (*dim-vec b*))) {⦇ ⟨*x*⟩ ⦈} ≥
((*vec-to-lpoly* (*0*$_v$ (*dim-vec c*) @$_v$ *b*)) {⦇ ⟨*x*⟩ ⦈})
using *assms simplex*(*3*) **by** *fastforce*

definition *split-i-j-x* **where**
split-i-j-x i j x = (*vec i* ⟨*x*⟩, *vec* (*j* − *i*) (λ*y*. ⟨*x*⟩ (*y*+*i*)))

abbreviation *split-n-m-x* **where**
split-n-m-x n m x ≡ *split-i-j-x n* (*n*+*m*) *x*

lemma *split-vec-dims*:
assumes *split-i-j-x i j x* = (*a* ,*b*)
shows *dim-vec a* = *i dim-vec b* = (*j* − *i*)

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* = ⟨*x*⟩ *k*
by (*metis Pair-inject assms(1) assms(2) index-vec split-i-j-x-def*)

lemma *split-access-snd-1*:
assumes *i* ≤ *k* **and** *k* < *j*
assumes *split-i-j-x i j x* = (*a*, *b*)
shows *b* $ (*k* − *i*) = ⟨*x*⟩ *k*
proof −
have *vec* (*j* − *i*) (λ*n*. ⟨*x*⟩ (*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*) = *split-i-j-x i j Z*
assumes *k* < *dim-vec x*
shows *x*$*k* = ⟨*Z*⟩ *k*
by (*metis assms(1) assms(2) split-access-fst-1 split-vec-dims(1)*)

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

lemma *from-index-geq0-vector-split-snd*:
assumes ⟨*X*⟩ ⊨$_{cs}$ *set* (*from-index-geq0-vector d v*)
assumes (*x*, *y*) = *split-n-m-x d m X*
shows ⋀*i*. *i* < *dim-vec v* ⟹ *i* < *m* ⟹ *y*$*i* ≥ *v*$*i*
using *assms* **unfolding** *split-i-j-x-def*
using *from-index-geq-vector-simplex*[*of d v* ⟨*X*⟩ -] *index-vec* **by** (*simp add*: *add.commute*)

lemma *split-coeff-vec-index-sum*:
assumes (*x*,*y*) = *split-i-j-x* (*dim-vec* (*lpoly-to-vec v*)) *l X*
shows (∑ *i* = *0*..<*dim-vec x*. *Abstract-Linear-Poly.coeff v i* ∗ ⟨*X*⟩ *i*) =
(∑ *i* = *0*..<*dim-vec x*. *lpoly-to-vec v* $ *i* ∗ *x* $ *i*)

proof −
from *valuate-with-dim-poly*[*of v* ⟨*X*⟩, *symmetric*]
have (∑ *i* = *0*..<*dim-vec x*. (*lpoly-to-vec v*) $ *i* ∗ ⟨*X*⟩ *i*) =
(∑ *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*) · *x* = (*v* ⦃⟨*X*⟩⦄)
proof −
from *valuate-with-dim-poly*[*of v* ⟨*X*⟩, *symmetric*]
have *1*: (*v* ⦃⟨*X*⟩⦄) = (∑ *i* = *0*..<*dim-poly v*. *Abstract-Linear-Poly.coeff v i* ∗
⟨*X*⟩ *i*) **by** *simp*
have (∑ *i* = *0*..<*dim-vec x*. (*lpoly-to-vec v*) $ *i* ∗ ⟨*X*⟩ *i*) =
(∑ *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 ... = (*lpoly-to-vec v*) · *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* ([-∗$_v$-]≤- [*1000*,*1000*,*100*])
where
[*A* ∗$_v$ *x*]≤*b* ⟷ (∀ *i* < *dim-vec b*. (*A* ∗$_v$ *x*)$*i* ≤ *b*$*i*) ∧
(*dim-row A* = *dim-vec b*) ∧
(*dim-col A* = *dim-vec x*)

definition *vec-times-mat-eq* ([-$_v$∗-]=- [*1000*,*1000*,*100*])
where
[*y* $_v$∗ *A*]=*c* ⟷ (∀ *i* < *dim-vec c*. (*A*^{T} ∗$_v$ *y*)$*i* = *c*$*i*) ∧
(*dim-col A*^{T} = *dim-vec y*) ∧
(*dim-row A*^{T} = *dim-vec c*)

definition *vec-times-mat-leq* ([-$_v$∗-]≤- [*1000*,*1000*,*100*])
where
[*y* $_v$∗ *A*]≤*c* ⟷ (∀ *i* < *dim-vec c*. (*A*^{T} ∗$_v$ *y*)$*i* ≤ *c*$*i*) ∧
(*dim-col A*^{T} = *dim-vec y*) ∧
(*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 $\bigwedge i.\ i < dim\text{-}vec\ b \Longrightarrow (A *_v x)\$i \leq b\$i$
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-row A = dim-vec b dim-col A = dim-vec x* $\bigwedge i.\ i < dim\text{-}vec\ b \Longrightarrow (A *_v x)\$i \leq b\$i$
using *assms mat-times-vec-leq-def* **by** *blast+*

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

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

lemma *mat-leqb-eqc-split-correct1*:
assumes *dim-vec b = dim-row A*
assumes $\langle X \rangle \models_{cs} set\ (mat\text{-}leqb\text{-}eqc\ A\ b\ c)$
assumes *(x,y) = split-i-j-x (dim-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* $\langle X \rangle$ *i*]
have *m*: $(matrix\text{-}to\text{-}lpolies\ A\ !\ i)\ \{\!|\ \langle X \rangle\ |\!\} \leq b\ \$\ i$
using *assms(1) assms(2)* **by** *linarith*
have *leq*: $dim\text{-}poly\ (vec\text{-}to\text{-}lpoly\ (row\ A\ i)) \leq dim\text{-}col\ A$
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* = *row A i* $@_v$ 0_v (*dim-col* A^T)
using *3 assms*(*1*) **by** *linarith*
have (*vec-to-lpoly* (*row A i* $@_v$ 0_v (*dim-col* A^T))) $\{\!|\langle X \rangle|\!\}$ = ((*vec-to-lpoly* (*row A i*)) $\{\!|\langle X \rangle|\!\}$)
using *lpoly-of-v-equals-v-append0* **by** *auto*
also have ... = ($\sum$ *a* = *0..<dim-poly* (*vec-to-lpoly* (*row A i*)).
Abstract-Linear-Poly.coeff (*vec-to-lpoly* (*row A i*)) *a* ∗ ⟨*X*⟩ *a*)
using *valuate-with-dim-poly*[*of vec-to-lpoly* (*row A i*) ⟨*X*⟩] **by** *blast*
also have ... = ($\sum$ *a* = *0..<dim-col A*. *Abstract-Linear-Poly.coeff* (*vec-to-lpoly* (*row A i*)) *a* ∗ ⟨*X*⟩ *a*)
using *split-coeff-vec-index-sum*[*of x y*]
sum-dim-vec-equals-sum-dim-poly[*of row A i* ⟨*X*⟩] **by** *auto*
also have ... = *row A i* ⋅ *x*
unfolding *scalar-prod-def* **using** ‹*dim-col A* = *dim-vec x*› *i assms*(*3*)
using *matrix-to-lpolies-coeff-access*[*of i A*] *matrix-to-lp-vec-to-lpoly-row*[*of i A*]
split-access-fst-1[*of* - (*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 *dim-vec b* = *dim-row A*
assumes *simplex* (*mat-leqb-eqc A b c*) = *Sat X*
assumes (*x*,*y*) = *split-i-j-x* (*dim-col A*) *l X*
shows [*A* $*_v$ *x*]≤*b*
using *mat-leqb-eqc-split-correct1*[*of b A c X x y*] *assms*(*1*) *assms*(*2*) *assms*(*3*) *simplex*(*3*) **by** *blast*

lemma *sat-mono*:
assumes *set A* ⊆ *set B*
shows ⟨*X*⟩ $\models_{cs}$ *set B* ⟹ ⟨*X*⟩ $\models_{cs}$ *set A*
using *assms*(*1*) *assms* **by** *blast*

lemma *mat-leqb-eqc-split-subset-correct1*:
assumes *dim-vec b* = *dim-row A*
assumes *set* (*mat-leqb-eqc A b c*) ⊆ *set S*
assumes *simplex S* = *Sat X*
assumes (*x*,*y*) = *split-i-j-x* (*dim-col A*) *l X*
shows [*A* $*_v$ *x*]≤*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 *dim-vec c* = *dim-row* A^T
assumes *dim-vec b* = *dim-col* A^T
assumes ⟨*X*⟩ $\models_{cs}$ *set* (*mat-leqb-eqc A b c*)
assumes (*x*, *y*) = *split-n-m-x* (*dim-row* A^T) (*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* A^T)
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*$*i*) . *i* <− [*0*..<*dim-vec b*]] @
[*EQ* (*lst*!*i*) ((*b*@$_v$*c*)$*i*) . *i* <− [*dim-vec b* ..< *dim-vec* (*b*@$_v$*c*)]]) ! (*db* + *i*) =
EQ (*lst*!(*db*+*i*)) ((*b*@$_v$*c*)$(*db*+*i*)) **using** *mat-leqb-eqc-for-EQ*[*of b db*+*i c A*]
nth-append[*of* [*LEQ* (*lst*!*i*) (*b*$*i*) . *i* <− [*0*..<*dim-vec b*]]
[*EQ* (*lst*!*i*) ((*b*@$_v$*c*)$*i*) . *i* <− [*dim-vec b* ..< *dim-vec* (*b*@$_v$*c*)]]]
by (*simp add*: *3 db*)
have *rowA*: *dim-vec b* = *dim-row A*
using *assms index-transpose-mat*(*3*)[*of A*] **by** *linarith*
have ⟨*X*⟩ ⊨$_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-eqc-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*$*i*
by *simp*
define *V* **where** *V*: *V* = *vec* (*db*+*dc*) (λ*i*. ⟨*X*⟩ *i*)
have *vdim*: *dim-vec V* = *dim-vec* (*b*@$_v$*c*) **using** *V db dc* **by** *simp*
have ∗: *db* + *i* < *dim-row* (*two-block-non-interfering A* A^T)
by (*metis dbi-dim assms*(*1*) *index-append-vec*(*2*) *rowA two-block-non-interfering-dims*(*1*))
have ∗∗: *dim-row A* ≤ *db* + *i*
by (*simp add*: *assms*(*2*) *db*)
from *two-block-non-interfering-row-comp2*[*of db*+*i A* A^T, *OF* ∗ ∗∗]
have *eql*: *row* (*two-block-non-interfering A* A^T) (*db* + *i*) = 0_v (*dim-col A*) @$_v$ *row* A^T *i*
by (*simp add*: *assms*(*2*) *db*)
with *matrix-to-lp-vec-to-lpoly-row*[*of i* A^T]
have *eqv*: *lst*!(*db*+*i*) = *vec-to-lpoly* (0_v (*dim-col A*) @$_v$ *row* A^T *i*)

using $*$ *lst matrix-to-lp-vec-to-lpoly-row* **by** *presburger*
then have $\forall j{<}$*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-to-lin-poly-coeff-access trans-less-add1)
moreover have $\forall j{\geq}db{+}dc$. *Abstract-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-nonzero-dim-vec-non-zero*(2) *index-append-vec*(2)
index-row(2) *index-transpose-mat*(2) *db dc*)
moreover have *vars* (*lst*!($db{+}i$)) $\subseteq$ {*dim-col A..*$<db{+}dc$}
by (*meson atLeastLessThan-iff calculation*(1) *calculation*(2) *coeff-zero not-le
subsetI*)
ultimately have (*lst*!($db{+}i$))$\{\!|\langle X\rangle|\!\}$ = ($\sum j{\in}$\{*dim-col A..*$<db{+}dc$\}. *Abstract-Linear-Poly.coeff*
(*lst*!($db{+}i$)) $j * \langle X\rangle\ j$)
using *eval-poly-with-sum-superset*[*of* {*dim-col A..*$<db{+}dc$} *lst*!($db{+}i$) $\langle X\rangle$] **by**
blast
also have ... = ($\sum j{\in}$\{*dim-col A..*$<db{+}dc$\}. *Abstract-Linear-Poly.coeff* (*lst*!($db{+}i$))
$j * V\$j$)
using V **by** *auto*
also have ... = ($\sum j{\in}$\{*dim-col A..*$<db{+}dc$\}. (0_v (*dim-col A*) $@_v$ *row* A^T i)\$$j$ $*$
$V\$j$)
proof −
have $\forall j{\in}$\{*dim-col A..*$<db{+}dc$\}. *Abstract-Linear-Poly.coeff* (*lst*!($db{+}i$)) j =
(0_v (*dim-col A*) $@_v$ *row* A^T i)\$$j$
by (*metis* ‹$V \equiv$ *vec* (db + dc) $\langle X\rangle$› *vdim assms*(1) *assms*(2) *index-transpose-mat*(2)
atLeastLessThan-iff dim-vec eql eqv index-append-vec(2) *index-row*(2)
vec-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-col A*\}. (0_v (*dim-col A*) $@_v$ *row* A^T i)\$$j$ $*$ $V\$j$)
+
($\sum j{\in}$\{*dim-col A..*$<db{+}dc$\}. (0_v (*dim-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-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-col A*) $@_v$ *row* A^T i) $\cdot$ V
unfolding *scalar-prod-def* **by** (*simp add*: V)
also have ... = 0_v (*dim-col A*) $\cdot$ *vec-first* V (*dim-vec* (0_v (*dim-col A*))) +
row A^T i $\cdot$ *vec-last* V (*dim-vec* (*row* A^T i))
using *append-split-vec-distrib-scalar-prod*[*of* 0_v (*dim-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-row*(*2*) *index-transpose-mat*(*2*)
index-transpose-mat(*3*) *index-zero-vec*(*2*))
also have *0*$_v$ (*dim-col A*) ⋅ *vec-first V* (*dim-vec* (*0*$_v$ (*dim-col A*))) +
row A^T *i* ⋅ *vec-last V* (*dim-vec* (*row A*^{T} *i*)) = (*row A*^{T} *i*) ⋅ *y*
proof −
have *vec-last V* (*dim-vec* (*row A*^{T} *i*)) = *y*
proof (*standard*, *goal-cases*)
case (*1 i*)
then show *?case*
proof −
have *f1*: *dim-col A*^{T} = *db*
by (*simp add*: *assms*(*2*) *db*)
then have ∀ *v va*. *vec db* (λ*n*. ⟨*X*⟩ (*n* + *dc*)) = *v* ∨ (*x*, *y*) ≠ (*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 vec-last-def*
using *split-access-fst-1*[*of* (*dim-row A*^{T}) *i* (*dim-col A*^{T}) *X x y*]
by (*metis 1 add.commute add-diff-cancel-left*′ *add-less-cancel-left*
dim-vec f1 index-row(*2*) *index-vec*)
qed
next
case *2*
then show *?case*
using ‹*dim-col A*^{T} = *dim-vec y*› **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 *dim-vec c* = *dim-row A*^{T}
assumes *dim-vec b* = *dim-col A*^{T}
assumes *simplex* (*mat-leqb-eqc A b c*) = *Sat X*
assumes (*x*, *y*) = *split-n-m-x* (*dim-row A*^{T}) (*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 *dim-vec c* = *dim-row A*^{T}
and *dim-vec b* = *dim-col A*^{T}
assumes *simplex* (*mat-leqb-eqc A b c*) = *Sat X*
assumes (*x*, *y*) = *split-n-m-x* (*dim-row A*^{T}) (*dim-col A*^{T}) *X*
shows [*y* $_v$∗ *A*]=*c* [*A* ∗$_v$ *x*]≤*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*(*3*)
mat-leqb-eqc-split-simplex-correct1[*of b A c X x y*])

lemma *eval-lpoly-eq-dot-prod-split1*:
assumes (*x*, *y*) = *split-n-m-x* (*dim-vec c*) (*dim-vec b*) *X*
shows(*vec-to-lpoly c*) ⟦⟨*X*⟩⟧ = *c* · *x*
proof −
have ∗: (*vec-to-lpoly c*) ⟦⟨*X*⟩⟧ =
(∑ *i*∈*vars* (*vec-to-lpoly c*). *Abstract-Linear-Poly.coeff* (*vec-to-lpoly c*) *i* ∗ ⟨*X*⟩ *i*)
using *linear-poly-sum sum.cong eval-poly-with-sum* **by** *auto*
also have ... = (∑ *i*∈{*0*..<*dim-vec c*}. *Abstract-Linear-Poly.coeff* (*vec-to-lpoly c*) *i* ∗ ⟨*X*⟩ *i*)
using *vars-subset-dim-vec-to-lpoly-dim*[*of c*] *linear-poly-sum*[*of vec-to-lpoly c* ⟨*X*⟩]
eval-poly-with-sum-superset[*of* {*0*..<*dim-vec c*} *vec-to-lpoly c* ⟨*X*⟩] **by** *auto*
also have ... = (∑ *i*∈{*0*..<*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 ... = *c* · *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*) = *split-n-m-x* (*dim-vec b*) (*dim-vec c*) *X*
shows(*vec-to-lpoly* (*0*$_v$ (*dim-vec b*) @$_v$ *c*)) ⟦⟨*X*⟩⟧ = *c* · *y*
proof −
let *?p* = (*vec-to-lpoly* ((*0*$_v$ (*dim-vec b*) @$_v$ *c*)))
let *?v0* = (*0*$_v$ (*dim-vec b*) @$_v$ *c*)
have ∗: ∀ *i*<*dim-vec b*. *Abstract-Linear-Poly.coeff ?p i* = *0*
using *coeff-nonzero-dim-vec-non-zero*(*1*) **by** *fastforce*
have ∗∗: *dim-vec ?v0* = *dim-vec b* + *dim-vec c*
by *simp*
have *?p* ⟦⟨*X*⟩⟧ = (∑ *i*∈*vars ?p*. *Abstract-Linear-Poly.coeff ?p i* ∗ ⟨*X*⟩ *i*)
using *eval-poly-with-sum* **by** *blast*
also have ... = (∑ *i*∈{*0*..<*dim-vec ?v0*}. *Abstract-Linear-Poly.coeff ?p i* ∗ ⟨*X*⟩ *i*)
using *eval-poly-with-sum-superset*[*of* {*0*..<*dim-vec ?v0*} *?p* ⟨*X*⟩] *calculation*
vars-subset-dim-vec-to-lpoly-dim[*of ?v0*] **by** *force*
also have ... = (∑ *i*∈{*0*..<*dim-vec b*}. *Abstract-Linear-Poly.coeff ?p i* ∗ ⟨*X*⟩ *i*)

```
+
                (∑ i∈{(dim-vec b)..<dim-vec ?v0}. Abstract-Linear-Poly.coeff ?p i
* ⟨X⟩ i)
    by (simp add: sum.atLeastLessThan-concat)
  also have ... = (∑ i∈{(dim-vec b)..<dim-vec ?v0}. Abstract-Linear-Poly.coeff
?p i * ⟨X⟩ i)
    using * by simp
  also have ... = (∑ i∈{(dim-vec b)..<dim-vec ?v0}. ?v0$i * ⟨X⟩ i)
    using vec-to-lin-poly-coeff-access by auto
  also have ... = (∑ i∈{0..<dim-vec c}. ?v0$(i+dim-vec b) * ⟨X⟩ (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 (λi. ?v0$i * ⟨X⟩ i) 0 dim-vec b dim-vec c]
    by (simp add: add.commute)
  also have ... = (∑ i∈{0..<dim-vec c}. c$i * ⟨X⟩ (i+dim-vec b))
    by auto
  also have ... = (∑ i∈{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 · 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 ⟨X⟩ ⊨_c x-times-c-geq-y-times-b c b
  assumes (x, y) = split-n-m-x (dim-vec c) (dim-vec b) X
  shows c · x ≥ b · 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

lemma mult-right-leq:
  fixes A :: ('a::{comm-semiring-1,ordered-semiring}) mat
  assumes dim-vec y = dim-vec b
    and ∀ i < dim-vec y. y$i ≥ 0
    and [A *_v x]≤ b
  shows (A *_v x) · y ≤ b · y
proof −
  have (∑ n<dim-vec b. (A *_v x) $ n * y $ n) ≤ (∑ 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 - - λi. (A^T *_v y) $ i * x $ i λi. c $ i * x $ i*]
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *assms*(*2*))

lemma *soundness-mat-x-leq*:
assumes *dim-row A = dim-vec b*
assumes *simplex* (*mat-x-leq-vec A b*) = *Sat X*
shows ∃ *x*. [*A *_v x*]≤*b*
proof
define *x* **where** *x*: *x* = *fst* (*split-n-m-x* (*dim-col A*) (*dim-row A*) *X*)
have ∗: *dim-vec x* = *dim-col A* **by** (*simp add*: *split-i-j-x-def x*)
have ∀ *i*<*dim-vec b*. (*A *_v x*) $ *i* ≤ *b* $ *i*
proof (*standard*, *standard*)
fix *i*
assume *i* < *dim-vec b*
have *row A i* · *x* ≤ *b*$*i*
using *mat-x-leq-vec-sol*[*of A b X i*]
by (*metis* ‹*i* < *dim-vec b*› *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 *_v x*) $ *i* ≤ *b* $ *i*
by (*simp add*: ‹*i* < *dim-vec b*› *assms*(*1*))
qed
then show [*A *_v x*]≤*b*
using *mat-times-vec-leqI*[*of A b x*, *OF assms*(*1*) ∗[*symmetric*]] **by** *auto*
qed

lemma *completeness-mat-x-leq*:
assumes ∃ *x*. [*A *_v x*]≤*b*
shows ∃ *X*. *simplex* (*mat-x-leq-vec A b*) = *Sat X*
proof (*rule ccontr*)
assume *1*: ∄ *X*. *simplex* (*mat-x-leq-vec A b*) = *Inr X*
have ∗: ∄ *v*. *v* ⊨*_cs set* (*mat-x-leq-vec A b*)
using *simplex*(*1*)[*of mat-x-leq-vec A b*] **using** *1 sum.exhaust-sel* **by** *blast*
then have *dim-vec b* = *dim-row A* **using** *assms mat-times-vec-leqD*(*1*)[*of A - b*]
by *auto*
then obtain *x* **where** *x*: [*A *_v x*]≤*b*
using *assms* **by** *blast*
have *x-A*: *dim-vec x* = *dim-col A*
using *x* **by** *auto*
define *v* **where** *v*: *v* = (λ*i*. (*if i* < *dim-vec x then x*$*i else 0*))
have *v-d*: ∀ *i* < *dim-vec x*. *x*$*i* = *v i*
by (*simp add*: *v*)
have *v* ⊨*_cs set* (*mat-x-leq-vec A b*)
proof
fix *c*
assume *c* ∈ *set* (*mat-x-leq-vec A b*)

```
    then obtain i where i: c = LEQ (matrix-to-lpolies A!i) (b$i) i < dim-vec b
      by auto
    let ?p = matrix-to-lpolies A!i
    have 2: ?p⦃ v ⦄ = (row A i) · x
      using matrix-to-lpolies-lambda-valuate-scalarP[of i A x] v
      by (metis ‹dim-vec b = dim-row A› i(2) x-A)
    also have ... ≤ b$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 A^T = dim-vec c
  assumes simplex (x-mat-eq-vec c A^T) = Sat X
  shows ∃ x. [x _v∗ A]=c
proof
  define x where x: x = fst (split-n-m-x (dim-col A^T) (dim-row A^T) X)
  have dim-vec x = dim-col A^T
   unfolding split-i-j-x-def using split-vec-dims(1)[of (dim-col A^T) - X x] fst-conv[of
x]
    by (simp add: split-i-j-x-def x)
  have ∀ i < dim-vec c. (A^T ∗_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 A^T)
      using assms(2) simplex(3) by blast
    then have row A^T i · x = c$i
      using x-mat-eq-vec-sol[of c A^T ⟨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 (A^T ∗_v x) $ i = c $ i
      unfolding mult-mat-vec-def using a assms(1) by auto
  qed
  then show [x _v∗ A]=c
    using mat-times-vec-eqI[of A x c, OF ‹dim-vec x = dim-col A^T›[symmetric]
assms(1)] by auto
qed

lemma completeness-mat-x-eq-vec:
  assumes ∃ x. [x _v∗ A]=c
  shows ∃ X. simplex (x-mat-eq-vec c A^T) = Sat X
proof (rule ccontr)
  assume 1: ∄ X. simplex (x-mat-eq-vec c A^T) = Inr X
```

```
  then have ∗: ∄v. v ⊨cs set (x-mat-eq-vec c A^T)
    using simplex(1)[of x-mat-eq-vec c A^T] 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 v∗ A]=c using assms by auto
  then have dim-vec x = dim-col A^T using assms
    by (metis ‹[x v∗ A]=c› vec-times-mat-eq-def)
  define v where v: v = (λi. (if i < dim-vec x then x$i else 0))
  have v-d: ∀ i < dim-vec x. x$i = v i
    by (simp add: v)
  have v ⊨cs set (x-mat-eq-vec c A^T)
  proof
    fix a
    assume a ∈ set (x-mat-eq-vec c A^T)
    then obtain i where i: a = EQ (matrix-to-lpolies A^T!i) (c$i) i < dim-vec c
      by (metis (no-types, lifting) add-cancel-right-left diff-zero in-set-conv-nth
length-map length-upt nth-map-upt x-mat-eq-vec.simps)
    let ?p = matrix-to-lpolies A^T!i
    have 2: ?p⟪ v ⟫ = (row A^T i) · x
      using matrix-to-lpolies-lambda-valuate-scalarP[of i A^T x] v
    by (metis ‹dim-vec c = dim-col A› ‹dim-vec x = dim-col A^T› i(2) index-transpose-mat(2))
    also have ... = c$i
      by (metis ‹[x v∗ A]=c› ‹dim-vec c = dim-col A› i(2) index-mult-mat-vec
index-transpose-mat(2) vec-times-mat-eqD(1))
    finally show v ⊨c a
      using i(1) satisfies-constraint.simps(5)[of v (matrix-to-lpolies A^T ! i) (c $
i)] by simp
  qed
  then show False
    using ∗ by blast
qed

lemma soundness-mat-leqb-eqc1:
  assumes dim-row A = dim-vec b
  assumes simplex (mat-leqb-eqc A b c) = Sat X
  shows ∃x. [A ∗v x]≤b
proof
  define x where x: x = fst (split-n-m-x (dim-col A) (dim-row A) X)
  have ∗: dim-vec x = dim-col A by (simp add: split-i-j-x-def x)
  have ∀ i<dim-vec b. (A ∗v x) $ i ≤ b $ i
  proof (standard, standard)
    fix i
    assume i < dim-vec b
    have row A i · x ≤ b$i
      using mat-x-leq-vec-sol[of A b X i]
      by (metis ‹i < dim-vec b› assms(1) assms(2) fst-conv split-i-j-x-def x
        index-mult-mat-vec mat-leqb-eqc-split-simplex-correct1 mat-times-vec-leqD(3))
    then show (A ∗v x) $ i ≤ b $ i
      by (simp add: ‹i < dim-vec b› assms(1))
```

```
  qed
  then show [A *v x]≤b
    using mat-times-vec-leqI[of A b x, OF assms(1) *[symmetric]] by auto
qed

lemma soundness-mat-leqb-eqc2:
  assumes dim-row A^T = dim-vec c
  assumes dim-col A^T = dim-vec b
  assumes simplex (mat-leqb-eqc A b c) = Sat X
  shows ∃ y. [y v* A]=c
proof (standard, intro mat-times-vec-eqI)
  define y where x: y = snd (split-n-m-x (dim-col A) (dim-row A) X)
  have *: dim-vec y = dim-row A by (simp add: split-i-j-x-def x)
  show dim-col A^T = dim-vec y by (simp add: *)
  show dim-row A^T = dim-vec c using assms(1) by blast
  show ⋀i. i < dim-vec c ⟹ (A^T *v y) $ i = c $ i
  proof −
    fix i
    assume a: i < dim-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 ∃ x. [A *v x]≤b
    and ∃ y. [y v* A]=c
  shows ∃ X. simplex (mat-leqb-eqc A b c) = Sat X
proof (rule ccontr)
  assume 1: ∄ X. simplex (mat-leqb-eqc A b c) = Sat X
  have *: ∄ v. v ⊨cs set (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-vec b = dim-row A
    using assms mat-times-vec-leqD(1)[of A - b] by presburger
  then obtain x y where x: [A *v x]≤b [y v* A]=c
    using assms by blast
  have x-A: dim-vec x = dim-col A
    using x by auto
  have yr: dim-vec y = dim-row A
    using vec-times-mat-eq-def x(2) by force
  define v where v: v = (λi. (if i < dim-vec (x@v y) then (x@v y)$i else 0))
  have v-d: ∀ i < dim-vec (x@v y). (x@v y)$i = v i
    by (simp add: v)
  have i-in: ∀ i ∈ {0..< dim-vec y}. y$i = v (i+dim-vec x)
```

```
    by (simp add: v)
  have v ⊨_cs set (mat-leqb-eqc A b c)
  proof
    fix e
    assume asm: e ∈ set (mat-leqb-eqc A b c)
    define lst where lst: lst = matrix-to-lpolies (two-block-non-interfering A A^T)
    let ?L = [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)]]
    have L: mat-leqb-eqc A b c = ?L
      by (metis (full-types) lst mat-leqb-eqc.simps)
    then obtain i where i: e = ?L!i i ∈{0..<length ?L}
      using asm by (metis atLeastLessThan-iff in-set-conv-nth not-le not-less0)
    have ldimbc: length ?L = dim-vec (b@_v c)
      using i(2) by auto
    consider (leqb) i ∈ {0..<dim-vec b} | (eqc) i ∈ {dim-vec b..<length ?L}
      using i(2) leI by auto
    then show v ⊨_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-eqc-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..<dim-vec x} using matrix-to-lpolies-vec-of-row
        by (metis ‹i < dim-row A› eqmp index-row(2)
            vars-subset-dim-vec-to-lpoly-dim x-A)
      have ∗∗: ((lst!i) ⦃ v ⦄) = ((vec-to-lpoly (row A i)) ⦃ v ⦄)
        by (simp add: ‹i < dim-row A› eqmp)
      also have ... = (∑ j∈vars(lst!i). Abstract-Linear-Poly.coeff (lst!i) j ∗ v j)
        using ∗∗ eval-poly-with-sum by auto
      also have ... = (∑ j∈{0..<dim-vec x}. Abstract-Linear-Poly.coeff (lst!i) j ∗
v j)
        using sset eval-poly-with-sum-superset[of {0..<dim-vec x} lst!i v,
            OF finite-atLeastLessThan sset] ∗∗ using calculation by linarith
      also have ... = (∑ j∈{0..<dim-vec x}. Abstract-Linear-Poly.coeff (lst!i) j ∗
x$j)
        using v by (auto split: if-split)
      also have ... = (∑ j∈{0..<dim-vec x}. (row A i)$j ∗ x$j)
        using matrix-to-lpolies-vec-of-row[of i A, OF iA]
          vec-to-lin-poly-coeff-access[of - row A i] index-row(2)[of A i]
          atLeastLessThan-iff by (metis (no-types, lifting) eqmp sum.cong x-A)
      also have  ... = row A i · x unfolding scalar-prod-def by (simp)
```

also have ... $\leq b\$i$
by (*metis* ‹$i <$ *dim-vec* b› *index-mult-mat-vec mat-times-vec-leq-def* $x(1)$)
finally show *?thesis*
by (*simp add*: $*$)
next
case *eqc*
have *igeq*: $i \geq$ *dim-vec* b
using *atLeastLessThan-iff eqc* **by** *blast*
have $*$: $i <$ *length ?L*
using *atLeastLessThan-iff eqc* **by** *blast*
have $e =$*?L*!i
using L $i(1)$ **by** *presburger*
have *?L*!$i \in$ *set* [*EQ* (*lst*!i) (($b@_v c$)\$$i$). $i <-$ [*dim-vec* b..< *dim-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* (*lst*!i) (($b@_v c$)\$$i$). $i <-$ [*dim-vec* b..< *dim-vec* ($b@_v c$)]]!($i-$*dim-vec* b)
by (*metis* (*no-types*, *lifting*) ‹*dim-vec* $b \leq i$› *diff-zero leD*
length-map length-upt nth-append)
then have *?L*!i = *EQ* (*lst*!i) (($b@_v c$)\$$i$)
using *add-diff-inverse-nat diff-less-mono*
by (*metis* (*no-types*, *lifting*) ‹*dim-vec* $b \leq i$› $*$ *ldimbc leD nth-map-upt*)
then have e: e = *EQ* (*lst*!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*: (*lst*!i) = (*vec-to-lpoly* (0_v (*dim-col* A) $@_v$ *row* A^T (i − *dim-vec* b)))
by (*metis* (*no-types*, *lifting*) ‹*dim-vec* b = *dim-row* A› $*$ *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* = (*vec-to-lpoly* (0_v (*dim-col* A) $@_v$ *row* A^T (i − *dim-vec* b)))
have *dim-poly ?p* ≤ *dim-col* A + *dim-row* A
using *dim-poly-of-append-vec*[*of* 0_v (*dim-col* A) *row* A^T (i − *dim-vec* b)]
index-zero-vec(2)[*of dim-col* A]
by (*metis* ‹*dim-vec* (0_v (*dim-col* A)) = *dim-col* A› *index-row*(2) *index-transpose-mat*(3))
have $\forall\, i <$ *dim-col* A. *Abstract-Linear-Poly.coeff ?p* i = 0
using *vec-coeff-append1*[*of* - 0_v (*dim-col* A) *row* A^T (i − *dim-vec* b)]
by (*metis atLeastLessThan-iff index-zero-vec*(1) *index-zero-vec*(2) *zero-le*)
then have *dim-vec* (0_v (*dim-col* A) $@_v$ *row* A^T (i − *dim-vec* b)) = *dim-col* A + *dim-row* A
by (*metis index-append-vec*(2) *index-row*(2) *index-transpose-mat*(3) *index-zero-vec*(2))
then have *allcr*: $\forall\, j \in \{0..<$*dim-row* $A\}$. *Abstract-Linear-Poly.coeff ?p* (j+*dim-col* A) = (*row* A^T (i − *dim-vec* b))\$$j$
by (*metis add-diff-cancel-right' atLeastLessThan-iff diff-add-inverse index-zero-vec*(2)
le-add-same-cancel2 less-diff-conv vec-coeff-append2)
have *vs*: *vars ?p* ⊆ {*dim-col* A..<*dim-col* A + *dim-row* A}
using *vars-vec-append-subset* **by** (*metis index-row*(2) *index-transpose-mat*(3))

have *?p ⦃ v ⦄* = (∑ *j*∈*vars ?p*. *Abstract-Linear-Poly.coeff ?p j* ∗ *v j*)
using *eval-poly-with-sum* **by** *blast*
also have ... = (∑ *j*∈{*dim-col A*..<*dim-col A* + *dim-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 ... = (∑ *j*∈{*0*..<*dim-row A*}. *Abstract-Linear-Poly.coeff ?p* (*j*+*dim-col A*) ∗ *v* (*j*+*dim-col A*))
using *sum.shift-bounds-nat-ivl*[*of* λ*j*. *Abstract-Linear-Poly.coeff ?p j* ∗ *v j* *0 dim-col A dim-row A*]
by (*metis* (*no-types*, *lifting*) *add.commute add-cancel-left-left*)
also have ... = (∑ *j*∈{*0*..<*dim-row A*}. *Abstract-Linear-Poly.coeff ?p* (*j*+*dim-col A*) ∗ *y*$*j*)
using *v i-in yr* **by** (*metis* (*no-types*, *lifting*) *sum.cong x-A*)
also have ... = (∑ *j*∈{*0*..<*dim-row A*}. (*row A*^{T} (*i* − *dim-vec b*))$*j* ∗ *y*$*j*)
using *allcr* **by** (*metis* (*no-types*, *lifting*) *sum.cong*)
also have ... = (*row A*^{T} (*i* − *dim-vec b*)) ⋅ *y*
by (*metis* ‹*dim-vec y* = *dim-row A*› *scalar-prod-def*)
also have ... = (*b*@$_v$ *c*)$*i*
using *vec-times-mat-eqD*[*OF x*(*2*)] ∗ *igeq* **by** *auto*
finally show *?thesis*
using *e lsta satisfies-constraint.simps*(*5*)[*of* - (*lst* ! *i*) ((*b* @$_v$ *c*) $ *i*)] **by** *simp*
qed
qed
then show *False* **using** ∗ **by** *blast*
qed

lemma *sound-and-compltete-mat-leqb-eqc* [*iff*]:
assumes *dim-row A*^{T} = *dim-vec c*
assumes *dim-col A*^{T} = *dim-vec b*
shows (∃ *x*. [*A* ∗$_v$ *x*]≤*b*) ∧ (∃ *y*. [*y* $_v$∗ *A*]=*c*) ⟷ (∃ *X*. *simplex* (*mat-leqb-eqc A b c*) = *Sat X*)
by (*metis assms*(*1*) *assms*(*2*) *completeness-mat-leqb-eqc index-transpose-mat*(*3*)

soundness-mat-leqb-eqc1 soundness-mat-leqb-eqc2)

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 (*LEQPP p q*) = *True* |
nonstrict-constr (*GEQPP p q*) = *True* |
nonstrict-constr (*EQPP p q*) = *True* |
nonstrict-constr - = *False*

abbreviation *nonstrict-constrs cs* ≡ (∀ *a* ∈ *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 (*LEQPP p q*) = [*LEQ* (*p* − *q*) *0*] |
transf-constraint (*GEQPP p q*) = [*LEQ* (−(*p* − *q*)) *0*] |
transf-constraint (*EQPP p q*) = [*LEQ* (*p* − *q*) *0*, *LEQ* (−(*p* − *q*)) *0*] |
transf-constraint - = []

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

lemma *trans-constraint-creats-LEQ-only*:
assumes *transf-constraint x* ≠ []
shows (∀ *x* ∈ *set* (*transf-constraint x*). ∃ *a b*. *x* = *LEQ a b*)
using *assms* **by** (*cases x*, *auto*+)

lemma *trans-constraints-creats-LEQ-only*:
assumes *transf-constraints xs* ≠ []
assumes *x* ∈ *set* (*transf-constraints xs*)
shows ∃ *p r*. *LEQ p r* = *x*
using *assms* **apply**(*induction xs*)
using *trans-constraint-creats-LEQ-only* **apply**(*auto*)
apply *fastforce*
apply (*metis in-set-simps*(*3*) *trans-constraint-creats-LEQ-only*)
by *fastforce*

lemma *non-strict-constr-no-LT*:
assumes *nonstrict-constrs cs*
shows ∀ *x* ∈ *set cs*. ¬(∃ *a b*. *LT a b* = *x*)
using *assms nonstrict-constr.simps*(*7*) **by** *blast*

lemma *non-strict-constr-no-GT*:
assumes *nonstrict-constrs cs*
shows ∀ *x* ∈ *set cs*. ¬(∃ *a b*. *GT a b* = *x*)
using *assms nonstrict-constr.simps*(*8*) **by** *blast*

lemma *non-strict-constr-no-LTPP*:
assumes *nonstrict-constrs cs*
shows ∀ *x* ∈ *set cs*. ¬(∃ *a b*. *LTPP a b* = *x*)
using *assms nonstrict-constr.simps*(*9*) **by** *blast*

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

using *assms nonstrict-constr.simps(10)* **by** *blast*

lemma *non-strict-consts-cond*:
assumes $\bigwedge x.\ x \in set\ cs \Longrightarrow \neg(\exists\, a\ b.\ LT\ a\ b = x)$
assumes $\bigwedge x.\ x \in set\ cs \Longrightarrow \neg(\exists\, a\ b.\ GT\ a\ b = x)$
assumes $\bigwedge x.\ x \in set\ cs \Longrightarrow \neg(\exists\, a\ b.\ LTPP\ a\ b = x)$
assumes $\bigwedge x.\ x \in set\ cs \Longrightarrow \neg(\exists\, a\ b.\ GTPP\ a\ b = x)$
shows *nonstrict-constrs cs*
by (*metis assms(1) assms(2) assms(3) assms(4) nonstrict-constr.elims(3)*)

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

lemma *sat-constrs-sat-transf-constrs*:
assumes $v \models_{cs} set\ cs$
shows $v \models_{cs} set\ (transf\text{-}constraints\ cs)$
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 $v \models_{cs} set\ (transf\text{-}constraint\ cs)$
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 $v \models_{cs} set\ (transf\text{-}constraints\ cs)$
shows $v \models_{cs} set\ cs$
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 $sat\text{-}primal\ A\ b = \{\ x.\ [A *_v x] \leq b\ \}$

Dual Problem

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

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

abbreviation *max-lp* **where**
max-lp A b c ≡ *optimal-set* $(\lambda x\ y.\ (y \cdot c) \leq (x \cdot c))$ (*sat-primal A b*)

abbreviation *min-lp* **where**
min-lp A b c ≡ *optimal-set* $(\lambda x\ y.\ (y \cdot c) \geq (x \cdot c))$ (*sat-dual A c*)

lemma *optimal-setI*[*intro*]:
assumes $x \in S$
assumes $\bigwedge y.\ y \in S \Longrightarrow (\lambda x\ y.\ (y \cdot c) \geq (x \cdot c))\ x\ y$
shows $x \in$ *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 $[A *_v x] \leq b$
assumes $(\bigwedge y.\ [A *_v y] \leq b \Longrightarrow (\lambda x\ y.\ (y \cdot c) \geq (x \cdot c))\ y\ x)$
shows $x \in$ *max-lp A b c*
using *optimal-setI*[*of x* { *x*. $[A *_v x] \leq b$ } *c*]
unfolding *optimal-set-def optimal-setI*
by (*simp add*: *assms*(*1*) *assms*(*2*) *sat-primal-def*)

lemma *min-lpI* [*intro*]:
assumes $[y\ _v{*}\ A] = c$
and $(\bigwedge i.\ i < \textit{dim-vec}\ y \Longrightarrow y\ \$\ i \geq 0)$
assumes $(\bigwedge x.\ x \in \textit{sat-dual}\ A\ c \Longrightarrow (\lambda x\ y.\ (y \cdot c) \geq (x \cdot c))\ y\ x)$
shows $y \in$ *min-lp A b c*
using *optimal-setI*[*of y 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 $x \in$ *sat-primal A b*
shows $[A *_v x] \leq b$
using *assms sat-primal-def* **by** *force*

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

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

lemma *sat-dualI* [*intro*]:
assumes [*y* $_v*$ *A*]=*c* (∀ *i*<*dim-vec y*. *y* $ *i* ≥ *0*)
shows *y* ∈ *sat-dual A c*
using *assms sat-dual-def* **by** *auto*

lemma *sol-dim-in-sat-primal*: *x* ∈ *sat-primal A b* ⟹ *dim-vec x* = *dim-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* ∈ *max-lp A b c* ⟹ *dim-vec x* = *dim-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* ∈ *sat-dual A c* ⟹ *dim-vec x* = *dim-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* ∈ *min-lp A b c* ⟹ *dim-vec x* = *dim-row A*
unfolding *optimal-set-def* **using** *sol-dim-in-sat-dual*[*of x A*] **by** *blast*

lemma *min-lp-in-sat-dual*: *x* ∈ *min-lp A b c* ⟹ *x* ∈ *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* ∈ *max-lp A b c* ⟹ *x* ∈ *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*::{*comm-semiring-1*,*ordered-semiring*,*linorder*}) *mat*
fixes *b* :: '*a vec*
fixes *c* :: '*a vec*
fixes *m*
fixes *n*
assumes *b* ∈ *carrier-vec m*
assumes *c* ∈ *carrier-vec n*
assumes *A* ∈ *carrier-mat m n*
begin

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

lemma *dim-b-col-A*: *dim-vec c* = *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* ∈ {*c* ⋅ *x* |*x*. *x* ∈ *sat-primal A b*}
and *j* ∈ {*b* ⋅ *y* |*y*. *y* ∈ *sat-dual A c*}
shows *i* ≤ *j*

proof −
obtain x **where** x: $i = c \cdot x$ $[A *_v x] \leq b$
using *assms* **by** *blast*
obtain y **where** y: $j = b \cdot y$ $[y\ _v* A] = c$ $(\forall i < \textit{dim-vec}\ y.\ 0 \leq y\ \$\ i)$
using *assms* **by** *blast*
have *d1*: *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*: *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 ... $= (A^T *_v y) \cdot x$
using *mult-right-eq carrier-vecD y abstract-LP-def*
by (*metis abstract-LP-axioms calculation d1*)
also have ... $= (A *_v x) \cdot y$
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 ... $\leq b \cdot y$
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 $i \leq j$ **.**
qed

theorem *weak-duality-theorem*:
assumes $x \in$ *max-lp A b c*
assumes $y \in$ *min-lp A b c*
shows $x \cdot c \leq y \cdot b$
proof −
define i **where** i: $i = x \cdot c$
define j **where** j: $j = y \cdot b$
have *dx*: *dim-vec* $x = n$
using *sol-dim-in-max-lp*[*of x c A b*, *OF assms*(*1*)] *abstract-LP-axioms abstract-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 $*$: $i \in \{c \cdot x \mid x.\ [A *_v x] \leq b\}$ **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 $**$: $j \in \{b \cdot y \mid y.\ [y\ _v* A] = c \wedge (\forall i < \textit{dim-vec}\ y.\ y\$i \geq 0)\}$
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-eqc A b c @
                                  from-index-geq0-vector (dim-vec c) (0_v (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 = Sat X
  shows ⟨X⟩ |=_cs set ((x-times-c-geq-y-times-b c b # mat-leqb-eqc A b c @
                    from-index-geq0-vector (dim-vec c) (0_v (dim-vec b))))
proof −
  have simplex (x-times-c-geq-y-times-b c b # mat-leqb-eqc A b c @
       from-index-geq0-vector (dim-vec c) (0_v (dim-vec b))) = Sat X
  proof (rule ccontr)
   assume simplex (x-times-c-geq-y-times-b c b # mat-leqb-eqc A b c @ from-index-geq0-vector
(dim-vec c) (0_v (dim-vec b))) ≠ Inr X
    then have ∃ e. simplex (x-times-c-geq-y-times-b c b # mat-leqb-eqc A b c @
from-index-geq0-vector (dim-vec c) (0_v (dim-vec b))) = Unsat e
      by (metis assms create-optimal-solutions.simps sum.case(2) sumE)
    then have ∃ e. create-optimal-solutions A b c = 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 dim-vec b = dim-row A
  assumes create-optimal-solutions A b c = Sat X
    and (x, y) = split-i-j-x (dim-col A) (dim-vec b) X
  shows [A ∗_v x]≤b
proof −
  have ∗: ⟨X⟩ |=_cs set (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 *dim-vec c* = *dim-row* A^T
and *dim-vec b* = *dim-col* A^T
assumes *create-optimal-solutions A b c* = *Sat X*
and (*x*, *y*) = *split-i-j-x* (*dim-vec c*) (*dim-vec c* + *dim-vec b*) *X*
shows [*y* $_v$∗ *A*]=*c*
proof −
have ∗: ⟨*X*⟩ $\models_{cs}$ *set* (*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 *dim-row* A^T = *dim-vec c*
using *assms*(*1*) **by** *linarith*
moreover have *dim-col* A^T = *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*) ∗]
‹*dim-vec b* = *dim-col* A^T› ‹*dim-vec c* = *dim-row* A^T›)
qed

lemma *create-opt-sol-satisfies-leq*:
assumes *create-optimal-solutions A b c* = *Sat X*
assumes (*x*, *y*) = *split-n-m-x* (*dim-vec c*) (*dim-vec b*) *X*
shows *x* ⋅ *c* ≥ *y* ⋅ *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*) = *split-n-m-x* (*dim-vec c*) (*dim-vec b*) *X*
shows ⋀*i*. *i* < *dim-vec y* ⟹ *y*$*i* ≥ *0*
proof −
fix *i*
assume *i* < *dim-vec y*
have ∗: ⟨*X*⟩ $\models_{cs}$ *set* (*from-index-geq0-vector* (*dim-vec c*) (*0*$_v$ (*dim-vec b*)))
using *assms*(*1*) *create-opt-sol-satisfies* **by** (*metis UnCI append-Cons set-append*)
have ∗∗: *i* < *dim-vec b*
by (*metis* ‹*i* < *dim-vec y*› *assms*(*2*) *split-n-m-x-abbrev-dims*(*2*))
then show *0* ≤ *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*) = *split-n-m-x* (*dim-vec c*) (*dim-vec b*) *X*
shows [*A* $*_v$ *x*]≤*b* [*y* $_v*$ *A*]=*c x* · *c* ≥ *y* · *b* ⋀*i*. *i* < *dim-vec y* ⟹ *y*$*i* ≥ *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* = *Sat X*
assumes (*x*, *y*) = *split-n-m-x* (*dim-vec c*) (*dim-vec b*) *X*
shows *c* · *x* ∈ {*c* · *x* |*x*. [*A* $*_v$ *x*]≤*b*}
b · *y* ∈ {*b* · *y* |*y*. [*y* $_v*$ *A*]=*c* ∧ (∀ *i* < *dim-vec y*. *y*$*i* ≥ *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* = *Sat X*
assumes (*x*, *y*) = *split-n-m-x* (*dim-vec c*) (*dim-vec b*) *X*
shows *c* · *x* ≤ *b* · *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* = *Sat X*
assumes (*x*, *y*) = *split-n-m-x* (*dim-vec c*) (*dim-vec b*) *X*
shows *c* · *x* = *b* · *y*
using *create-opt-sol-in-LP*(*3*)[*of X x y*, *OF assms*] *cx-leq-bx-in-creating-opt*[*OF assms*]
comm-scalar-prod[*of c dim-vec c x*] *comm-scalar-prod*[*of b 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 = Sat X*
assumes *(x, y) = split-n-m-x (dim-vec c) (dim-vec b) X*
shows *x ∈ max-lp A b c*
proof
show *[A *$_v$ x]≤b*
using *assms(1) assms(2) create-opt-sol-in-LP(1)* **by** *blast*
fix *z*
assume *a*: *[A *$_v$ z]≤b*
have *1*: *c ∙ z ∈ {c ∙ x |x. x ∈ sat-primal A b}*
using *sat-primalI[of A z b, OF a]* **by** *blast*
have *2*: *b ∙ y ∈ {b ∙ y |y. y ∈ 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 ∙ z ≤ b ∙ 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 *... = x ∙ 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 ∙ c ≤ x ∙ 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 = Sat x*
shows *x ∈ max-lp A b c*
proof −
obtain *X* **where** *X*: *create-optimal-solutions A b c = Sat X*
by *(metis Inr-Inl-False assms old.sum.exhaust old.sum.simps(5) optimize-no-cond.simps)*
have *x = (fst (split-n-m-x (dim-vec c) (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 = Sat x*
shows *x ∈ 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 = (if dim-vec b = dim-row A ∧ dim-vec c = dim-col A then Some (optimize-no-cond A b c) else None)

lemma *optimize-sound*:
assumes *maximize A b c = Some (Sat x)*
shows *x ∈ max-lp A b c*
proof −
have ∗: *dim-vec b = dim-row A ∧ dim-vec c = 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 *Sat x = 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 = Some x*
shows *dim-vec b = dim-row A dim-vec c = dim-col A*
by (*metis assms maximize.simps option.distinct*(*1*))+

lemma *optimize-sol-dimension*:
assumes *maximize A b c = Some (Sat x)*
shows *x ∈ carrier-vec* (*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 = Some (Sat x)*
shows [*A* ∗$_v$ *x*]≤*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*) $\Longrightarrow$ *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. Marić, M. Spasić, and R. Thiemann. An incremental simplex algorithm with unsatisfiable core generation. *Archive of Formal Proofs*, Aug. 2018. http://isa-afp.org/entries/Simplex.html, Formal proof development.

[4] 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.