Linear-Programming

Julian Parsert

February 23, 2021

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*}}
⟨*proof*⟩

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

2 General Theorems used later, that could be moved

lemma *split-four-block-dual-fst-lst*:
assumes *split-block* (*four-block-mat A B C D*) (*dim-row A*) (*dim-col A*) = (*U*, *X*, *Y*, *V*)
shows *U* = *A V* = *D*
⟨*proof*⟩

lemma *append-split-vec-distrib-scalar-prod*:
assumes *dim-vec* (*u* @$_v$ *w*) = *dim-vec x*
shows (*u* @$_v$ *w*) · *x* = *u* · (*vec-first x* (*dim-vec u*)) + *w* · (*vec-last x* (*dim-vec w*))
⟨*proof*⟩

lemma *append-dot-product-split*:
assumes *dim-vec* (*u* @$_v$ *w*) = *dim-vec x*
shows (*u* @$_v$ *w*) · *x* = (∑ *i*∈{*0*..< *dim-vec u*}. *u*$*i* ∗ *x*$*i*) + (∑ *i*∈{*0*..< *dim-vec w*}. *w*$*i* ∗ *x*$(*i* + *dim-vec u*))
⟨*proof*⟩

lemma *assoc-scalar-prod-mult-mat-vec*:
fixes *A* :: *'a*::*comm-semiring-1 mat*
assumes *y* ∈ *carrier-vec n*
assumes *x* ∈ *carrier-vec m*
assumes *A* ∈ *carrier-mat n m*
shows (*A* ∗$_v$ *x*) · *y* = (A^T ∗$_v$ *y*) · *x*
⟨*proof*⟩

3 Vectors

abbreviation *singletonV* ([-]$_v$) **where** *singletonV e* ≡ (*vec 1* (λ*i*. *e*))

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

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

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

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

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

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

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

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

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 $\textit{row}\ (A \mathbin{@_r} B)\ i = \textit{row}\ B\ (i - \textit{dim-row}\ A)$
⟨*proof*⟩

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

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)*
⟨*proof*⟩

lemma *mat-append-row-dims*:
shows *mat-append-row A c ∈ carrier-mat (dim-row A + 1) (dim-col A)*
⟨*proof*⟩

lemma *mat-append-col-col*:
assumes *dim-row A = dim-vec b*
shows *col (mat-append-col A b) (dim-col A) = b*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *mat-append-row-row*:
assumes *dim-col A = dim-vec c*
shows *row (mat-append-row A c) (dim-row A) = c*
⟨*proof*⟩

lemma *mat-append-row-in-mat*:
assumes *i < dim-row A*
shows *row (mat-append-row A r) i = row A i*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *constructing-append-col-row*:
assumes $i < \textit{dim-row}\ A$
and $\textit{dim-row}\ A = \textit{dim-vec}\ b$
shows $\textit{row}\ (\textit{mat-append-col}\ A\ b)\ i = \textit{row}\ A\ i\ @_v\ [\textit{vec-index}\ b\ i]_v$
⟨*proof*⟩

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

lemma *one-element-vec-carrier*: $\textit{one-element-vec}\ n\ e \in \textit{carrier-vec}\ n$
⟨*proof*⟩

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

lemma *one-element-vec-access* [*simp*]: $\bigwedge i.\ i < n \Longrightarrow \textit{vec-index}\ (\textit{one-element-vec}\ n\ e)\ i = e$
⟨*proof*⟩

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

lemma *single-nz-val-carrier*: $\textit{single-nz-val}\ n\ i\ v \in \textit{carrier-vec}\ n$
⟨*proof*⟩

lemma *single-nz-val-access1* [*simp*]: $i < n \Longrightarrow \textit{single-nz-val}\ n\ i\ v\ \$\ i = v$
⟨*proof*⟩

lemma *single-nz-val-access2* [*simp*]: $i < n \Longrightarrow j < n \Longrightarrow i \neq j \Longrightarrow \textit{single-nz-val}\ n\ i\ v\ \$\ j = 0$
⟨*proof*⟩

lemma $i < n \Longrightarrow (v \cdot_v \textit{unit-vec}\ n\ i)\ \$\ i = (v::'a::\{\textit{monoid-mult},\textit{times},\textit{zero-neq-one}\})$
⟨*proof*⟩

lemma *single-nz-val-unit-vec*:
fixes $v::'a::\{\textit{monoid-mult},\textit{times},\textit{zero-neq-one},\textit{mult-zero}\}$
shows $v \cdot_v (\textit{unit-vec}\ n\ i) = \textit{single-nz-val}\ n\ i\ v$
⟨*proof*⟩

lemma *single-nz-valI* [*intro*]:
fixes $v\ i\ \textit{val}$
assumes $\bigwedge j.\ j < \textit{dim-vec}\ v \Longrightarrow j \neq i \Longrightarrow v\$j = 0$
assumes $v\$i = \textit{val}$
shows $v = \textit{single-nz-val}\ (\textit{dim-vec}\ v)\ i\ \textit{val}$
⟨*proof*⟩

lemma *single-nz-val-dotP*:
assumes $i < n$
assumes $\textit{dim-vec}\ x = n$

shows *single-nz-val n i v* $\cdot$ *x* = *v* $*$ *x* \$ *i*
⟨*proof*⟩

lemma *single-nz-zero-singleton*: *single-nz-val 1 0 v* = $[v]_v$
⟨*proof*⟩

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*∈{*0* ..< *dim-vec x*}. *e* $*$ *x* \$ *i*)
⟨*proof*⟩

lemma *one-element-vec-dotP*:
assumes *dim-vec x* = *n*
shows (*one-element-vec n e*) $\cdot$ *x* = ($\sum$ *i*∈{*0* ..< *dim-vec x*}. *e* $*$ *x* \$ *i*)
⟨*proof*⟩

lemma *singleton-dotP* [*simp*]: *dim-vec x* = *1* ⟹ $[v]_v$ $\cdot$ *x* = *v* $*$ *x*\$*0*
⟨*proof*⟩

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

lemma *singleton-appends-dotP* [*simp*]: *dim-vec x* = *dim-vec y* ⟹ (*x* @$_v$ $[v]_v$) $\cdot$ (*y* @$_v$ $[w]_v$) = *x* $\cdot$ *y* + *v* $*$ *w*
⟨*proof*⟩

end
theory *Matrix-LinPoly*
imports
Jordan-Normal-Form.Matrix-Impl
Farkas.Simplex-for-Reals
Farkas.Matrix-Farkas
begin

Add this to linear polynomials in Simplex

lemma *eval-poly-with-sum*: (*v* ⦃ *X* ⦄) = ($\sum$ *x*∈ *vars v*. *coeff v x* $*$ *X x*)
⟨*proof*⟩

lemma *eval-poly-with-sum-superset*:
assumes *finite S*
assumes *S* ⊇ *vars v*
shows (*v* ⦃*X*⦄) = ($\sum$ *x*∈*S*. *coeff v x* $*$ *X x*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *sum-list-map-upto-coeff-limit*:
assumes *i ≥ length L*
shows *coeff (list-to-lpoly L) i = 0*
⟨*proof*⟩

lemma *rl-lpoly-coeff-nth-non-empty*:
assumes *i < length cs*
assumes *cs ≠ []*
shows *coeff (list-to-lpoly cs) i = cs!i*
⟨*proof*⟩

lemma *list-to-lpoly-coeff-nth*:
assumes *i < length cs*
shows *coeff (list-to-lpoly cs) i = cs ! i*
⟨*proof*⟩

lemma *rat-list-outside-zero*:
assumes *length cs ≤ i*
shows *coeff (list-to-lpoly cs) i = 0*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *lpoly-to-vec-0-iff-zero-poly* [*iff*]:
shows (*lpoly-to-vec p*) = 0_v *0* ⟷ *p* = *0*
⟨*proof*⟩

lemma *dim-poly-dim-vec-equiv*:
dim-vec (*lpoly-to-vec p*) = *dim-poly p*
⟨*proof*⟩

lemma *dim-poly-greater-ex-coeff*: *dim-poly x* > *d* ⟹ ∃ *i*≥*d*. *coeff x i* ≠ *0*
⟨*proof*⟩

lemma *dimpoly-all-zero-limit*:
assumes ⋀*i*. *i* ≥ *d* ⟹ *coeff x i* = *0*
shows *dim-poly x* ≤ *d*
⟨*proof*⟩

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*⟩

lemma *vars-subset-0-dim-poly*:
vars z ⊆ {*0*..<*dim-poly z*}
⟨*proof*⟩

lemma *in-dim-and-not-var-zero*: *x* ∈ {*0*..<*dim-poly z*} − *vars z* ⟹ *coeff z x* = *0*
⟨*proof*⟩

lemma *valuate-with-dim-poly*: *z* ⦃ *X* ⦄ = (∑ *i*∈{*0*..<*dim-poly z*}. *coeff z i* ∗ *X i*)
⟨*proof*⟩

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

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

lemma *list-to-lpoly-dim-less*: *length cs* ≥ *dim-poly* (*list-to-lpoly cs*)
⟨*proof*⟩

Transform rational vectors to linear polynomials

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

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

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

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 $(x + y)\ \$\ a = coeff\ (vec\text{-}to\text{-}lpoly\ x + vec\text{-}to\text{-}lpoly\ y)\ a$
⟨*proof*⟩

lemma *outside-list-coeff0*:
assumes $i \geq dim\text{-}vec\ xs$
shows $coeff\ (vec\text{-}to\text{-}lpoly\ xs)\ i = 0$
⟨*proof*⟩

lemma *vec-to-poly-dim-less*:
$dim\text{-}poly\ (vec\text{-}to\text{-}lpoly\ x) \leq dim\text{-}vec\ x$
⟨*proof*⟩

lemma *vec-to-lpoly-from-lpoly-coeff-dual1*:
$coeff\ (vec\text{-}to\text{-}lpoly\ (lpoly\text{-}to\text{-}vec\ p))\ i = coeff\ p\ i$
⟨*proof*⟩

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 $(lpoly\text{-}to\text{-}vec\ (vec\text{-}to\text{-}lpoly\ v))\ \$\ i = v\ \$\ i$
⟨*proof*⟩

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

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*⟩

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

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

lemma *coeff-nonzero-dim-vec-non-zero*:
assumes $coeff\ (vec\text{-}to\text{-}lpoly\ v)\ i \neq 0$
shows $v\ \$\ i \neq 0\ i < dim\text{-}vec\ v$

⟨*proof*⟩

lemma *lpoly-of-v-equals-v-append0*:
vec-to-lpoly v = *vec-to-lpoly* (*v* @$_v$ 0$_v$ *a*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *vec-to-lpoly-eval-dot-prod*:
(*vec-to-lpoly v*) {‖ *x* ‖} = *v* ⋅ (*vec* (*dim-vec v*) *x*)
⟨*proof*⟩

lemma *dim-poly-of-append-vec*:
dim-poly (*vec-to-lpoly* (*a*@$_v$*b*)) ≤ *dim-vec a* + *dim-vec b*
⟨*proof*⟩

lemma *vec-coeff-append1*: *i* ∈ {*0*..<*dim-vec a*} ⟹ *coeff* (*vec-to-lpoly* (*a*@$_v$*b*)) *i* = *a*$*i*
⟨*proof*⟩

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*)
⟨*proof*⟩

Maybe Code Equation

lemma *vec-to-lpoly-poly-of-vec-eq*: *vec-to-lpoly v* = *poly-of-vec v*
⟨*proof*⟩

lemma *vars-vec-append-subset*: *vars* (*vec-to-lpoly* (0$_v$ *n* @$_v$ *v*)) ⊆ {*n*..<*n*+*dim-vec v*}
⟨*proof*⟩

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* ⟹ *matrix-to-lpolies A* ! *i* = *vec-to-lpoly* (*row A i*)
⟨*proof*⟩

lemma *outside-of-col-range-is-0*:
assumes *i* < *dim-row A* **and** *j* ≥ *dim-col A*
shows *coeff* ((*matrix-to-lpolies A*)!*i*) *j* = *0*
⟨*proof*⟩

lemma *polys-greater-col-zero*:
assumes *x* ∈ *set* (*matrix-to-lpolies A*)
assumes *j* ≥ *dim-col A*
shows *coeff x j* = *0*

⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *vec-to-lpoly-valuate-equiv-dot-prod*:
assumes *dim-vec y* = *dim-vec x*
shows (*vec-to-lpoly y*) ⦃ ($)*x* ⦄ = *y* · *x*
⟨*proof*⟩

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*) · *x*
⟨*proof*⟩

lemma *matrix-to-lpolies-lambda-valuate-scalarP*:
assumes $i <$ *dim-row A*
assumes *dim-col A* = *dim-vec x*
shows (*matrix-to-lpolies A*!*i*) ⦃ (λ*i*. (*if i* < *dim-vec x then x*$*i else 0*)) ⦄ = (*row A i*) · *x*
⟨*proof*⟩

end
theory *LP-Preliminaries*
imports
More-Jordan-Normal-Forms
Matrix-LinPoly
Jordan-Normal-Form.Matrix-Impl

Farkas.Simplex-for-Reals
HOL−Library.Mapping
begin

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

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

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

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

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

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

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

6 Get different matrices into same space, without interference

fun *two-block-non-interfering* **where**
two-block-non-interfering A B = (*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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *two-block-non-interfering-row-comp2*:
assumes *i* <*dim-row* (*two-block-non-interfering A B*)
and *i* ≥ *dim-row A*
shows *row* (*two-block-non-interfering A B*) *i* = (0_v (*dim-col A*)) $@_v$ *row B* (*i* − *dim-row A*)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

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*⟩ ⟧})
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *split-n-m-x-abbrev-dims*:
assumes *split-n-m-x n m x* = (*a*, *b*)
shows *dim-vec a* = *n dim-vec b* = *m*
⟨*proof*⟩

lemma *split-access-fst-1*:
assumes *k* < *i*
assumes *split-i-j-x i j x* = (*a*, *b*)
shows *a* $ *k* = ⟨*x*⟩ *k*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*⟩

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 ⋀*i*. *i* < *dim-vec b* ⟹ (*A* ∗$_v$ *x*)$*i* ≤ *b*$*i*
shows [*A* ∗$_v$ *x*]≤*b*
⟨*proof*⟩

lemma *mat-times-vec-leqD*[*dest*]:
assumes [*A* ∗$_v$ *x*]≤*b*
shows *dim-row A* = *dim-vec b dim-col A* = *dim-vec x* ⋀*i*. *i* < *dim-vec b* ⟹ (*A* ∗$_v$ *x*)$*i* ≤ *b*$*i*
⟨*proof*⟩

lemma *vec-times-mat-eqD*[*dest*]:
assumes [*y* $_v$∗ *A*]=*c*
shows (∀ *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*)
⟨*proof*⟩

lemma *vec-times-mat-leqD*[*dest*]:
assumes [*y* $_v$∗ *A*]≤*c*
shows (∀ *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*)

⟨*proof*⟩

lemma *mat-times-vec-eqI*[*intro*]:
assumes *dim-col* A^T = *dim-vec* x
assumes *dim-row* A^T = *dim-vec* c
assumes $\bigwedge i.\ i <$ *dim-vec* $c \Longrightarrow (A^T *_v x)\$i = c\$i$
shows $[x\ _v* A]{=}c$
⟨*proof*⟩

lemma *mat-leqb-eqc-split-correct1*:
assumes *dim-vec* b = *dim-row* A
assumes $\langle X\rangle \models_{cs}$ *set* (*mat-leqb-eqc* $A\ b\ c$)
assumes (x,y) = *split-i-j-x* (*dim-col* A) $l\ X$
shows $[A *_v x]{\leq}b$
⟨*proof*⟩

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 $[A *_v x]{\leq}b$
⟨*proof*⟩

lemma *sat-mono*:
assumes *set* $A \subseteq$ *set* B
shows $\langle X\rangle \models_{cs}$ *set* $B \Longrightarrow \langle X\rangle \models_{cs}$ *set* A
⟨*proof*⟩

lemma *mat-leqb-eqc-split-subset-correct1*:
assumes *dim-vec* b = *dim-row* A
assumes *set* (*mat-leqb-eqc* $A\ b\ c$) $\subseteq$ *set* S
assumes *simplex* S = *Sat* X
assumes (x,y) = *split-i-j-x* (*dim-col* A) $l\ X$
shows $[A *_v x]{\leq}b$
⟨*proof*⟩

lemma *mat-leqb-eqc-split-correct2*:
assumes *dim-vec* c = *dim-row* A^T
assumes *dim-vec* b = *dim-col* A^T
assumes $\langle X\rangle \models_{cs}$ *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*⟩

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 $[y\ {}_v{*}\ A]{=}c$
⟨*proof*⟩

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 $[y\ {}_v{*}\ A]{=}c\ [A *_v x]{\leq}b$
⟨*proof*⟩

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*) $\{\!|\langle X\rangle|\!\} = c \cdot x$
⟨*proof*⟩

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*)) $\{\!|\langle X\rangle|\!\} = c \cdot y$
⟨*proof*⟩

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

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

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

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

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

shows $\exists X.$ *simplex* (*mat-x-leq-vec A b*) = *Sat X*
⟨*proof*⟩

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 $\exists x.$ $[x\ {}_v{*}\ A]{=}c$
⟨*proof*⟩

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

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

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 $\exists y.$ $[y\ {}_v{*}\ A]{=}c$
⟨*proof*⟩

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

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 $(\exists x.\ [A *_v x]{\leq}b) \wedge (\exists y.\ [y\ {}_v{*}\ A]{=}c) \longleftrightarrow (\exists X.$ *simplex* (*mat-leqb-eqc A b c*) = *Sat X*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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

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

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

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

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

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*
⟨*proof*⟩

lemma *sat-constr-sat-transf-constrs*:
assumes $v \models_c cs$
shows $v \models_{cs} set\ (transf\text{-}constraint\ cs)$
⟨*proof*⟩

lemma *sat-constrs-sat-transf-constrs*:
assumes $v \models_{cs} set\ cs$
shows $v \models_{cs} set\ (transf\text{-}constraints\ cs)$
⟨*proof*⟩

lemma *sat-transf-constrs-sat-constr*:
assumes *nonstrict-constr cs*
assumes $v \models_{cs} set\ (transf\text{-}constraint\ cs)$
shows $v \models_c cs$
⟨*proof*⟩

lemma *sat-transf-constrs-sat-constrs*:
assumes *nonstrict-constrs cs*
assumes $v \models_{cs} set\ (transf\text{-}constraints\ cs)$
shows $v \models_{cs} set\ cs$
⟨*proof*⟩

end
theory *Linear-Programming*
imports
HOL−Library.Code-Target-Int
LP-Preliminaries
Farkas.Simplex-for-Reals
begin

8 Abstract LPs

Primal Problem

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

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

abbreviation *max-lp* **where**
max-lp A b c ≡ *optimal-set* (λ*x y*. (*y* · *c*) ≤ (*x* · *c*)) (*sat-primal A b*)

abbreviation *min-lp* **where**
min-lp A b c ≡ *optimal-set* (λ*x y*. (*y* · *c*) ≥ (*x* · *c*)) (*sat-dual A c*)

lemma *optimal-setI*[*intro*]:
assumes *x* ∈ *S*
assumes ⋀*y*. *y* ∈ *S* ⟹ (λ*x y*. (*y* · *c*) ≥ (*x* · *c*)) *x y*
shows *x* ∈ *optimal-set* (λ*x y*. (*y* · *c*) ≥ (*x* · *c*)) *S*
⟨*proof*⟩

lemma *max-lpI* [*intro*]:
assumes [*A* ∗$_v$ *x*]≤*b*
assumes (⋀*y*. [*A* ∗$_v$ *y*]≤*b* ⟹ (λ*x y*. (*y* · *c*) ≥ (*x* · *c*)) *y x*)
shows *x* ∈ *max-lp A b c*
⟨*proof*⟩

lemma *min-lpI* [*intro*]:
assumes [*y* $_v$∗ *A*]=*c*
and (⋀*i*. *i*<*dim-vec y* ⟹ *y* $ *i* ≥ 0)
assumes (⋀*x*. *x* ∈ *sat-dual A c* ⟹ (λ*x y*. (*y* · *c*) ≥ (*x* · *c*)) *y x*)
shows *y* ∈ *min-lp A b c*
⟨*proof*⟩

lemma *sat-primalD* [*dest*]:
assumes *x* ∈ *sat-primal A b*
shows [*A* ∗$_v$ *x*]≤*b*
⟨*proof*⟩

lemma *sat-primalI* [*intro*]:
assumes [*A* ∗$_v$ *x*]≤*b*
shows *x* ∈ *sat-primal A b*
⟨*proof*⟩

lemma *sat-dualD* [*dest*]:
assumes *y* ∈ *sat-dual A c*
shows [*y* $_v$∗ *A*]=*c* (∀ *i*<*dim-vec y*. *y* $ *i* ≥ 0)
⟨*proof*⟩

lemma *sat-dualI* [*intro*]:
assumes [*y* $_v$∗ *A*]=*c* (∀ *i*<*dim-vec y*. *y* $ *i* ≥ 0)
shows *y* ∈ *sat-dual A c*
⟨*proof*⟩

lemma *sol-dim-in-sat-primal*: *x* ∈ *sat-primal A b* ⟹ *dim-vec x* = *dim-col A*
⟨*proof*⟩

lemma *sol-dim-in-max-lp*: $x \in$ *max-lp* $A\ b\ c \Longrightarrow$ *dim-vec* $x =$ *dim-col* A
⟨*proof*⟩

lemma *sol-dim-in-sat-dual*: $x \in$ *sat-dual* $A\ c \Longrightarrow$ *dim-vec* $x =$ *dim-row* A
⟨*proof*⟩

lemma *sol-dim-in-min-lp*: $x \in$ *min-lp* $A\ b\ c \Longrightarrow$ *dim-vec* $x =$ *dim-row* A
⟨*proof*⟩

lemma *min-lp-in-sat-dual*: $x \in$ *min-lp* $A\ b\ c \Longrightarrow x \in$ *sat-dual* $A\ c$
⟨*proof*⟩

lemma *max-lp-in-sat-primal*: $x \in$ *max-lp* $A\ b\ c \Longrightarrow x \in$ *sat-primal* $A\ b$
⟨*proof*⟩

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

lemma *dim-b-row-A*: *dim-vec* $b =$ *dim-row* A
⟨*proof*⟩

lemma *dim-b-col-A*: *dim-vec* $c =$ *dim-col* A
⟨*proof*⟩

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

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*⟩

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*⟩

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*⟩

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

lemma *create-opt-sol-satisfies-leq*:
assumes *create-optimal-solutions A b c* = *Sat X*
assumes (*x*, *y*) = *split-n-m-x* (*dim-vec c*) (*dim-vec b*) *X*
shows *x* · *c* ≥ *y* · *b*
⟨*proof*⟩

lemma *create-opt-sol-satisfies-geq0*:
assumes *create-optimal-solutions A b c* = *Sat X*
assumes (*x*, *y*) = *split-n-m-x* (*dim-vec c*) (*dim-vec b*) *X*
shows ⋀*i*. *i* < *dim-vec y* ⟹ *y*$*i* ≥ 0
⟨*proof*⟩

locale *rat-LP* = *abstract-LP A b c m n*
for *A* ::*rat mat*

and b :: *rat vec*
and c :: *rat vec*
and m :: *nat*
and n :: *nat*
begin

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

lemma *create-optim-in-sols*:
assumes *create-optimal-solutions A b c* = *Sat X*
assumes (x, y) = *split-n-m-x* (*dim-vec c*) (*dim-vec b*) *X*
shows $c \cdot x \in \{c \cdot x\ |x.\ [A *_v x] \leq b\}$
$b \cdot y \in \{b \cdot y\ |y.\ [y\ _v* A] = c \wedge (\forall\, i < dim\text{-}vec\ y.\ y\$i \geq 0)\}$
⟨*proof*⟩

lemma *cx-leq-bx-in-creating-opt*:
assumes *create-optimal-solutions A b c* = *Sat X*
assumes (x, y) = *split-n-m-x* (*dim-vec c*) (*dim-vec b*) *X*
shows $c \cdot x \leq b \cdot y$
⟨*proof*⟩

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 $c \cdot x = b \cdot y$
⟨*proof*⟩

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 $x \in$ *max-lp A b c*
⟨*proof*⟩

lemma *optimize-no-cond-correct*:
assumes *optimize-no-cond A b c* = *Sat x*
shows $x \in$ *max-lp A b c*
⟨*proof*⟩

lemma *optimize-no-cond-sol-sat*:
assumes *optimize-no-cond A b c* = *Sat x*
shows $x \in$ *sat-primal A b*
⟨*proof*⟩

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*⟩

lemma *maximize-option-elim*:
assumes *maximize A b c* = *Some x*
shows *dim-vec b* = *dim-row A dim-vec c* = *dim-col A*
⟨*proof*⟩

lemma *optimize-sol-dimension*:
assumes *maximize A b c* = *Some* (*Sat x*)
shows *x* ∈ *carrier-vec* (*dim-col A*)
⟨*proof*⟩

lemma *optimize-sat*:
assumes *maximize A b c* = *Some* (*Sat x*)
shows [*A* ∗$_v$ *x*]≤*b*
⟨*proof*⟩

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*
⟨*proof*⟩

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.