# Diophantine Equations*

Florian Meßner Julian Parsert Jonas Schöpf
Christian Sternagel

June 16, 2019

**Abstract**

In this entry we formalize Huet's [1] bounds for minimal solutions of homogenous linear Diophantine equations (HLDEs). Based on these bounds, we further provide a certified algorithm for computing the set of all minimal solutions of a given HLDE.

## Contents



*This work is supported by the Austrian Science Fund (FWF): project P27502.

# 1 Vectors as Lists of Naturals

**theory** *List-Vector*
**imports** *Main*
**begin**

**lemma** *lex-lengthD*: $(x, y) \in lex\ P \Longrightarrow length\ x = length\ y$
⟨*proof*⟩

**lemma** *lexI*:
**assumes** *length ys* = *length xs* **and** $i <$ *length xs*
**and** *take i xs* = *take i ys* **and** $(xs\ !\ i, ys\ !\ i) \in r$
**shows** $(xs, ys) \in lex\ r$
⟨*proof*⟩

**lemma** *lex-take-index*:
**assumes** $(xs, ys) \in lex\ r$
**obtains** *i* **where** *length ys* = *length xs*
**and** $i <$ *length xs* **and** *take i xs* = *take i ys*
**and** $(xs\ !\ i, ys\ !\ i) \in r$
⟨*proof*⟩

**lemma** *mods-with-nats*:
**assumes** $(v::nat) > w$
**and** $(v * b)\ mod\ a = (w * b)\ mod\ a$
**shows** $((v - w) * b)\ mod\ a = 0$
⟨*proof*⟩
**abbreviation** *zeroes* :: *nat* ⇒ *nat list*
**where**
*zeroes n* ≡ *replicate n 0*

**lemma** *rep-upd-unit*:
**assumes** $x = (zeroes\ n)[i := a]$
**shows** $\forall j <$ *length x*. $(j \neq i \longrightarrow x\ !\ j = 0) \wedge (j = i \longrightarrow x\ !\ j = a)$
⟨*proof*⟩

**definition** *nonzero-iff*: *nonzero xs* $\longleftrightarrow$ $(\exists\, x{\in}set\ xs.\ x \neq 0)$

**lemma** *nonzero-append* [*simp*]:
*nonzero* (*xs* @ *ys*) $\longleftrightarrow$ *nonzero xs* $\vee$ *nonzero ys* ⟨*proof*⟩

## 1.1 The Inner Product

**definition** *dotprod* :: *nat list* $\Rightarrow$ *nat list* $\Rightarrow$ *nat* (**infixl** $\cdot$ *70*)
**where**
*xs* $\cdot$ *ys* = $(\sum i{<}min\ (length\ xs)\ (length\ ys).\ xs\ !\ i * ys\ !\ i)$

**lemma** *dotprod-code* [*code*]:
*xs* $\cdot$ *ys* = *sum-list* (*map* $(\lambda(x,\ y).\ x * y)$ (*zip xs ys*))
⟨*proof*⟩

**lemma** *dotprod-commute*:
**assumes** *length xs* = *length ys*
**shows** *xs* $\cdot$ *ys* = *ys* $\cdot$ *xs*
⟨*proof*⟩

**lemma** *dotprod-Nil* [*simp*]: [] $\cdot$ [] = *0*
⟨*proof*⟩

**lemma** *dotprod-Cons* [*simp*]:
(*x* # *xs*) $\cdot$ (*y* # *ys*) = *x* $*$ *y* + *xs* $\cdot$ *ys*
⟨*proof*⟩

**lemma** *dotprod-1-right* [*simp*]:
*xs* $\cdot$ *replicate* (*length xs*) *1* = *sum-list xs*
⟨*proof*⟩

**lemma** *dotprod-0-right* [*simp*]:
*xs* $\cdot$ *zeroes* (*length xs*) = *0*
⟨*proof*⟩

**lemma** *dotprod-unit* [*simp*]:
**assumes** *length a* = *n*
**and** *k* < *n*
**shows** *a* $\cdot$ (*zeroes n*)[*k* := *zk*] = *a* ! *k* $*$ *zk*
⟨*proof*⟩

**lemma** *dotprod-gt0*:
**assumes** *length x* = *length y* **and** $\exists\, i{<}length\ y.\ x\ !\ i > 0 \wedge y\ !\ i > 0$
**shows** *x* $\cdot$ *y* > *0*
⟨*proof*⟩

**lemma** *dotprod-gt0D*:
**assumes** *length x* = *length y*

**and** $x \cdot y > 0$
**shows** $\exists i{<}\mathit{length}\ y.\ x\ !\ i > 0 \wedge y\ !\ i > 0$
⟨*proof*⟩

**lemma** *dotprod-gt0-iff* [*iff*]:
**assumes** $\mathit{length}\ x = \mathit{length}\ y$
**shows** $x \cdot y > 0 \longleftrightarrow (\exists i{<}\mathit{length}\ y.\ x\ !\ i > 0 \wedge y\ !\ i > 0)$
⟨*proof*⟩

**lemma** *dotprod-append*:
**assumes** $\mathit{length}\ a = \mathit{length}\ b$
**shows**$(a\ @\ x) \cdot (b\ @\ y) = a \cdot b + x \cdot y$
⟨*proof*⟩

**lemma** *dotprod-le-take*:
**assumes** $\mathit{length}\ a = \mathit{length}\ b$
**and** $k \leq \mathit{length}\ a$
**shows**$\mathit{take}\ k\ a \cdot \mathit{take}\ k\ b \leq a \cdot b$
⟨*proof*⟩

**lemma** *dotprod-le-drop*:
**assumes** $\mathit{length}\ a = \mathit{length}\ b$
**and** $k \leq \mathit{length}\ a$
**shows** $\mathit{drop}\ k\ a \cdot \mathit{drop}\ k\ b \leq a \cdot b$
⟨*proof*⟩

**lemma** *dotprod-is-0* [*simp*]:
**assumes** $\mathit{length}\ x = \mathit{length}\ y$
**shows** $x \cdot y = 0 \longleftrightarrow (\forall i{<}\mathit{length}\ y.\ x\ !\ i = 0 \vee y\ !\ i = 0)$
⟨*proof*⟩

**lemma** *dotprod-eq-0-iff*:
**assumes** $\mathit{length}\ x = \mathit{length}\ a$
**and** $0 \notin \mathit{set}\ a$
**shows** $x \cdot a = 0 \longleftrightarrow (\forall e \in \mathit{set}\ x.\ e = 0)$
⟨*proof*⟩

**lemma** *dotprod-eq-nonzero-iff*:
**assumes** $a \cdot x = b \cdot y$ **and** $\mathit{length}\ x = \mathit{length}\ a$ **and** $\mathit{length}\ y = \mathit{length}\ b$
**and** $0 \notin \mathit{set}\ a$ **and** $0 \notin \mathit{set}\ b$
**shows** $\mathit{nonzero}\ x \longleftrightarrow \mathit{nonzero}\ y$
⟨*proof*⟩

**lemma** *eq-0-iff*:
$xs = \mathit{zeroes}\ n \longleftrightarrow \mathit{length}\ xs = n \wedge (\forall x{\in}\mathit{set}\ xs.\ x = 0)$
⟨*proof*⟩

**lemma** *not-nonzero-iff*: $\neg\ \mathit{nonzero}\ x \longleftrightarrow x = \mathit{zeroes}\ (\mathit{length}\ x)$
⟨*proof*⟩

**lemma** *neq-0-iff'*:
*xs* ≠ *zeroes n* ⟷ *length xs* ≠ *n* ∨ (∃ *x*∈*set xs*. *x* > *0*)
⟨*proof*⟩

**lemma** *dotprod-pointwise-le*:
**assumes** *length as* = *length xs*
**and** *i* < *length as*
**shows** *as* ! *i* ∗ *xs* ! *i* ≤ *as* ⋅ *xs*
⟨*proof*⟩

**lemma** *replicate-dotprod*:
**assumes** *length y* = *n*
**shows** *replicate n x* ⋅ *y* = *x* ∗ *sum-list y*
⟨*proof*⟩

## 1.2 The Pointwise Order on Vectors

**definition** *less-eq* :: *nat list* ⇒ *nat list* ⇒ *bool* (-/ ≤$_v$ -  [*51*, *51*] *50*)
**where**
*xs* ≤$_v$ *ys* ⟷ *length xs* = *length ys* ∧ (∀ *i*<*length xs*. *xs* ! *i* ≤ *ys* ! *i*)

**definition** *less* :: *nat list* ⇒ *nat list* ⇒ *bool* (-/ <$_v$ -  [*51*, *51*] *50*)
**where**
*xs* <$_v$ *ys* ⟷ *xs* ≤$_v$ *ys* ∧ ¬ *ys* ≤$_v$ *xs*

**interpretation** *order-vec*: *order less-eq less*
⟨*proof*⟩

**lemma** *less-eqI* [*intro?*]: *length xs* = *length ys* ⟹ ∀ *i*<*length xs*. *xs* ! *i* ≤ *ys* ! *i*
⟹ *xs* ≤$_v$ *ys*
⟨*proof*⟩

**lemma** *le0* [*simp*, *intro*]: *zeroes* (*length xs*) ≤$_v$ *xs* ⟨*proof*⟩

**lemma** *le-list-update* [*simp*]:
**assumes** *xs* ≤$_v$ *ys* **and** *i* < *length ys* **and** *z* ≤ *ys* ! *i*
**shows** *xs*[*i* := *z*] ≤$_v$ *ys*
⟨*proof*⟩

**lemma** *le-Cons*: *x* # *xs* ≤$_v$ *y* # *ys* ⟷ *x* ≤ *y* ∧ *xs* ≤$_v$ *ys*
⟨*proof*⟩

**lemma** *zero-less*:
**assumes** *nonzero x*
**shows** *zeroes* (*length x*) <$_v$ *x*
⟨*proof*⟩

**lemma** *le-append*:

**assumes** *length xs* = *length vs*
**shows** *xs* @ *ys* $\leq_v$ *vs* @ *ws* $\longleftrightarrow$ *xs* $\leq_v$ *vs* $\wedge$ *ys* $\leq_v$ *ws*
⟨*proof*⟩

**lemma** *less-Cons*:
(*x* # *xs*) $<_v$ (*y* # *ys*) $\longleftrightarrow$ *length xs* = *length ys* $\wedge$ (*x* $\leq$ *y* $\wedge$ *xs* $<_v$ *ys* $\vee$ *x* < *y* $\wedge$ *xs* $\leq_v$ *ys*)
⟨*proof*⟩

**lemma** *le-length* [*dest*]:
**assumes** *xs* $\leq_v$ *ys*
**shows** *length xs* = *length ys*
⟨*proof*⟩

**lemma** *less-length* [*dest*]:
**assumes** *x* $<_v$ *y*
**shows** *length x* = *length y*
⟨*proof*⟩

**lemma** *less-append*:
**assumes** *xs* $<_v$ *vs* **and** *ys* $\leq_v$ *ws*
**shows** *xs* @ *ys* $<_v$ *vs* @ *ws*
⟨*proof*⟩

**lemma** *less-appendD*:
**assumes** *xs* @ *ys* $<_v$ *vs* @ *ws*
**and** *length xs* = *length vs*
**shows** *xs* $<_v$ *vs* $\vee$ *ys* $<_v$ *ws*
⟨*proof*⟩

**lemma** *less-append-cases*:
**assumes** *xs* @ *ys* $<_v$ *vs* @ *ws* **and** *length xs* = *length vs*
**obtains** *xs* $<_v$ *vs* **and** *ys* $\leq_v$ *ws* | *xs* $\leq_v$ *vs* **and** *ys* $<_v$ *ws*
⟨*proof*⟩

**lemma** *less-append-swap*:
**assumes** *x* @ *y* $<_v$ *u* @ *v*
**and** *length x* = *length u*
**shows** *y* @ *x* $<_v$ *v* @ *u*
⟨*proof*⟩

**lemma** *le-sum-list-less*:
**assumes** *xs* $\leq_v$ *ys*
**and** *sum-list xs* < *sum-list ys*
**shows** *xs* $<_v$ *ys*
⟨*proof*⟩

**lemma** *dotprod-le-right*:
**assumes** *v* $\leq_v$ *w*

**and** *length b* = *length w*
**shows** $b \cdot v \leq b \cdot w$
⟨*proof*⟩

**lemma** *dotprod-pointwise-le-right*:
**assumes** *length z* = *length u*
**and** *length u* = *length v*
**and** $\forall i{<}$*length v*. *u* ! *i* ≤ *v* ! *i*
**shows** $z \cdot u \leq z \cdot v$
⟨*proof*⟩

**lemma** *dotprod-le-left*:
**assumes** $v \leq_v w$
**and** *length b* = *length w*
**shows** $v \cdot b \leq w \cdot b$
⟨*proof*⟩

**lemma** *dotprod-le*:
**assumes** $x \leq_v u$ **and** $y \leq_v v$
**and** *length y* = *length x* **and** *length v* = *length u*
**shows** $x \cdot y \leq u \cdot v$
⟨*proof*⟩

**lemma** *dotprod-less-left*:
**assumes** *length b* = *length w*
**and** $0 \notin$ *set b*
**and** $v <_v w$
**shows** $v \cdot b < w \cdot b$
⟨*proof*⟩

**lemma** *le-append-swap*:
**assumes** *length y* = *length v*
**and** $x$ @ $y \leq_v w$ @ $v$
**shows** $y$ @ $x \leq_v v$ @ $w$
⟨*proof*⟩

**lemma** *le-append-swap-iff*:
**assumes** *length y* = *length v*
**shows** $y$ @ $x \leq_v v$ @ $w \longleftrightarrow x$ @ $y \leq_v w$ @ $v$
⟨*proof*⟩

**lemma** *unit-less*:
**assumes** $i < n$
**and** $x <_v$ (*zeroes n*)[$i$ := $b$]
**shows** $x$ ! $i < b \wedge (\forall j{<}n.\ j \neq i \longrightarrow x$ ! $j = 0)$
⟨*proof*⟩

**lemma** *le-sum-list-mono*:
**assumes** $xs \leq_v ys$

**shows** *sum-list xs* $\leq$ *sum-list ys*
⟨*proof*⟩

**lemma** *sum-list-less-diff-Ex*:
**assumes** $u \leq_v y$
**and** *sum-list u* < *sum-list y*
**shows** $\exists i{<}$*length y*. *u* ! *i* < *y* ! *i*
⟨*proof*⟩

**lemma** *less-vec-sum-list-less*:
**assumes** $v <_v w$
**shows** *sum-list v* < *sum-list w*
⟨*proof*⟩

**definition** *maxne0* :: *nat list* ⇒ *nat list* ⇒ *nat*
**where**
*maxne0 x a* =
(*if length x* = *length a* ∧ ($\exists i{<}$*length a*. *x* ! *i* ≠ *0*)
*then Max* {*a* ! *i* | *i*. *i* < *length a* ∧ *x* ! *i* ≠ *0*}
*else 0*)

**lemma** *maxne0-le-Max*:
*maxne0 x a* ≤ *Max* (*set a*)
⟨*proof*⟩

**lemma** *maxne0-Nil* [*simp*]:
*maxne0* [] *as* = *0*
*maxne0 xs* [] = *0*
⟨*proof*⟩

**lemma** *maxne0-Cons* [*simp*]:
*maxne0* (*x* # *xs*) (*a* # *as*) =
(*if length xs* = *length as then*
(*if x* = *0 then maxne0 xs as else max a* (*maxne0 xs as*))
*else 0*)
⟨*proof*⟩

**lemma** *maxne0-times-sum-list-gt-dotprod*:
**assumes** *length b* = *length ys*
**shows** *maxne0 ys b* ∗ *sum-list ys* ≥ *b* ⋅ *ys*
⟨*proof*⟩

**lemma** *max-times-sum-list-gt-dotprod*:
**assumes** *length b* = *length ys*
**shows** *Max* (*set b*) ∗ *sum-list ys* ≥ *b* ⋅ *ys*
⟨*proof*⟩

**lemma** *maxne0-mono*:
**assumes** $y \leq_v x$

**shows** *maxne0 y a* ≤ *maxne0 x a*
⟨*proof*⟩

**lemma** *all-leq-Max*:
**assumes** $x \leq_v y$
**and** $x \neq []$
**shows** $\forall xi \in set\ x.\ xi \leq Max\ (set\ y)$
⟨*proof*⟩

**lemma** *le-not-less-replicate*:
$\forall x \in set\ xs.\ x \leq b \Longrightarrow \neg\ xs <_v replicate\ (length\ xs)\ b \Longrightarrow xs = replicate\ (length\ xs)\ b$
⟨*proof*⟩

**lemma** *le-replicateI*: $\forall x \in set\ xs.\ x \leq b \Longrightarrow xs \leq_v replicate\ (length\ xs)\ b$
⟨*proof*⟩

**lemma** *le-take*:
**assumes** $x \leq_v y$ **and** $i \leq length\ x$ **shows** *take i x* $\leq_v$ *take i y*
⟨*proof*⟩

**lemma** *wf-less*:
*wf* $\{(x, y).\ x <_v y\}$
⟨*proof*⟩

## 1.3 Pointwise Subtraction

**definition** *vdiff* :: *nat list* ⇒ *nat list* ⇒ *nat list* (**infixl** $-_v$ *65*)
**where**
$w -_v v = map\ (\lambda i.\ w\ !\ i - v\ !\ i)\ [0\ ..<\ length\ w]$

**lemma** *vdiff-Nil* [*simp*]: $[] -_v [] = []$ ⟨*proof*⟩

**lemma** *upt-Cons-conv*:
**assumes** $j < n$
**shows** $[j..<n] = j\ \#\ [j+1..<n]$
⟨*proof*⟩

**lemma** *map-upt-Suc*: *map f* $[Suc\ m\ ..<\ Suc\ n] = map\ (f \circ Suc)\ [m\ ..<\ n]$
⟨*proof*⟩

**lemma** *vdiff-Cons* [*simp*]:
$(x\ \#\ xs) -_v (y\ \#\ ys) = (x - y)\ \#\ (xs -_v ys)$
⟨*proof*⟩

**lemma** *vdiff-alt-def*:
**assumes** *length w* = *length v*
**shows** $w -_v v = map\ (\lambda(x, y).\ x - y)\ (zip\ w\ v)$
⟨*proof*⟩

**lemma** *vdiff-dotprod-distr*:
**assumes** *length b* = *length w*
**and** $v \leq_v w$
**shows** $(w -_v v) \cdot b = w \cdot b - v \cdot b$
⟨*proof*⟩

**lemma** *sum-list-vdiff-distr* [*simp*]:
**assumes** $v \leq_v u$
**shows** *sum-list* $(u -_v v)$ = *sum-list u* − *sum-list v*
⟨*proof*⟩

**lemma** *vdiff-le*:
**assumes** $v \leq_v w$
**and** *length v* = *length x*
**shows** $v -_v x \leq_v w$
⟨*proof*⟩

**lemma** *mods-with-vec*:
**assumes** $v <_v w$
**and** $0 \notin$ *set b*
**and** *length b* = *length w*
**and** $(v \cdot b)$ *mod a* = $(w \cdot b)$ *mod a*
**shows** $((w -_v v) \cdot b)$ *mod a* = *0*
⟨*proof*⟩

**lemma** *mods-with-vec-2*:
**assumes** $v <_v w$
**and** $0 \notin$ *set b*
**and** *length b* = *length w*
**and** $(b \cdot v)$ *mod a* = $(b \cdot w)$ *mod a*
**shows** $(b \cdot (w -_v v))$ *mod a* = *0*
⟨*proof*⟩

## 1.4 The Lexicographic Order on Vectors

**abbreviation** *lex-less-than* (*-/* $<_{lex}$ *-* [*51*, *51*] *50*)
**where**
$xs <_{lex} ys \equiv (xs, ys) \in$ *lex less-than*

**definition** *rlex* (**infix** $<_{rlex}$ *50*)
**where**
$xs <_{rlex} ys \longleftrightarrow$ *rev xs* $<_{lex}$ *rev ys*

**lemma** *rev-le* [*simp*]:
*rev xs* $\leq_v$ *rev ys* $\longleftrightarrow$ *xs* $\leq_v$ *ys*
⟨*proof*⟩

**lemma** *rev-less* [*simp*]:

*rev xs* $<_v$ *rev ys* $\longleftrightarrow$ *xs* $<_v$ *ys*
⟨*proof*⟩

**lemma** *less-imp-lex*:
**assumes** *xs* $<_v$ *ys* **shows** *xs* $<_{lex}$ *ys*
⟨*proof*⟩

**lemma** *less-imp-rlex*:
**assumes** *xs* $<_v$ *ys* **shows** *xs* $<_{rlex}$ *ys*
⟨*proof*⟩

**lemma** *lex-not-sym*:
**assumes** *xs* $<_{lex}$ *ys*
**shows** $\neg$ *ys* $<_{lex}$ *xs*
⟨*proof*⟩

**lemma** *rlex-not-sym*:
**assumes** *xs* $<_{rlex}$ *ys*
**shows** $\neg$ *ys* $<_{rlex}$ *xs*
⟨*proof*⟩

**lemma** *lex-trans*:
**assumes** *x* $<_{lex}$ *y* **and** *y* $<_{lex}$ *z*
**shows** *x* $<_{lex}$ *z*
⟨*proof*⟩

**lemma** *rlex-trans*:
**assumes** *x* $<_{rlex}$ *y* **and** *y* $<_{rlex}$ *z*
**shows** *x* $<_{rlex}$ *z*
⟨*proof*⟩

**lemma** *lex-append-rightD*:
**assumes** *xs* @ *us* $<_{lex}$ *ys* @ *vs* **and** *length xs* = *length ys*
**and** $\neg$ *xs* $<_{lex}$ *ys*
**shows** *ys* = *xs* $\wedge$ *us* $<_{lex}$ *vs*
⟨*proof*⟩

**lemma** *rlex-Cons*:
*x* # *xs* $<_{rlex}$ *y* # *ys* $\longleftrightarrow$ *xs* $<_{rlex}$ *ys* $\vee$ *ys* = *xs* $\wedge$ *x* < *y* (**is** *?A* = *?B*)
⟨*proof*⟩

**lemma** *rlex-irrefl*:
$\neg$ *x* $<_{rlex}$ *x*
⟨*proof*⟩

## 1.5 Code Equations

**fun** *exists2*
**where**

*exists2 d P [] []* $\longleftrightarrow$ *False*
| *exists2 d P* (*x*#*xs*) (*y*#*ys*) $\longleftrightarrow$ *P x y* $\vee$ *exists2 d P xs ys*
| *exists2 d P - -* $\longleftrightarrow$ *d*

**lemma** *not-le-code* [*code-unfold*]: $\neg$ *xs* $\leq_v$ *ys* $\longleftrightarrow$ *exists2 True* (>) *xs ys*
⟨*proof*⟩

**end**

# 2 Homogeneous Linear Diophantine Equations

**theory** *Linear-Diophantine-Equations*
**imports** *List-Vector*
**begin**

**lemma** *lcm-div-le*:
**fixes** *a* :: *nat*
**shows** *lcm a b div b* $\leq$ *a*
⟨*proof*⟩

**lemma** *lcm-div-le'*:
**fixes** *a* :: *nat*
**shows** *lcm a b div a* $\leq$ *b*
⟨*proof*⟩

**lemma** *lcm-div-gt-0*:
**fixes** *a* :: *nat*
**assumes** *a* > *0* **and** *b* > *0*
**shows** *lcm a b div a* > *0*
⟨*proof*⟩

**lemma** *sum-list-list-update-Suc*:
**assumes** *i* < *length u*
**shows** *sum-list* (*u*[*i* := *Suc* (*u* ! *i*)]) = *Suc* (*sum-list u*)
⟨*proof*⟩

**lemma** *lessThan-conv*:
**assumes** *card A* = *n* **and** $\forall$ *x*$\in$*A*. *x* < *n*
**shows** *A* = {..<*n*}
⟨*proof*⟩

Given a non-empty list *xs* of *n* natural numbers, either there is a value in *xs* that is *0* modulo *n*, or there are two values whose moduli coincide.

**lemma** *list-mod-cases*:

**assumes** *length xs* = *n* **and** *n* > *0*
**shows** (∃ *x*∈*set xs*. *x mod n* = *0*) ∨
(∃ *i*<*length xs*. ∃ *j*<*length xs*. *i* ≠ *j* ∧ (*xs* ! *i*) *mod n* = (*xs* ! *j*) *mod n*)
⟨*proof*⟩

Homogeneous linear Diophantine equations: $a_1x_1 + \cdots + a_mx_m = b_1y_1 + \cdots + b_ny_n$

**locale** *hlde-ops* =
**fixes** *a b* :: *nat list*
**begin**

**abbreviation** *m* ≡ *length a*
**abbreviation** *n* ≡ *length b*

— The set of all solutions.
**definition** *Solutions* :: (*nat list* × *nat list*) *set*
**where**
*Solutions* = {(*x*, *y*). *a* ⋅ *x* = *b* ⋅ *y* ∧ *length x* = *m* ∧ *length y* = *n*}

**lemma** *in-Solutions-iff*:
(*x*, *y*) ∈ *Solutions* ⟷ *length x* = *m* ∧ *length y* = *n* ∧ *a* ⋅ *x* = *b* ⋅ *y*
⟨*proof*⟩
**definition** *Minimal-Solutions* :: (*nat list* × *nat list*) *set*
**where**
*Minimal-Solutions* = {(*x*, *y*) ∈ *Solutions*. *nonzero x* ∧
¬ (∃ (*u*, *v*) ∈ *Solutions*. *nonzero u* ∧ *u* @ *v* <$_v$ *x* @ *y*)}

**definition** *dij* :: *nat* ⇒ *nat* ⇒ *nat*
**where**
*dij i j* = *lcm* (*a* ! *i*) (*b* ! *j*) *div* (*a* ! *i*)

**definition** *eij* :: *nat* ⇒ *nat* ⇒ *nat*
**where**
*eij i j* = *lcm* (*a* ! *i*) (*b* ! *j*) *div* (*b* ! *j*)

**definition** *sij* :: *nat* ⇒ *nat* ⇒ (*nat list* × *nat list*)
**where**
*sij i j* = ((*zeroes m*)[*i* := *dij i j*], (*zeroes n*)[*j* := *eij i j*])

## 2.1 Further Constraints on Minimal Solutions

**definition** *Ej* :: *nat* ⇒ *nat list* ⇒ *nat set*
**where**
*Ej j x* = { *eij i j* − *1* | *i*. *i* < *length x* ∧ *x* ! *i* ≥ *dij i j* }

**definition** *Di* :: *nat* ⇒ *nat list* ⇒ *nat set*
**where**
*Di i y* = { *dij i j* − *1* | *j*. *j* < *length y* ∧ *y* ! *j* ≥ *eij i j* }

**definition** *Di'* :: *nat* ⇒ *nat list* ⇒ *nat set*
**where**
*Di' i y* = { *dij i* (*j* + *length b* − *length y*) − *1* | *j*. *j* < *length y* ∧ *y* ! *j* ≥ *eij i* (*j* + *length b* − *length y*) }

**lemma** *Ej-take-subset*:
*Ej j* (*take k x*) ⊆ *Ej j x*
⟨*proof*⟩

**lemma** *Di-take-subset*:
*Di i* (*take l y*) ⊆ *Di i y*
⟨*proof*⟩

**lemma** *Di'-drop-subset*:
*Di' i* (*drop l y*) ⊆ *Di' i y*
⟨*proof*⟩

**lemma** *finite-Ej*:
*finite* (*Ej j x*)
⟨*proof*⟩

**lemma** *finite-Di*:
*finite* (*Di i y*)
⟨*proof*⟩

**lemma** *finite-Di'*:
*finite* (*Di' i y*)
⟨*proof*⟩

**definition** *max-y* :: *nat list* ⇒ *nat* ⇒ *nat*
**where**
*max-y x j* = (*if j* < *n* ∧ *Ej j x* ≠ {} *then Min* (*Ej j x*) *else Max* (*set a*))

**definition** *max-x* :: *nat list* ⇒ *nat* ⇒ *nat*
**where**
*max-x y i* = (*if i* < *m* ∧ *Di i y* ≠ {} *then Min* (*Di i y*) *else Max* (*set b*))

**definition** *max-x'* :: *nat list* ⇒ *nat* ⇒ *nat*
**where**
*max-x' y i* = (*if i* < *m* ∧ *Di' i y* ≠ {} *then Min* (*Di' i y*) *else Max* (*set b*))

**lemma** *Min-Ej-le*:
**assumes** *j* < *n*
**and** *e* ∈ *Ej j x*
**and** *length x* ≤ *m*
**shows** *Min* (*Ej j x*) ≤ *Max* (*set a*) (**is** *?m* ≤ -)
⟨*proof*⟩

**lemma** *Min-Di-le*:

**assumes** $i < m$
**and** $e \in Di\ i\ y$
**and** $length\ y \leq n$
**shows** $Min\ (Di\ i\ y) \leq Max\ (set\ b)$ (**is** $?m \leq$ -)
⟨*proof*⟩

**lemma** *Min-Di'-le*:
**assumes** $i < m$
**and** $e \in Di'\ i\ y$
**and** $length\ y \leq n$
**shows** $Min\ (Di'\ i\ y) \leq Max\ (set\ b)$ (**is** $?m \leq$ -)
⟨*proof*⟩

**lemma** *max-y-le-take*:
**assumes** $length\ x \leq m$
**shows** $max\text{-}y\ x\ j \leq max\text{-}y\ (take\ k\ x)\ j$
⟨*proof*⟩

**lemma** *max-x-le-take*:
**assumes** $length\ y \leq n$
**shows** $max\text{-}x\ y\ i \leq max\text{-}x\ (take\ l\ y)\ i$
⟨*proof*⟩

**lemma** *max-x'-le-drop*:
**assumes** $length\ y \leq n$
**shows** $max\text{-}x'\ y\ i \leq max\text{-}x'\ (drop\ l\ y)\ i$
⟨*proof*⟩

**end**

**abbreviation** $Solutions \equiv hlde\text{-}ops.Solutions$
**abbreviation** $Minimal\text{-}Solutions \equiv hlde\text{-}ops.Minimal\text{-}Solutions$

**abbreviation** $dij \equiv hlde\text{-}ops.dij$
**abbreviation** $eij \equiv hlde\text{-}ops.eij$
**abbreviation** $sij \equiv hlde\text{-}ops.sij$

**declare** *hlde-ops.dij-def* [*code*]
**declare** *hlde-ops.eij-def* [*code*]
**declare** *hlde-ops.sij-def* [*code*]

**lemma** *Solutions-sym*: $(x, y) \in Solutions\ a\ b \longleftrightarrow (y, x) \in Solutions\ b\ a$
⟨*proof*⟩

**lemma** *Minimal-Solutions-imp-Solutions*: $(x, y) \in Minimal\text{-}Solutions\ a\ b \Longrightarrow (x, y) \in Solutions\ a\ b$
⟨*proof*⟩

**lemma** *Minimal-SolutionsI*:

**assumes** $(x, y) \in \mathit{Solutions}\ a\ b$
**and** $\mathit{nonzero}\ x$
**and** $\neg\ (\exists (u, v) \in \mathit{Solutions}\ a\ b.\ \mathit{nonzero}\ u \wedge u\ @\ v <_v x\ @\ y)$
**shows** $(x, y) \in \mathit{Minimal\text{-}Solutions}\ a\ b$
⟨*proof*⟩

**lemma** *minimize-nonzero-solution*:
**assumes** $(x, y) \in \mathit{Solutions}\ a\ b$ **and** $\mathit{nonzero}\ x$
**obtains** $u$ **and** $v$ **where** $u\ @\ v \leq_v x\ @\ y$ **and** $(u, v) \in \mathit{Minimal\text{-}Solutions}\ a\ b$
⟨*proof*⟩

**lemma** *Minimal-SolutionsI'*:
**assumes** $(x, y) \in \mathit{Solutions}\ a\ b$
**and** $\mathit{nonzero}\ x$
**and** $\neg\ (\exists (u, v) \in \mathit{Minimal\text{-}Solutions}\ a\ b.\ u\ @\ v <_v x\ @\ y)$
**shows** $(x, y) \in \mathit{Minimal\text{-}Solutions}\ a\ b$
⟨*proof*⟩

**lemma** *Minimal-Solutions-length*:
$(x, y) \in \mathit{Minimal\text{-}Solutions}\ a\ b \Longrightarrow \mathit{length}\ x = \mathit{length}\ a \wedge \mathit{length}\ y = \mathit{length}\ b$
⟨*proof*⟩

**lemma** *Minimal-Solutions-gt0*:
$(x, y) \in \mathit{Minimal\text{-}Solutions}\ a\ b \Longrightarrow \mathit{zeroes}\ (\mathit{length}\ x) <_v x$
⟨*proof*⟩

**lemma** *Minimal-Solutions-sym*:
**assumes** $0 \notin \mathit{set}\ a$ **and** $0 \notin \mathit{set}\ b$
**shows** $(xs, ys) \in \mathit{Minimal\text{-}Solutions}\ a\ b \longrightarrow (ys, xs) \in \mathit{Minimal\text{-}Solutions}\ b\ a$
⟨*proof*⟩

**locale** *hlde* = *hlde-ops* +
**assumes** *no0*: $0 \notin \mathit{set}\ a\ 0 \notin \mathit{set}\ b$
**begin**

**lemma** *nonzero-Solutions-iff*:
**assumes** $(x, y) \in \mathit{Solutions}$
**shows** $\mathit{nonzero}\ x \longleftrightarrow \mathit{nonzero}\ y$
⟨*proof*⟩

**lemma** *Minimal-Solutions-min*:
**assumes** $(x, y) \in \mathit{Minimal\text{-}Solutions}$
**and** $u\ @\ v <_v x\ @\ y$
**and** $a \cdot u = b \cdot v$
**and** [*simp*]: $\mathit{length}\ u = m$
**and** *non0*: $\mathit{nonzero}\ (u\ @\ v)$
**shows** *False*
⟨*proof*⟩

**lemma** *Solutions-snd-not-0*:
**assumes** $(x, y) \in$ *Solutions*
**and** *nonzero x*
**shows** *nonzero y*
⟨*proof*⟩

**end**

## 2.2 Pointwise Restricting Solutions

Constructing the list of $u$ vectors from Huet's proof [1], satisfying

- $\forall i{<}$*length u*. *u* ! *i* $\leq$ *y* ! *i* and
- *0* $<$ *sum-list u* $\leq a_k$.

Given $y$, increment a "previous" $u$ vector at first position starting from $i$ where $u$ is strictly smaller than $y$. If this is not possible, return $u$ unchanged.

**function** *inc* :: *nat list* ⇒ *nat* ⇒ *nat list* ⇒ *nat list*
**where**
*inc y i u* =
(*if i* < *length y then*
*if u* ! *i* < *y* ! *i then u*[*i* := *u* ! *i* + *1*]
*else inc y* (*Suc i*) *u*
*else u*)
⟨*proof*⟩
**termination** *inc*
⟨*proof*⟩

**declare** *inc.simps* [*simp del*]

Starting from the 0-vector produce $u$s by iteratively incrementing with respect to $y$.

**definition** *huets-us* :: *nat list* ⇒ *nat* ⇒ *nat list* (**u** *1000*)
**where**
**u** *y i* = ((*inc y 0*) ^^ *Suc i*) (*zeroes* (*length y*))

**lemma** *huets-us-simps* [*simp*]:
**u** *y 0* = *inc y 0* (*zeroes* (*length y*))
**u** *y* (*Suc i*) = *inc y 0* (**u** *y i*)
⟨*proof*⟩

**lemma** *length-inc* [*simp*]: *length* (*inc y i u*) = *length u*
⟨*proof*⟩

**lemma** *length-us* [*simp*]:
*length* (**u** *y i*) = *length y*

⟨*proof*⟩

*inc* produces vectors that are pointwise smaller than $y$

**lemma** *inc-le*:
**assumes** *length* $u$ = *length* $y$ **and** $i$ < *length* $y$ **and** $u \leq_v y$
**shows** *inc* $y$ $i$ $u \leq_v y$
⟨*proof*⟩

**lemma** *us-le*:
**assumes** *length* $y > 0$
**shows** **u** $y$ $i \leq_v y$
⟨*proof*⟩

**lemma** *sum-list-inc-le*:
$u \leq_v y \Longrightarrow$ *sum-list* (*inc* $y$ $i$ $u$) ≤ *sum-list* $y$
⟨*proof*⟩

**lemma** *sum-list-inc-gt0*:
**assumes** *sum-list* $u > 0$ **and** *length* $y$ = *length* $u$
**shows** *sum-list* (*inc* $y$ $i$ $u$) > 0
⟨*proof*⟩

**lemma** *sum-list-inc-gt0'*:
**assumes** *length* $u$ = *length* $y$ **and** $i$ < *length* $y$ **and** $y$ ! $i > 0$ **and** $j \leq i$
**shows** *sum-list* (*inc* $y$ $j$ $u$) > 0
⟨*proof*⟩

**lemma** *sum-list-us-gt0*:
**assumes** *sum-list* $y \neq 0$
**shows** 0 < *sum-list* (**u** $y$ $i$)
⟨*proof*⟩

**lemma** *sum-list-inc-le'*:
**assumes** *length* $u$ = *length* $y$
**shows** *sum-list* (*inc* $y$ $i$ $u$) ≤ *sum-list* $u$ + 1
⟨*proof*⟩

**lemma** *sum-list-us-le*:
*sum-list* (**u** $y$ $i$) ≤ $i$ + 1
⟨*proof*⟩

**lemma** *sum-list-us-bounded*:
**assumes** $i < k$
**shows** *sum-list* (**u** $y$ $i$) ≤ $k$
⟨*proof*⟩

**lemma** *sum-list-inc-eq-sum-list-Suc*:
**assumes** *length* $u$ = *length* $y$ **and** $i$ < *length* $y$
**and** $\exists j \geq i.\ j <$ *length* $y \wedge u$ ! $j < y$ ! $j$

**shows** *sum-list* (*inc y i u*) = *Suc* (*sum-list u*)
⟨*proof*⟩

**lemma** *sum-list-us-eq*:
**assumes** $i <$ *sum-list y*
**shows** *sum-list* (**u** *y i*) = *i* + *1*
⟨*proof*⟩

**lemma** *inc-ge*: *length u* = *length y* $\Longrightarrow$ *u* $\leq_v$ *inc y i u*
⟨*proof*⟩

**lemma** *us-le-mono*:
**assumes** $i < j$
**shows** **u** *y i* $\leq_v$ **u** *y j*
⟨*proof*⟩

**lemma** *us-mono*:
**assumes** $i < j$ **and** $j <$ *sum-list y*
**shows** **u** *y i* $<_v$ **u** *y j*
⟨*proof*⟩

**context** *hlde*
**begin**

**lemma** *max-coeff-bound-right*:
**assumes** (*xs*, *ys*) ∈ *Minimal-Solutions*
**shows** ∀ *x* ∈ *set xs*. *x* ≤ *maxne0 ys b* (**is** ∀ *x*∈*set xs*. *x* ≤ *?m*)
⟨*proof*⟩

Proof of Lemma 1 of Huet's paper.

**lemma** *max-coeff-bound*:
**assumes** (*xs*, *ys*) ∈ *Minimal-Solutions*
**shows** (∀ *x* ∈ *set xs*. *x* ≤ *maxne0 ys b*) ∧ (∀ *y* ∈ *set ys*. *y* ≤ *maxne0 xs a*)
⟨*proof*⟩

**lemma** *max-coeff-bound′*:
**assumes** (*x*, *y*) ∈ *Minimal-Solutions*
**shows** ∀ *i*<*length x*. *x* ! *i* ≤ *Max* (*set b*) **and** ∀ *j*<*length y*. *y* ! *j* ≤ *Max* (*set a*)
⟨*proof*⟩

**lemma** *Minimal-Solutions-alt-def*:
*Minimal-Solutions* = {(*x*, *y*)∈*Solutions*.
(*x*, *y*) ≠ (*zeroes m*, *zeroes n*) ∧
*x* $\leq_v$ *replicate m* (*Max* (*set b*)) ∧
*y* $\leq_v$ *replicate n* (*Max* (*set a*)) ∧
¬ (∃(*u*, *v*)∈*Solutions*. *nonzero u* ∧ *u* @ *v* $<_v$ *x* @ *y*)}
⟨*proof*⟩

## 2.3 Special Solutions

**definition** *Special-Solutions* :: (*nat list* × *nat list*) *set*
  **where**
    *Special-Solutions* = {*sij i j* | *i j*. *i* < *m* ∧ *j* < *n*}

**lemma** *dij-neq-0*:
  **assumes** *i* < *m*
    **and** *j* < *n*
  **shows** *dij i j* ≠ *0*
⟨*proof*⟩

**lemma** *eij-neq-0*:
  **assumes** *i* < *m*
    **and** *j* < *n*
  **shows** *eij i j* ≠ *0*
⟨*proof*⟩

**lemma** *Special-Solutions-in-Solutions*:
  *x* ∈ *Special-Solutions* ⟹ *x* ∈ *Solutions*
  ⟨*proof*⟩

**lemma** *Special-Solutions-in-Minimal-Solutions*:
  **assumes** (*x*, *y*) ∈ *Special-Solutions*
  **shows** (*x*, *y*) ∈ *Minimal-Solutions*
⟨*proof*⟩

**lemma** *non-special-solution-non-minimal*:
  **assumes** (*x*, *y*) ∈ *Solutions* − *Special-Solutions*
    **and** *ij*: *i* < *m j* < *n*
    **and** *x* ! *i* ≥ *dij i j* **and** *y* ! *j* ≥ *eij i j*
  **shows** (*x*, *y*) ∉ *Minimal-Solutions*
⟨*proof*⟩

## 2.4 Huet's conditions

**definition** *cond-A xs ys* ⟷ (∀ *x*∈*set xs*. *x* ≤ *maxne0 ys b*)

**definition** *cond-B x* ⟷
  (∀ *k*≤*m*. *take k a* · *take k x* ≤ *b* · *map* (*max-y* (*take k x*)) [*0* ..< *n*])

**definition** *boundr x y* ⟷ (∀ *j*<*n*. *y* ! *j* ≤ *max-y x j*)

**definition** *cond-D x y* ⟷ (∀ *l*≤*n*. *take l b* · *take l y* ≤ *a* · *x*)

## 2.5 New conditions: facilitating generation of candidates from right to left

**definition** *subdprodr y* $\longleftrightarrow$
$(\forall\, l{\leq}n.\ take\ l\ b \cdot take\ l\ y \leq a \cdot map\ (max\text{-}x\ (take\ l\ y))\ [0\ ..{<}\ m])$

**definition** *subdprodl x y* $\longleftrightarrow$ $(\forall\, k{\leq}m.\ take\ k\ a \cdot take\ k\ x \leq b \cdot y)$

**definition** *boundl x y* $\longleftrightarrow$ $(\forall\, i{<}m.\ x\ !\ i \leq max\text{-}x\ y\ i)$

**lemma** *boundr*:
**assumes** *min*: $(x,\ y) \in$ *Minimal-Solutions*
**and** $(x,\ y) \notin$ *Special-Solutions*
**shows** *boundr x y*
⟨*proof*⟩

**lemma** *boundl*:
**assumes** *min*: $(x,\ y) \in$ *Minimal-Solutions*
**and** $(x,\ y) \notin$ *Special-Solutions*
**shows** *boundl x y*
⟨*proof*⟩

**lemma** *Solution-imp-cond-D*:
**assumes** $(x,\ y) \in$ *Solutions*
**shows** *cond-D x y*
⟨*proof*⟩

**lemma** *Solution-imp-subdprodl*:
**assumes** $(x,\ y) \in$ *Solutions*
**shows** *subdprodl x y*
⟨*proof*⟩

**theorem** *conds*:
**assumes** *min*: $(x,\ y) \in$ *Minimal-Solutions*
**shows** *cond-A*: *cond-A x y*
**and** *cond-B*: $(x,\ y) \notin$ *Special-Solutions* $\Longrightarrow$ *cond-B x*
**and** $(x,\ y) \notin$ *Special-Solutions* $\Longrightarrow$ *boundr x y*
**and** *cond-D*: *cond-D x y*
**and** *subdprodr*: $(x,\ y) \notin$ *Special-Solutions* $\Longrightarrow$ *subdprodr y*
**and** *subdprodl*: *subdprodl x y*
⟨*proof*⟩

**lemma** *le-imp-Ej-subset*:
**assumes** $u \leq_v x$
**shows** *Ej j u* $\subseteq$ *Ej j x*
⟨*proof*⟩

**lemma** *le-imp-max-y-ge*:

**assumes** $u \leq_v x$
**and** *length* $x \leq m$
**shows** *max-y* $u\ j \geq$ *max-y* $x\ j$
⟨*proof*⟩

**lemma** *le-imp-Di-subset*:
**assumes** $v \leq_v y$
**shows** *Di* $i\ v \subseteq$ *Di* $i\ y$
⟨*proof*⟩

**lemma** *le-imp-max-x-ge*:
**assumes** $v \leq_v y$
**and** *length* $y \leq n$
**shows** *max-x* $v\ i \geq$ *max-x* $y\ i$
⟨*proof*⟩

**end**

**end**

**theory** *Sorted-Wrt*
**imports** *Main*
**begin**

**lemma** *sorted-wrt-filter*:
*sorted-wrt* $P\ xs \Longrightarrow$ *sorted-wrt* $P$ (*filter* $Q\ xs$)
⟨*proof*⟩

**lemma** *sorted-wrt-map-mono*:
**assumes** *sorted-wrt* $Q\ xs$
**and** $\bigwedge x\ y.\ Q\ x\ y \Longrightarrow P\ (f\ x)\ (f\ y)$
**shows** *sorted-wrt* $P$ (*map* $f\ xs$)
⟨*proof*⟩

**lemma** *sorted-wrt-concat-map-map*:
**assumes** *sorted-wrt* $Q\ xs$
**and** *sorted-wrt* $Q\ ys$
**and** $\bigwedge a\ x\ y.\ Q\ x\ y \Longrightarrow P\ (f\ x\ a)\ (f\ y\ a)$
**and** $\bigwedge x\ y\ u\ v.\ x \in$ *set* $xs \Longrightarrow y \in$ *set* $xs \Longrightarrow Q\ u\ v \Longrightarrow P\ (f\ x\ u)\ (f\ y\ v)$
**shows** *sorted-wrt* $P$ $[f\ x\ y\ .\ y \leftarrow ys,\ x \leftarrow xs]$
⟨*proof*⟩

**lemma** *sorted-wrt-concat-map*:
**assumes** *sorted-wrt* $P$ (*map* $h\ xs$)
**and** $\bigwedge x.\ x \in$ *set* $xs \Longrightarrow$ *sorted-wrt* $P$ (*map* $h$ ($f\ x$))
**and** $\bigwedge x\ y\ u\ v.\ P\ (h\ x)\ (h\ y) \Longrightarrow x \in$ *set* $xs \Longrightarrow y \in$ *set* $xs \Longrightarrow u \in$ *set* ($f\ x$)
$\Longrightarrow v \in$ *set* ($f\ y$) $\Longrightarrow P\ (h\ u)\ (h\ v)$
**shows** *sorted-wrt* $P$ (*concat* (*map* (*map* $h \circ f$) $xs$))

⟨*proof*⟩

**lemma** *sorted-wrt-map-distr*:
**assumes** *sorted-wrt* (λ*x y*. *P x y*) (*map f xs*)
**shows** *sorted-wrt* (λ*x y*. *P* (*f x*) (*f y*)) *xs*
⟨*proof*⟩

**lemma** *sorted-wrt-tl*:
*xs* ≠ [] ⟹ *sorted-wrt P xs* ⟹ *sorted-wrt P* (*tl xs*)
⟨*proof*⟩

**end**

# 3 Minimization

**theory** *Minimize-Wrt*
**imports** *Sorted-Wrt*
**begin**

**fun** *minimize-wrt*
**where**
*minimize-wrt P* [] = []
| *minimize-wrt P* (*x* # *xs*) = *x* # *filter* (*P x*) (*minimize-wrt P xs*)

**lemma** *minimize-wrt-subset*: *set* (*minimize-wrt P xs*) ⊆ *set xs*
⟨*proof*⟩

**lemmas** *minimize-wrtD* = *minimize-wrt-subset* [*THEN subsetD*]

**lemma** *sorted-wrt-minimize-wrt*:
*sorted-wrt P* (*minimize-wrt P xs*)
⟨*proof*⟩

**lemma** *sorted-wrt-imp-sorted-wrt-minimize-wrt*:
*sorted-wrt Q xs* ⟹ *sorted-wrt Q* (*minimize-wrt P xs*)
⟨*proof*⟩

**lemma** *in-minimize-wrt-False*:
**assumes** ⋀*x y*. *Q x y* ⟹ ¬ *Q y x*
**and** *sorted-wrt Q xs*
**and** *x* ∈ *set* (*minimize-wrt P xs*)
**and** ¬ *P y x* **and** *Q y x* **and** *y* ∈ *set xs* **and** *y* ≠ *x*
**shows** *False*
⟨*proof*⟩

**lemma** *in-minimize-wrtI*:
**assumes** *x* ∈ *set xs*
**and** ∀ *y*∈*set xs*. *P y x*
**shows** *x* ∈ *set* (*minimize-wrt P xs*)

⟨*proof*⟩

**lemma** *minimize-wrt-eq*:
**assumes** *distinct xs* **and** $\bigwedge x\ y.\ x \in set\ xs \Longrightarrow y \in set\ xs \Longrightarrow P\ x\ y \longleftrightarrow Q\ x\ y \vee x = y$
**shows** *minimize-wrt P xs* = *minimize-wrt Q xs*
⟨*proof*⟩

**lemma** *minimize-wrt-ni*:
**assumes** $x \in set\ xs$
**and** $x \notin set\ (minimize\text{-}wrt\ Q\ xs)$
**shows** $\exists\, y \in set\ xs.\ (\neg\ Q\ y\ x) \wedge x \neq y$
⟨*proof*⟩

**lemma** *in-minimize-wrtD*:
**assumes** $\bigwedge x\ y.\ Q\ x\ y \Longrightarrow \neg\ Q\ y\ x$
**and** *sorted-wrt Q xs*
**and** $x \in set\ (minimize\text{-}wrt\ P\ xs)$
**and** $\bigwedge x\ y.\ \neg\ P\ x\ y \Longrightarrow Q\ x\ y$
**and** $\bigwedge x.\ P\ x\ x$
**shows** $x \in set\ xs \wedge (\forall\, y{\in}set\ xs.\ P\ y\ x)$
⟨*proof*⟩

**lemma** *in-minimize-wrt-iff*:
**assumes** $\bigwedge x\ y.\ Q\ x\ y \Longrightarrow \neg\ Q\ y\ x$
**and** *sorted-wrt Q xs*
**and** $\bigwedge x\ y.\ \neg\ P\ x\ y \Longrightarrow Q\ x\ y$
**and** $\bigwedge x.\ P\ x\ x$
**shows** $x \in set\ (minimize\text{-}wrt\ P\ xs) \longleftrightarrow x \in set\ xs \wedge (\forall\, y{\in}set\ xs.\ P\ y\ x)$
⟨*proof*⟩

**lemma** *set-minimize-wrt*:
**assumes** $\bigwedge x\ y.\ Q\ x\ y \Longrightarrow \neg\ Q\ y\ x$
**and** *sorted-wrt Q xs*
**and** $\bigwedge x\ y.\ \neg\ P\ x\ y \Longrightarrow Q\ x\ y$
**and** $\bigwedge x.\ P\ x\ x$
**shows** $set\ (minimize\text{-}wrt\ P\ xs) = \{x \in set\ xs.\ \forall\, y{\in}set\ xs.\ P\ y\ x\}$
⟨*proof*⟩

**lemma** *minimize-wrt-append*:
**assumes** $\forall\, x{\in}set\ xs.\ \forall\, y{\in}set\ (xs\ @\ ys).\ P\ y\ x$
**shows** $minimize\text{-}wrt\ P\ (xs\ @\ ys) = xs\ @\ filter\ (\lambda y.\ \forall\, x{\in}set\ xs.\ P\ x\ y)\ (minimize\text{-}wrt\ P\ ys)$
⟨*proof*⟩

**end**

**theory** *Simple-Algorithm*

**imports**
*Linear-Diophantine-Equations*
*Minimize-Wrt*
**begin**

**lemma** *concat-map-nth0*: $xs \neq [] \Longrightarrow f\ (xs\ !\ 0) \neq [] \Longrightarrow concat\ (map\ f\ xs)\ !\ 0 = f\ (xs\ !\ 0)\ !\ 0$
⟨*proof*⟩

## 3.1 Reverse-Lexicographic Enumeration of Potential Minimal Solutions

**fun** *rlex2* :: $(nat\ list \times nat\ list) \Rightarrow (nat\ list \times nat\ list) \Rightarrow bool$ (**infix** $<_{rlex2}$ *50*)
**where**
$(xs,\ ys) <_{rlex2} (us,\ vs) \longleftrightarrow xs\ @\ ys <_{rlex}\ us\ @\ vs$

**lemma** *rlex2-irrefl*:
$\neg\ x <_{rlex2} x$
⟨*proof*⟩

**lemma** *rlex2-not-sym*: $x <_{rlex2} y \Longrightarrow \neg\ y <_{rlex2} x$
⟨*proof*⟩

**lemma** *less-imp-rlex2*: $\neg\ (case\ x\ of\ (x,\ y) \Rightarrow \lambda(u,\ v).\ \neg\ x\ @\ y <_v u\ @\ v)\ y \Longrightarrow x <_{rlex2} y$
⟨*proof*⟩

Generate all lists (of natural numbers) of length $n$ with elements bounded by $B$.

**fun** *gen* :: $nat \Rightarrow nat \Rightarrow nat\ list\ list$
**where**
$gen\ B\ 0 = [[]]$
| $gen\ B\ (Suc\ n) = [x\#xs\ .\ xs \leftarrow gen\ B\ n,\ x \leftarrow [0\ ..<\ B\ +\ 1]]$

**definition** $generate\ A\ B\ m\ n = tl\ [(x,\ y)\ .\ y \leftarrow gen\ B\ n,\ x \leftarrow gen\ A\ m]$

**definition** $check\ a\ b = filter\ (\lambda(x,\ y).\ a \cdot x = b \cdot y)$

**definition** $minimize = minimize\text{-}wrt\ (\lambda(x,\ y)\ (u,\ v).\ \neg\ x\ @\ y <_v u\ @\ v)$

**definition** *solutions a b* =
(*let A* = *Max* (*set b*); *B* = *Max* (*set a*); *m* = *length a*; *n* = *length b*
*in minimize* (*check a b* (*generate A B m n*)))

**lemma** *set-gen*: $set\ (gen\ B\ n) = \{xs.\ length\ xs = n \wedge (\forall\, i{<}n.\ xs\ !\ i \leq B)\}$ (**is** *- = ?A n*)
⟨*proof*⟩

**abbreviation** *gen2 A B m n* ≡ [(*x*, *y*) . *y* ← *gen B n*, *x* ← *gen A m*]

**lemma** *sorted-wrt-gen*:
*sorted-wrt* (<$_{rlex}$) (*gen B n*)
⟨*proof*⟩

**lemma** *sorted-wrt-gen2*: *sorted-wrt* (<$_{rlex2}$) (*gen2 A B m n*)
⟨*proof*⟩

**lemma** *gen-ne* [*simp*]: *gen B n* ≠ [] ⟨*proof*⟩

**lemma** *gen2-ne*: *gen2 A B m n* ≠ [] ⟨*proof*⟩

**lemma** *sorted-wrt-generate*: *sorted-wrt* (<$_{rlex2}$) (*generate A B m n*)
⟨*proof*⟩

**abbreviation** *check-generate a b* ≡ *check a b* (*generate* (*Max* (*set b*)) (*Max* (*set a*)) (*length a*) (*length b*))

**lemma** *sorted-wrt-check-generate*: *sorted-wrt* (<$_{rlex2}$) (*check-generate a b*)
⟨*proof*⟩

**lemma** *in-tl-gen2*: *x* ∈ *set* (*tl* (*gen2 A B m n*)) ⟹ *x* ∈ *set* (*gen2 A B m n*)
⟨*proof*⟩

**lemma** *gen-nth0* [*simp*]: *gen B n* ! *0* = *zeroes n*
⟨*proof*⟩

**lemma** *gen2-nth0* [*simp*]:
*gen2 A B m n* ! *0* = (*zeroes m*, *zeroes n*)
⟨*proof*⟩

**lemma** *set-gen2*:
*set* (*gen2 A B m n*) = {(*x*, *y*). *length x* = *m* ∧ *length y* = *n* ∧ (∀ *i*<*m*. *x* ! *i* ≤ *A*) ∧ (∀ *j*<*n*. *y* ! *j* ≤ *B*)}
⟨*proof*⟩

**lemma** *gen2-unique*:
**assumes** *i* < *j*
**and** *j* < *length* (*gen2 A B m n*)
**shows** *gen2 A B m n* ! *i* ≠ *gen2 A B m n* ! *j*
⟨*proof*⟩

**lemma** *zeroes-ni-tl-gen2*:
(*zeroes m*, *zeroes n*) ∉ *set* (*tl* (*gen2 A B m n*))
⟨*proof*⟩

**lemma** *set-generate*:
*set* (*generate A B m n*) = {(*x*, *y*). (*x*, *y*) ≠ (*zeroes m*, *zeroes n*) ∧ (*x*, *y*) ∈ *set*

$(gen2\ A\ B\ m\ n)\}$
⟨*proof*⟩

**lemma** *set-check-generate*:
$set\ (check\text{-}generate\ a\ b) = \{(x,\ y).$
$(x,\ y) \neq (zeroes\ (length\ a),\ zeroes\ (length\ b)) \wedge$
$length\ x = length\ a \wedge length\ y = length\ b \wedge a \cdot x = b \cdot y \wedge$
$(\forall\, i{<}length\ a.\ x\ !\ i \leq Max\ (set\ b)) \wedge (\forall\, j{<}length\ b.\ y\ !\ j \leq Max\ (set\ a))\}$
⟨*proof*⟩

**lemma** *set-minimize-check-generate*:
$set\ (minimize\ (check\text{-}generate\ a\ b)) =$
$\{(x,\ y){\in}set\ (check\text{-}generate\ a\ b).\ \neg\ (\exists\,(u,\ v){\in}set\ (check\text{-}generate\ a\ b).\ u\ @\ v <_v x\ @\ y)\}$
⟨*proof*⟩

**lemma** *set-solutions-iff*:
$set\ (solutions\ a\ b) =$
$\{(x,\ y) \in set\ (check\text{-}generate\ a\ b).\ \neg\ (\exists\,(u,\ v){\in}set\ (check\text{-}generate\ a\ b).\ u\ @\ v <_v x\ @\ y)\}$
⟨*proof*⟩

### 3.1.1 Completeness: every minimal solution is generated by *solutions*

**lemma** (**in** *hlde*) *solutions-complete*:
$Minimal\text{-}Solutions \subseteq set\ (solutions\ a\ b)$
⟨*proof*⟩

### 3.1.2 Correctness: *solutions* generates only minimal solutions.

**lemma** (**in** *hlde*) *solutions-sound*:
$set\ (solutions\ a\ b) \subseteq Minimal\text{-}Solutions$
⟨*proof*⟩

**lemma** (**in** *hlde*) *set-solutions* [*simp*]: $set\ (solutions\ a\ b) = Minimal\text{-}Solutions$
⟨*proof*⟩

**end**

# 4 Computing Minimal Complete Sets of Solutions

**theory** *Algorithm*
**imports** *Simple-Algorithm*
**begin**

**lemma** *all-Suc-le-conv*: $(\forall\, i{\leq}Suc\ n.\ P\ i) \longleftrightarrow P\ 0 \wedge (\forall\, i{\leq}n.\ P\ (Suc\ i))$
⟨*proof*⟩

**lemma** *concat-map-filter-filter*:
**assumes** $\bigwedge x.\ x \in set\ xs \Longrightarrow \neg\ Q\ x \Longrightarrow filter\ P\ (f\ x) = []$
**shows** $concat\ (map\ (filter\ P \circ f)\ (filter\ Q\ xs)) = concat\ (map\ (filter\ P \circ f)\ xs)$
⟨*proof*⟩

**lemma** *filter-pairs-conj*:
$filter\ (\lambda(x,\ y).\ P\ x\ y \wedge Q\ y)\ xs = filter\ (\lambda(x,\ y).\ P\ x\ y)\ (filter\ (Q \circ snd)\ xs)$
⟨*proof*⟩

**lemma** *concat-map-filter*:
$concat\ (map\ f\ (filter\ P\ xs)) = concat\ (map\ (\lambda x.\ if\ P\ x\ then\ f\ x\ else\ [])\ xs)$
⟨*proof*⟩

**fun** *alls*
**where**
$alls\ B\ [] = [([],\ 0)]$
| $alls\ B\ (a\ \#\ as) = [(x\ \#\ xs,\ s + a * x).\ (xs,\ s) \leftarrow alls\ B\ as,\ x \leftarrow [0\ ..<\ B + 1]]$

**lemma** *alls-ne* [*simp*]:
$alls\ B\ as \neq []$
⟨*proof*⟩

**lemma** *set-alls*: $set\ (alls\ B\ a) =$
$\{(x,\ s).\ length\ x = length\ a \wedge (\forall i{<}length\ a.\ x\ !\ i \leq B) \wedge s = a \cdot x\}$
(**is** *?L a* = *?R a*)
⟨*proof*⟩

**lemma** *alls-nth0* [*simp*]: $alls\ A\ as\ !\ 0 = (zeroes\ (length\ as),\ 0)$
⟨*proof*⟩

**lemma** *alls-Cons-tl-conv*: $alls\ A\ as = (zeroes\ (length\ as),\ 0)\ \#\ tl\ (alls\ A\ as)$
⟨*proof*⟩

**lemma** *sorted-wrt-alls*:
$sorted\text{-}wrt\ (<_{rlex})\ (map\ fst\ (alls\ B\ xs))$
⟨*proof*⟩

**definition** $alls2\ A\ B\ a\ b = [(xs,\ ys).\ ys \leftarrow alls\ B\ b,\ xs \leftarrow alls\ A\ a]$

**lemma** *alls2-ne* [*simp*]:
$alls2\ A\ B\ a\ b \neq []$
⟨*proof*⟩

**lemma** *set-alls2*:

*set (alls2 A B a b) = {((x, s), (y, t)). length x = length a ∧ length y = length b*
*∧*
*(∀ i<length a. x ! i ≤ A) ∧ (∀ j<length b. y ! j ≤ B) ∧ s = a · x ∧ t = b · y}*
⟨*proof*⟩

**lemma** *alls2-nth0* [*simp*]: *alls2 A B as bs ! 0 = ((zeroes (length as), 0), (zeroes (length bs), 0))*
⟨*proof*⟩

**lemma** *alls2-Cons-tl-conv*: *alls2 A B as bs =*
*((zeroes (length as), 0), (zeroes (length bs), 0)) # tl (alls2 A B as bs)*
⟨*proof*⟩

**abbreviation** *gen2*
**where**
*gen2 A B a b ≡ map (λ(x, y). (fst x, fst y)) (alls2 A B a b)*

**lemma** *sorted-wrt-gen2*:
*sorted-wrt (<rlex2) (gen2 A B a b)*
⟨*proof*⟩

**definition** *generate′*
**where**
*generate′ A B a b = tl (map (λ(x, y). (fst x, fst y)) (alls2 A B a b))*

**lemma** *sorted-wrt-generate′*:
*sorted-wrt (<rlex2) (generate′ A B a b)*
⟨*proof*⟩

**lemma** *gen2-nth0* [*simp*]:
*gen2 A B a b ! 0 = (zeroes (length a), zeroes (length b))*
⟨*proof*⟩

**lemma** *gen2-ne* [*simp*, *intro*]: *gen2 m n b c ≠ []* ⟨*proof*⟩

**lemma** *in-generate′*: *x ∈ set (generate′ m n c b) ⟹ x ∈ set (gen2 m n c b)*
⟨*proof*⟩

**definition** *cond-cons P = (λ(ys, s). case ys of [] ⇒ True | ys ⇒ P ys s)*

**lemma** *cond-cons-simp* [*simp*]:
*cond-cons P ([], s) = True*
*cond-cons P (x # xs, s) = P (x # xs) s*
⟨*proof*⟩

**fun** *suffs*
**where**
*suffs P as (xs, s) ⟷*
*length xs = length as ∧*

$s = as \cdot xs \wedge$
$(\forall i{\leq}length\ xs.\ cond\text{-}cons\ P\ (drop\ i\ xs,\ drop\ i\ as \cdot drop\ i\ xs))$
**declare** *suffs.simps* [*simp del*]

**lemma** *suffs-Nil* [*simp*]: *suffs P* [] ([], *s*) $\longleftrightarrow$ *s = 0*
⟨*proof*⟩

**lemma** *suffs-Cons*:
*suffs P* (*a* # *as*) (*x* # *xs*, *s*) $\longleftrightarrow$
*s = a* ∗ *x* + *as* · *xs* ∧ *cond-cons P* (*x* # *xs*, *s*) ∧ *suffs P as* (*xs*, *as* · *xs*)
⟨*proof*⟩

## 4.1 The Algorithm

**fun** *maxne0-impl*
**where**
*maxne0-impl* [] *a = 0*
| *maxne0-impl x* [] *= 0*
| *maxne0-impl* (*x*#*xs*) (*a*#*as*) = (*if x > 0 then max a* (*maxne0-impl xs as*) *else maxne0-impl xs as*)

**lemma** *maxne0-impl*:
**assumes** *length x = length a*
**shows** *maxne0-impl x a = maxne0 x a*
⟨*proof*⟩

**lemma** *maxne0-impl-le*:
*maxne0-impl x a* ≤ *Max* (*set* (*a*::*nat list*))
⟨*proof*⟩

**context**
**fixes** *a b* :: *nat list*
**begin**

**definition** *special-solutions* :: (*nat list* × *nat list*) *list*
**where**
*special-solutions* = [*sij a b i j* . *i* ← [*0* ..< *length a*], *j* ← [*0* ..< *length b*]]

**definition** *big-e* :: *nat list* ⇒ *nat* ⇒ *nat list*
**where**
*big-e x j = map* (λ*i*. *eij a b i j* − *1*) (*filter* (λ*i*. *x* ! *i* ≥ *dij a b i j*) [*0* ..< *length x*])

**definition** *big-d* :: *nat list* ⇒ *nat* ⇒ *nat list*
**where**
*big-d y i = map* (λ*j*. *dij a b i j* − *1*) (*filter* (λ*j*. *y* ! *j* ≥ *eij a b i j*) [*0* ..< *length y*])

**definition** *big-d′* :: *nat list* ⇒ *nat* ⇒ *nat list*

**where**
*big-d′ y i* =
(*let l* = *length y*; *n* = *length b in*
*if l* > *n then* [] *else*
(*let k* = *n* − *l in*
*map* (λ*j*. *dij a b i* (*j* + *k*) − *1*) (*filter* (λ*j*. *y* ! *j* ≥ *eij a b i* (*j* + *k*)) [*0* ..< *length y*]))))

**definition** *max-y-impl* :: *nat list* ⇒ *nat* ⇒ *nat*
**where**
*max-y-impl x j* =
(*if j* < *length b* ∧ *big-e x j* ≠ [] *then Min* (*set* (*big-e x j*))
*else Max* (*set a*))

**definition** *max-x-impl* :: *nat list* ⇒ *nat* ⇒ *nat*
**where**
*max-x-impl y i* =
(*if i* < *length a* ∧ *big-d y i* ≠ [] *then Min* (*set* (*big-d y i*))
*else Max* (*set b*))

**definition** *max-x-impl′* :: *nat list* ⇒ *nat* ⇒ *nat*
**where**
*max-x-impl′ y i* =
(*if i* < *length a* ∧ *big-d′ y i* ≠ [] *then Min* (*set* (*big-d′ y i*))
*else Max* (*set b*))

**definition** *cond-a* :: *nat list* ⇒ *nat list* ⇒ *bool*
**where**
*cond-a xs ys* ⟷ (∀ *x*∈*set xs*. *x* ≤ *maxne0 ys b*)

**definition** *cond-b* :: *nat list* ⇒ *bool*
**where**
*cond-b xs* ⟷ (∀ *k*≤*length a*.
*take k a* · *take k xs* ≤ *b* · (*map* (*max-y-impl* (*take k xs*)) [*0* ..< *length b*]))

**definition** *boundr-impl* :: *nat list* ⇒ *nat list* ⇒ *bool*
**where**
*boundr-impl x y* ⟷ (∀ *j*<*length b*. *y* ! *j* ≤ *max-y-impl x j*)

**definition** *cond-d* :: *nat list* ⇒ *nat list* ⇒ *bool*
**where**
*cond-d xs ys* ⟷ (∀ *l*≤*length b*. *take l b* · *take l ys* ≤ *a* · *xs*)

**definition** *subdprodr-impl* :: *nat list* ⇒ *bool*
**where**
*subdprodr-impl ys* ⟷ (∀ *l*≤*length b*.
*take l b* · *take l ys* ≤ *a* · *map* (*max-x-impl* (*take l ys*)) [*0* ..< *length a*])

**definition** *subdprodl-impl* :: *nat list* ⇒ *nat list* ⇒ *bool*

**where**
*subdprodl-impl x y* $\longleftrightarrow$ $(\forall\, k{\le}$*length a. take k a* $\cdot$ *take k x* $\le$ *b* $\cdot$ *y*$)$

**definition** *boundl-impl x y* $\longleftrightarrow$ $(\forall\, i{<}$*length a. x* ! *i* $\le$ *max-x-impl y i*$)$

**definition** *static-bounds*
**where**
*static-bounds x y* $\longleftrightarrow$
(*let mx* = *maxne0-impl y b*; *my* = *maxne0-impl x a in*
$(\forall\, x{\in}$*set x. x* $\le$ *mx*$)$ $\wedge$ $(\forall\, y{\in}$*set y. y* $\le$ *my*$)$)

**definition** *check-cond* =
$(\lambda(x,\, y).$ *static-bounds x y* $\wedge$ *a* $\cdot$ *x* = *b* $\cdot$ *y* $\wedge$ *boundr-impl x y* $\wedge$ *subdprodl-impl x y* $\wedge$ *subdprodr-impl y*$)$

**definition** *check′* = *filter check-cond*

**definition** *non-special-solutions* =
(*let A* = *Max* (*set b*); *B* = *Max* (*set a*)
*in minimize* (*check′* (*generate′ A B a b*)))

**definition** *solve* = *special-solutions* @ *non-special-solutions*

**end**

**lemma** *sorted-wrt-check-generate′*:
*sorted-wrt* $(<_{rlex2})$ (*check′ a b* (*generate′ A B a b*))
⟨*proof*⟩

**lemma** *big-e*:
*set* (*big-e a b xs j*) = *hlde-ops.Ej a b j xs*
⟨*proof*⟩

**lemma** *big-d*:
*set* (*big-d a b ys i*) = *hlde-ops.Di a b i ys*
⟨*proof*⟩

**lemma** *big-d′*:
*length ys* $\le$ *length b* $\Longrightarrow$ *set* (*big-d′ a b ys i*) = *hlde-ops.Di′ a b i ys*
⟨*proof*⟩

**lemma** *max-y-impl*:
*max-y-impl a b x j* = *hlde-ops.max-y a b x j*
⟨*proof*⟩

**lemma** *max-x-impl*:
*max-x-impl a b y i* = *hlde-ops.max-x a b y i*
⟨*proof*⟩

**lemma** *max-x-impl'*:
**assumes** *length y* ≤ *length b*
**shows** *max-x-impl' a b y i* = *hlde-ops.max-x' a b y i*
⟨*proof*⟩

**lemma** (**in** *hlde*) *cond-a* [*simp*]: *cond-a b x y* = *cond-A x y*
⟨*proof*⟩

**lemma** (**in** *hlde*) *cond-b* [*simp*]: *cond-b a b x* = *cond-B x*
⟨*proof*⟩

**lemma** (**in** *hlde*) *boundr-impl* [*simp*]: *boundr-impl a b x y* = *boundr x y*
⟨*proof*⟩

**lemma** (**in** *hlde*) *cond-d* [*simp*]: *cond-d a b x y* = *cond-D x y*
⟨*proof*⟩

**lemma** (**in** *hlde*) *subdprodr-impl* [*simp*]: *subdprodr-impl a b y* = *subdprodr y*
⟨*proof*⟩

**lemma** (**in** *hlde*) *subdprodl-impl* [*simp*]: *subdprodl-impl a b x y* = *subdprodl x y*
⟨*proof*⟩

**lemma** (**in** *hlde*) *cond-bound-impl* [*simp*]: *boundl-impl a b x y* = *boundl x y*
⟨*proof*⟩

**lemma** (**in** *hlde*) *check* [*simp*]:
*check' a b* =
*filter* (λ(*x*, *y*). *static-bounds a b x y* ∧ *a* · *x* = *b* · *y* ∧ *boundr x y* ∧
*subdprodl x y* ∧
*subdprodr y*)
⟨*proof*⟩

conditions B, C, and D from Huet as well as "subdprodr" and "subdprodl" are preserved by smaller solutions

**lemma** (**in** *hlde*) *le-imp-conds*:
**assumes** *le*: *u* ≤$_v$ *x v* ≤$_v$ *y*
**and** *len*: *length x* = *m length y* = *n*
**shows** *cond-B x* ⟹ *cond-B u*
**and** *boundr x y* ⟹ *boundr u v*
**and** *a* · *u* = *b* · *v* ⟹ *cond-D x y* ⟹ *cond-D u v*
**and** *a* · *u* = *b* · *v* ⟹ *subdprodl x y* ⟹ *subdprodl u v*
**and** *subdprodr y* ⟹ *subdprodr v*
⟨*proof*⟩

**lemma** (**in** *hlde*) *special-solutions* [*simp*]:
**shows** *set* (*special-solutions a b*) = *Special-Solutions*
⟨*proof*⟩

**lemma** *set-gen2*:
*set* (*gen2 A B a b*) = {(*x*, *y*). *x* ≤$_v$ *replicate* (*length a*) *A* ∧ *y* ≤$_v$ *replicate* (*length b*) *B*}
(**is** *?L* = *?R*)
⟨*proof*⟩

**lemma** *set-gen2′*:
(λ(*x*, *y*). (*fst x*, *fst y*)) ‘ *set* (*alls2 A B a b*) =
{(*x*, *y*). *x* ≤$_v$ *replicate* (*length a*) *A* ∧ *y* ≤$_v$ *replicate* (*length b*) *B*}
⟨*proof*⟩

**lemma** (**in** *hlde*) *in-non-special-solutions*:
**assumes** (*x*, *y*) ∈ *set* (*non-special-solutions a b*)
**shows** (*x*, *y*) ∈ *Solutions*
⟨*proof*⟩

**lemma** *generate-unique*:
**assumes** *i* < *j*
**and** *j* < *length* (*generate A B a b*)
**shows** *generate A B a b* ! *i* ≠ *generate A B a b* ! *j*
⟨*proof*⟩

**lemma** *gen2-unique*:
**assumes** *i* < *j*
**and** *j* < *length* (*gen2 A B a b*)
**shows** *gen2 A B a b* ! *i* ≠ *gen2 A B a b* ! *j*
⟨*proof*⟩

**lemma** *zeroes-ni-generate′*:
(*zeroes* (*length a*), *zeroes* (*length b*)) ∉ *set* (*generate′ A B a b*)
⟨*proof*⟩

**lemma** *set-generate′*:
*set* (*generate′ A B a b*) =
{(*x*, *y*). (*x*, *y*) ≠ (*zeroes* (*length a*), *zeroes* (*length b*)) ∧ (*x*, *y*) ∈ *set* (*gen2 A B a b*)}
⟨*proof*⟩

**lemma** *set-generate′′*:
*set* (*generate′ A B a b*) =
{(*x*, *y*). (*x*, *y*) ≠ (*zeroes* (*length a*), *zeroes* (*length b*)) ∧ *x* ≤$_v$ *replicate* (*length a*) *A* ∧ *y* ≤$_v$ *replicate* (*length b*) *B*}
⟨*proof*⟩

**lemma** (**in** *hlde*) *zeroes-ni-non-special-solutions*:
**shows** (*zeroes m*, *zeroes n*) ∉ *set* (*non-special-solutions a b*)
⟨*proof*⟩

### 4.1.1 Correctness: *solve* generates only minimal solutions.

**lemma** (**in** *hlde*) *solve-subset-Minimal-Solutions*:
**shows** *set* (*solve a b*) ⊆ *Minimal-Solutions*
⟨*proof*⟩

### 4.1.2 Completeness: every minimal solution is generated by *solve*

**lemma** (**in** *hlde*) *Minimal-Solutions-subset-solve*:
**shows** *Minimal-Solutions* ⊆ *set* (*solve a b*)
⟨*proof*⟩

The main correctness and completeness result of our algorithm.

**lemma** (**in** *hlde*) *solve* [*simp*]:
**shows** *set* (*solve a b*) = *Minimal-Solutions*
⟨*proof*⟩

# 5 Making the Algorithm More Efficient

**locale** *bounded-gen-check* =
**fixes** *C* :: *nat list* ⇒ *nat* ⇒ *bool*
**and** *B* :: *nat*
**assumes** *bound*: ⋀*x xs s*. *x* > *B* ⟹ *C* (*x* # *xs*) *s* = *False*
**and** *cond-antimono*: ⋀*x x' xs s s'*. *C* (*x* # *xs*) *s* ⟹ *x'* ≤ *x* ⟹ *s'* ≤ *s* ⟹ *C* (*x'* # *xs*) *s'*
**begin**

**function** *incs* :: *nat* ⇒ *nat* ⇒ (*nat list* × *nat*) ⇒ (*nat list* × *nat*) *list*
**where**
*incs a x* (*xs*, *s*) =
(*let t* = *s* + *a* ∗ *x in*
*if C* (*x* # *xs*) *t then* (*x* # *xs*, *t*) # *incs a* (*Suc x*) (*xs*, *s*) *else* [])
⟨*proof*⟩
**termination**
⟨*proof*⟩
**declare** *incs.simps* [*simp del*]

**lemma** *in-incs*:
**assumes** (*ys*, *t*) ∈ *set* (*incs a x* (*xs*, *s*))
**shows** *length ys* = *length xs* + *1* ∧ *t* = *s* + *hd ys* ∗ *a* ∧ *tl ys* = *xs* ∧ *C ys t*
⟨*proof*⟩

**lemma** *incs-Nil* [*simp*]: *x* > *B* ⟹ *incs a x* (*xs*, *s*) = []
⟨*proof*⟩

**lemma** *incs-filter*:
**assumes** *x* ≤ *B*
**shows** *incs a x* = (λ(*xs*, *s*). *filter* (*cond-cons C*) (*map* (λ*x*. (*x* # *xs*, *s* + *a* ∗ *x*)) [*x* ..< *B* + *1*]))

⟨*proof*⟩

**fun** *gen-check* :: *nat list* ⇒ (*nat list* × *nat*) *list*
**where**
*gen-check* [] = [([], 0)]
| *gen-check* (*a* # *as*) = *concat* (*map* (*incs a* 0) (*gen-check as*))

**lemma** *gen-check-len*:
**assumes** (*ys*, *s*) ∈ *set* (*gen-check as*)
**shows** *length ys* = *length as*
⟨*proof*⟩

**lemma** *in-gen-check*:
**assumes** (*xs*, *s*) ∈ *set* (*gen-check as*)
**shows** *length xs* = *length as* ∧ *s* = *as* · *xs*
⟨*proof*⟩

**lemma** *gen-check-filter*:
*gen-check as* = *filter* (*suffs C as*) (*alls B as*)
⟨*proof*⟩

**lemma** *in-gen-check-cond*:
**assumes** (*xs*, *s*) ∈ *set* (*gen-check as*)
**shows** ∀*j*≤*length xs*. *drop j xs* ≠ [] ⟶ *C* (*drop j xs*) (*s* − *take j as* · *take j xs*)
⟨*proof*⟩

**lemma** *sorted-gen-check*:
*sorted-wrt* (<$_{rlex}$) (*map fst* (*gen-check xs*))
⟨*proof*⟩

**end**

**locale** *bounded-generate-check* =
*c2*: *bounded-gen-check* $C_2$ $B_2$ **for** $C_2$ $B_2$ +
**fixes** $C_1$ **and** $B_1$
**assumes** *cond1*: ⋀*b ys*. *ys* ∈ *fst* ‘ *set* (*c2.gen-check b*) ⟹ *bounded-gen-check* ($C_1$ *b ys*) ($B_1$ *b*)
**begin**

**definition** *generate-check a b* =
[(*xs*, *ys*). *ys* ← *c2.gen-check b*, *xs* ← *bounded-gen-check.gen-check* ($C_1$ *b* (*fst ys*)) *a*]

**lemma** *generate-check-filter-conv*:
*generate-check a b* = [(*xs*, *ys*).
*ys* ← *filter* (*suffs* $C_2$ *b*) (*alls* $B_2$ *b*),
*xs* ← *filter* (*suffs* ($C_1$ *b* (*fst ys*)) *a*) (*alls* ($B_1$ *b*) *a*)]
⟨*proof*⟩

**lemma** *generate-check-filter*:
*generate-check a b* = [(*xs*, *ys*) ← *alls2* (*B*$_1$ *b*) *B*$_2$ *a b*. *suffs* (*C*$_1$ *b* (*fst ys*)) *a xs* ∧ *suffs C*$_2$ *b ys*]
⟨*proof*⟩

**lemma** *tl-generate-check-filter*:
**assumes** *suffs* (*C*$_1$ *b* (*zeroes* (*length b*))) *a* (*zeroes* (*length a*), *0*)
**and** *suffs C*$_2$ *b* (*zeroes* (*length b*), *0*)
**shows** *tl* (*generate-check a b*) = [(*xs*, *ys*) ← *tl* (*alls2* (*B*$_1$ *b*) *B*$_2$ *a b*). *suffs* (*C*$_1$ *b* (*fst ys*)) *a xs* ∧ *suffs C*$_2$ *b ys*]
⟨*proof*⟩

**end**

**context**
**fixes** *a b* :: *nat list*
**begin**

**fun** *cond1*
**where**
*cond1 ys* [] *s* ⟷ *True*
| *cond1 ys* (*x* # *xs*) *s* ⟷ *s* ≤ *b* ⋅ *ys* ∧ *x* ≤ *maxne0-impl ys b*

**lemma** *max-x-impl'-conv*:
*i* < *length a* ⟹ *length y* = *length b* ⟹ *max-x-impl' a b y i* = *max-x-impl a b y i*
⟨*proof*⟩

**fun** *cond2*
**where**
*cond2* [] *s* ⟷ *True*
| *cond2* (*y* # *ys*) *s* ⟷ *y* ≤ *Max* (*set a*) ∧ *s* ≤ *a* ⋅ *map* (*max-x-impl' a b* (*y* # *ys*)) [*0* ..< *length a*]

**lemma** *le-imp-big-d'-subset*:
**assumes** *v* ≤$_v$ *y*
**shows** *set* (*big-d' a b v i*) ⊆ *set* (*big-d' a b y i*)
⟨*proof*⟩

**lemma** *finite-big-d'*:
*finite* (*set* (*big-d' a b y i*))
⟨*proof*⟩

**lemma** *Min-big-d'-le*:
**assumes** *i* < *length a*
**and** *big-d' a b y i* ≠ []
**and** *length y* ≤ *length b*
**shows** *Min* (*set* (*big-d' a b y i*)) ≤ *Max* (*set b*) (**is** *?m* ≤ -)
⟨*proof*⟩

**lemma** *le-imp-max-x-impl'-ge*:
**assumes** $v \leq_v y$
**and** $i < length\ a$
**shows** $max\text{-}x\text{-}impl'\ a\ b\ v\ i \geq max\text{-}x\text{-}impl'\ a\ b\ y\ i$
⟨*proof*⟩

**end**

**global-interpretation** *c12*: *bounded-generate-check* (*cond2 a b*) *Max* (*set a*) *cond1* λ*b*. *Max* (*set b*)
**defines** *c2-gen-check* = *c12.c2.gen-check* **and** *c2-incs* = *c12.c2.incs*
**and** *c12-generate-check* = *c12.generate-check*
⟨*proof*⟩

**definition** *post-cond a b* = (λ(*x*, *y*). *static-bounds a b x y* ∧ *a* · *x* = *b* · *y* ∧ *boundr-impl a b x y*)

**definition** *fast-filter a b* =
*filter* (*post-cond a b*) (*map* (λ(*x*, *y*). (*fst x*, *fst y*)) (*tl* (*c12-generate-check a b a b*)))

**lemma** *cond1-cond2-zeroes*:
**shows** *suffs* (*cond1 b* (*zeroes* (*length b*))) *a* (*zeroes* (*length a*), *0*)
**and** *suffs* (*cond2 a b*) *b* (*zeroes* (*length b*), *0*)
⟨*proof*⟩

**lemma** *suffs-cond1I*:
**assumes** ∀ *y*∈*set aa*. *y* ≤ *maxne0-impl aaa b*
**and** *length aa* = *length a*
**and** *a* · *aa* = *b* · *aaa*
**shows** *suffs* (*cond1 b aaa*) *a* (*aa*, *b* · *aaa*)
⟨*proof*⟩

**lemma** *suffs-cond2-conv*:
**assumes** *length ys* = *length b*
**shows** *suffs* (*cond2 a b*) *b* (*ys*, *b* · *ys*) ⟷
(∀ *y*∈*set ys*. *y* ≤ *Max* (*set a*)) ∧ *subdprodr-impl a b ys*
(**is** *?L* ⟷ *?R*)
⟨*proof*⟩

**lemma** *suffs-cond2I*:
**assumes** ∀ *y*∈*set aaa*. *y* ≤ *Max* (*set a*)
**and** *length aaa* = *length b*
**and** *subdprodr-impl a b aaa*
**shows** *suffs* (*cond2 a b*) *b* (*aaa*, *b* · *aaa*)
⟨*proof*⟩

**lemma** *check-cond-conv*:

**assumes** $(x, y) \in set\ (alls2\ (Max\ (set\ b))\ (Max\ (set\ a))\ a\ b)$
**shows** *check-cond a b* (*fst x*, *fst y*) $\longleftrightarrow$
*static-bounds a b* (*fst x*) (*fst y*) $\wedge$ $a \cdot fst\ x = b \cdot fst\ y \wedge$ *boundr-impl a b* (*fst x*) (*fst y*) $\wedge$
*suffs* (*cond1 b* (*fst y*)) *a x* $\wedge$
*suffs* (*cond2 a b*) *b y*
⟨*proof*⟩

**lemma** *tune*:
*check′ a b* (*generate′* (*Max* (*set b*)) (*Max* (*set a*)) *a b*) = *fast-filter a b*
⟨*proof*⟩

**locale** *bounded-incs* =
**fixes** *cond* :: *nat list* ⇒ *nat* ⇒ *bool*
**and** *B* :: *nat*
**assumes** *bound*: $\bigwedge x\ xs\ s.\ x > B \Longrightarrow cond\ (x \#\ xs)\ s = False$
**begin**

**function** *incs* :: *nat* ⇒ *nat* ⇒ (*nat list* × *nat*) ⇒ (*nat list* × *nat*) *list*
**where**
*incs a x* (*xs*, *s*) =
(*let t* = *s* + *a* ∗ *x in*
*if cond* (*x* # *xs*) *t then* (*x* # *xs*, *t*) # *incs a* (*Suc x*) (*xs*, *s*) *else* [])
⟨*proof*⟩
**termination**
⟨*proof*⟩
**declare** *incs.simps* [*simp del*]

**lemma** *in-incs*:
**assumes** (*ys*, *t*) ∈ *set* (*incs a x* (*xs*, *s*))
**shows** *length ys* = *length xs* + 1 $\wedge$ *t* = *s* + *hd ys* ∗ *a* $\wedge$ *tl ys* = *xs* $\wedge$ *cond ys t*
⟨*proof*⟩

**lemma** *incs-Nil* [*simp*]: $x > B \Longrightarrow incs\ a\ x\ (xs, s) = []$
⟨*proof*⟩

**end**

**global-interpretation** *incs1*:
*bounded-incs* (*cond1 b ys*) (*Max* (*set b*))
**for** *b ys* :: *nat list*
**defines** *c1-incs* = *incs1.incs*
⟨*proof*⟩

**fun** *c1-gen-check*
**where**
*c1-gen-check b ys* [] = [([], 0)]
| *c1-gen-check b ys* (*a* # *as*) = *concat* (*map* (*c1-incs b ys a* 0) (*c1-gen-check b ys as*))

**definition** *generate-check a b* = [(*xs*, *ys*). *ys* ← *c2-gen-check a b b*, *xs* ← *c1-gen-check b* (*fst ys*) *a*]

**lemma** *c1-gen-check-conv*:
**assumes** (*ys*, *s*) ∈ *set* (*c2-gen-check a b b*)
**shows** *c1-gen-check b ys a* = *bounded-gen-check.gen-check* (*cond1 b ys*) *a*
⟨*proof*⟩

### 5.1 Code Generation

**lemma** *solve-efficient* [*code*]:
*solve a b* = *special-solutions a b* @ *minimize* (*fast-filter a b*)
⟨*proof*⟩

**lemma** *c12-generate-check-code* [*code-unfold*]:
*c12-generate-check a b a b* = *generate-check a b*
⟨*proof*⟩

**end**

## References


[1] G. Huet. An algorithm to generate the basis of solutions to homogeneous linear diophantine equations. *Information Processing Letters*, 7(3):144–147, 1978.