

Diophantine Equations*

Florian MeSSner Julian Parsert Jonas Schöpf
Christian Sternagel

March 17, 2025

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

1	Vectors as Lists of Naturals	2
1.1	The Inner Product	3
1.2	The Pointwise Order on Vectors	5
1.3	Pointwise Subtraction	9
1.4	The Lexicographic Order on Vectors	10
1.5	Code Equations	11
2	Homogeneous Linear Diophantine Equations	12
2.1	Further Constraints on Minimal Solutions	13
2.2	Pointwise Restricting Solutions	17
2.3	Special Solutions	19
2.4	Huet’s conditions	20
2.5	New conditions: facilitating generation of candidates from right to left	20
3	Minimization	23
3.1	Reverse-Lexicographic Enumeration of Potential Minimal So- lutions	25
3.1.1	Completeness: every minimal solution is generated by <i>solutions</i>	27
3.1.2	Correctness: <i>solutions</i> generates only minimal solutions.	27

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

4	Computing Minimal Complete Sets of Solutions	27
4.1	The Algorithm	30
4.1.1	Correctness: <i>solve</i> generates only minimal solutions. . .	34
4.1.2	Completeness: every minimal solution is generated by <i>solve</i>	34
5	Making the Algorithm More Efficient	35
5.1	Code Generation	40

1 Vectors as Lists of Naturals

```
theory List-Vector
  imports Main
begin
```

```
lemma lex-lengthD:  $(x, y) \in \text{lex } P \implies \text{length } x = \text{length } y$ 
  <proof>
```

```
lemma lex-take-index:
  assumes  $(xs, ys) \in \text{lex } r$ 
  obtains  $i$  where  $\text{length } ys = \text{length } xs$ 
    and  $i < \text{length } xs$  and  $\text{take } i \text{ } xs = \text{take } i \text{ } ys$ 
    and  $(xs ! i, ys ! i) \in r$ 
  <proof>
```

```
lemma mods-with-nats:
  assumes  $(v::\text{nat}) > w$ 
  and  $(v * b) \bmod a = (w * b) \bmod a$ 
  shows  $((v - w) * b) \bmod a = 0$ 
  <proof>
```

```
abbreviation zeroes ::  $\text{nat} \Rightarrow \text{nat list}$ 
  where
    zeroes  $n \equiv \text{replicate } n \ 0$ 
```

```
lemma rep-upd-unit:
  assumes  $x = (\text{zeroes } n)[i := a]$ 
  shows  $\forall j < \text{length } x. (j \neq i \longrightarrow x ! j = 0) \wedge (j = i \longrightarrow x ! j = a)$ 
  <proof>
```

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

```
lemma nonzero-append [simp]:
   $\text{nonzero } (xs @ ys) \longleftrightarrow \text{nonzero } xs \vee \text{nonzero } ys$  <proof>
```

1.1 The Inner Product

definition *dotprod* :: *nat list* \Rightarrow *nat list* \Rightarrow *nat* (**infixl** $\langle \cdot \rangle$ 70)

where

$$xs \cdot ys = (\sum_{i < \min(\text{length } xs) (\text{length } ys)}. xs ! i * ys ! i)$$

lemma *dotprod-code* [*code*]:

$$xs \cdot ys = \text{sum-list } (\text{map } (\lambda(x, y). x * y) (\text{zip } xs \text{ } ys))$$

$\langle \text{proof} \rangle$

lemma *dotprod-commute*:

assumes *length xs = length ys*

shows $xs \cdot ys = ys \cdot xs$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *dotprod-Cons* [*simp*]:

$$(x \# xs) \cdot (y \# ys) = x * y + xs \cdot ys$$

$\langle \text{proof} \rangle$

lemma *dotprod-1-right* [*simp*]:

$$xs \cdot \text{replicate } (\text{length } xs) \ 1 = \text{sum-list } xs$$

$\langle \text{proof} \rangle$

lemma *dotprod-0-right* [*simp*]:

$$xs \cdot \text{zeroes } (\text{length } xs) = 0$$

$\langle \text{proof} \rangle$

lemma *dotprod-unit* [*simp*]:

assumes *length a = n*

and $k < n$

shows $a \cdot (\text{zeroes } n)[k := zk] = a ! k * zk$

$\langle \text{proof} \rangle$

lemma *dotprod-gt0*:

assumes *length x = length y* **and** $\exists i < \text{length } y. x ! i > 0 \wedge y ! i > 0$

shows $x \cdot y > 0$

$\langle \text{proof} \rangle$

lemma *dotprod-gt0D*:

assumes *length x = length y*

and $x \cdot y > 0$

shows $\exists i < \text{length } y. x ! i > 0 \wedge y ! i > 0$

$\langle \text{proof} \rangle$

lemma *dotprod-gt0-iff* [*iff*]:

assumes *length x = length y*

shows $x \cdot y > 0 \iff (\exists i < \text{length } y. x ! i > 0 \wedge y ! i > 0)$

<proof>

lemma *dotprod-append*:

assumes *length a = length b*

shows $(a @ x) \cdot (b @ y) = a \cdot b + x \cdot y$

<proof>

lemma *dotprod-le-take*:

assumes *length a = length b*

and $k \leq \text{length } a$

shows $\text{take } k \ a \cdot \text{take } k \ b \leq a \cdot b$

<proof>

lemma *dotprod-le-drop*:

assumes *length a = length b*

and $k \leq \text{length } a$

shows $\text{drop } k \ a \cdot \text{drop } k \ b \leq a \cdot b$

<proof>

lemma *dotprod-is-0* [*simp*]:

assumes *length x = length y*

shows $x \cdot y = 0 \iff (\forall i < \text{length } y. x ! i = 0 \vee y ! i = 0)$

<proof>

lemma *dotprod-eq-0-iff*:

assumes *length x = length a*

and $0 \notin \text{set } a$

shows $x \cdot a = 0 \iff (\forall e \in \text{set } x. e = 0)$

<proof>

lemma *dotprod-eq-nonzero-iff*:

assumes $a \cdot x = b \cdot y$ **and** *length x = length a* **and** *length y = length b*

and $0 \notin \text{set } a$ **and** $0 \notin \text{set } b$

shows $\text{nonzero } x \iff \text{nonzero } y$

<proof>

lemma *eq-0-iff*:

$xs = \text{zeroes } n \iff \text{length } xs = n \wedge (\forall x \in \text{set } xs. x = 0)$

<proof>

lemma *not-nonzero-iff*: $\neg \text{nonzero } x \iff x = \text{zeroes } (\text{length } x)$

<proof>

lemma *neq-0-iff'*:

$xs \neq \text{zeroes } n \iff \text{length } xs \neq n \vee (\exists x \in \text{set } xs. x > 0)$

<proof>

lemma *dotprod-pointwise-le*:

assumes *length as = length xs*

and $i < \text{length } as$
shows $as ! i * xs ! i \leq as \cdot xs$
 ⟨proof⟩

lemma *replicate-dotprod*:
assumes $\text{length } y = n$
shows $\text{replicate } n \ x \cdot y = x * \text{sum-list } y$
 ⟨proof⟩

1.2 The Pointwise Order on Vectors

definition *less-eq* :: $\text{nat list} \Rightarrow \text{nat list} \Rightarrow \text{bool}$ ($\langle - / \leq_v \rightarrow [51, 51] 50$)
where
 $xs \leq_v ys \iff \text{length } xs = \text{length } ys \wedge (\forall i < \text{length } xs. xs ! i \leq ys ! i)$

definition *less* :: $\text{nat list} \Rightarrow \text{nat list} \Rightarrow \text{bool}$ ($\langle - / <_v \rightarrow [51, 51] 50$)
where
 $xs <_v ys \iff xs \leq_v ys \wedge \neg ys \leq_v xs$

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

lemma *less-eqI* [*intro?*]: $\text{length } xs = \text{length } ys \implies \forall i < \text{length } xs. xs ! i \leq ys ! i$
 $\implies xs \leq_v ys$
 ⟨proof⟩

lemma *le0* [*simp, intro*]: $\text{zeroes } (\text{length } xs) \leq_v xs$ ⟨proof⟩

lemma *le-list-update* [*simp*]:
assumes $xs \leq_v ys$ **and** $i < \text{length } ys$ **and** $z \leq ys ! i$
shows $xs[i := z] \leq_v ys$
 ⟨proof⟩

lemma *le-Cons*: $x \# xs \leq_v y \# ys \iff x \leq y \wedge xs \leq_v ys$
 ⟨proof⟩

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

lemma *le-append*:
assumes $\text{length } xs = \text{length } vs$
shows $xs @ ys \leq_v vs @ ws \iff xs \leq_v vs \wedge ys \leq_v ws$
 ⟨proof⟩

lemma *less-Cons*:
 $(x \# xs) <_v (y \# ys) \iff \text{length } xs = \text{length } ys \wedge (x \leq y \wedge xs <_v ys \vee x < y \wedge xs \leq_v ys)$

$\langle proof \rangle$

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

lemma *less-length* [*dest*]:
 assumes $x <_v y$
 shows $length\ x = length\ y$
 $\langle proof \rangle$

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

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

lemma *less-append-cases*:
 assumes $xs @ ys <_v vs @ ws$ **and** $length\ xs = length\ vs$
 obtains $xs <_v vs$ **and** $ys \leq_v ws \mid xs \leq_v vs$ **and** $ys <_v ws$
 $\langle proof \rangle$

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

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

lemma *dotprod-le-right*:
 assumes $v \leq_v w$
 and $length\ b = length\ w$
 shows $b \cdot v \leq b \cdot w$
 $\langle proof \rangle$

lemma *dotprod-pointwise-le-right*:
 assumes $length\ z = length\ u$
 and $length\ u = length\ v$

and $\forall i < \text{length } v. u ! i \leq v ! i$
shows $z \cdot u \leq z \cdot v$
 ⟨proof⟩

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

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

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

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

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

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

lemma *le-sum-list-mono*:
assumes $xs \leq_v ys$
shows $\text{sum-list } xs \leq \text{sum-list } ys$
 ⟨proof⟩

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

<proof>

lemma *less-vec-sum-list-less*:

assumes $v <_v w$

shows $\text{sum-list } v < \text{sum-list } w$

<proof>

definition $\text{maxne0} :: \text{nat list} \Rightarrow \text{nat list} \Rightarrow \text{nat}$

where

$\text{maxne0 } x \ a =$

(if $\text{length } x = \text{length } a \wedge (\exists i < \text{length } a. x ! i \neq 0)$

then $\text{Max } \{a ! i \mid i. i < \text{length } a \wedge x ! i \neq 0\}$

else 0)

lemma *maxne0-le-Max*:

$\text{maxne0 } x \ a \leq \text{Max } (\text{set } a)$

<proof>

lemma *maxne0-Nil* [simp]:

$\text{maxne0 } [] \ as = 0$

$\text{maxne0 } xs \ [] = 0$

<proof>

lemma *maxne0-Cons* [simp]:

$\text{maxne0 } (x \# xs) \ (a \# as) =$

(if $\text{length } xs = \text{length } as$ then

(if $x = 0$ then $\text{maxne0 } xs \ as$ else $\text{max } a \ (\text{maxne0 } xs \ as)$)

else 0)

<proof>

lemma *maxne0-times-sum-list-gt-dotprod*:

assumes $\text{length } b = \text{length } ys$

shows $\text{maxne0 } ys \ b * \text{sum-list } ys \geq b \cdot ys$

<proof>

lemma *max-times-sum-list-gt-dotprod*:

assumes $\text{length } b = \text{length } ys$

shows $\text{Max } (\text{set } b) * \text{sum-list } ys \geq b \cdot ys$

<proof>

lemma *maxne0-mono*:

assumes $y \leq_v x$

shows $\text{maxne0 } y \ a \leq \text{maxne0 } x \ a$

<proof>

lemma *all-leq-Max*:

assumes $x \leq_v y$

and $x \neq []$

shows $\forall xi \in \text{set } x. xi \leq \text{Max } (\text{set } y)$

<proof>

lemma *le-not-less-replicate*:

$\forall x \in \text{set } xs. x \leq b \implies \neg xs <_v \text{replicate } (\text{length } xs) b \implies xs = \text{replicate } (\text{length } xs) b$

<proof>

lemma *le-replicateI*: $\forall x \in \text{set } xs. x \leq b \implies xs \leq_v \text{replicate } (\text{length } xs) b$

<proof>

lemma *le-take*:

assumes $x \leq_v y$ **and** $i \leq \text{length } x$ **shows** $\text{take } i x \leq_v \text{take } i y$

<proof>

lemma *wf-less*:

wf $\{(x, y). x <_v y\}$

<proof>

1.3 Pointwise Subtraction

definition *vdiff* :: $\text{nat list} \Rightarrow \text{nat list} \Rightarrow \text{nat list}$ (**infixl** $\langle -_v \rangle$ 65)

where

$w -_v v = \text{map } (\lambda i. w ! i - v ! i) [0 .. < \text{length } w]$

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*: $\text{map } f [Suc\ m .. < Suc\ n] = \text{map } (f \circ Suc) [m .. < n]$

<proof>

lemma *vdiff-Cons* [*simp*]:

$(x \# xs) -_v (y \# ys) = (x - y) \# (xs -_v ys)$

<proof>

lemma *vdiff-alt-def*:

assumes $\text{length } w = \text{length } v$

shows $w -_v v = \text{map } (\lambda(x, y). x - y) (\text{zip } w v)$

<proof>

lemma *vdiff-dotprod-distr*:

assumes $\text{length } b = \text{length } w$

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 $\text{sum-list } (u -_v v) = \text{sum-list } u - \text{sum-list } v$
 $\langle \text{proof} \rangle$

lemma *vdiff-le*:
assumes $v \leq_v w$
and $\text{length } v = \text{length } x$
shows $v -_v x \leq_v w$
 $\langle \text{proof} \rangle$

lemma *mods-with-vec*:
assumes $v <_v w$
and $0 \notin \text{set } b$
and $\text{length } b = \text{length } w$
and $(v \cdot b) \bmod a = (w \cdot b) \bmod a$
shows $((w -_v v) \cdot b) \bmod a = 0$
 $\langle \text{proof} \rangle$

lemma *mods-with-vec-2*:
assumes $v <_v w$
and $0 \notin \text{set } b$
and $\text{length } b = \text{length } w$
and $(b \cdot v) \bmod a = (b \cdot w) \bmod a$
shows $(b \cdot (w -_v v)) \bmod a = 0$
 $\langle \text{proof} \rangle$

1.4 The Lexicographic Order on Vectors

abbreviation *lex-less-than* ($\langle _ / _ <_{lex} _ \rangle$ [*51, 51*] *50*)

where

$xs <_{lex} ys \equiv (xs, ys) \in \text{lex less-than}$

definition *rllex* (**infix** $\langle _ <_{rllex} _ \rangle$ *50*)

where

$xs <_{rllex} ys \longleftrightarrow \text{rev } xs <_{lex} \text{rev } ys$

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

$\text{rev } xs \leq_v \text{rev } ys \longleftrightarrow xs \leq_v ys$

$\langle \text{proof} \rangle$

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

$\text{rev } xs <_v \text{rev } ys \longleftrightarrow xs <_v ys$

$\langle \text{proof} \rangle$

lemma *less-imp-lex*:

assumes $xs <_v ys$ **shows** $xs <_{lex} ys$

$\langle \text{proof} \rangle$

lemma *less-imp-rlex*:
assumes $xs <_v ys$ **shows** $xs <_{rlex} ys$
 $\langle proof \rangle$

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

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

lemma *lex-trans*:
assumes $x <_{lex} y$ **and** $y <_{lex} z$
shows $x <_{lex} z$
 $\langle proof \rangle$

lemma *rlex-trans*:
assumes $x <_{rlex} y$ **and** $y <_{rlex} z$
shows $x <_{rlex} z$
 $\langle proof \rangle$

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

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

lemma *rlex-irrefl*:
 $\neg x <_{rlex} x$
 $\langle proof \rangle$

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*]: $\neg xs \leq_v ys \longleftrightarrow exists2\ True\ (>) xs\ ys$
 $\langle proof \rangle$

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 $(\exists x \in set\ xs. x\ mod\ n = 0) \vee$
 $(\exists i < length\ xs. \exists j < length\ xs. i \neq j \wedge (xs\ !\ i)\ mod\ n = (xs\ !\ j)\ mod\ n)$
<proof>

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

locale *hlde-ops* =
fixes $a\ b :: \text{nat list}$
begin

abbreviation $m \equiv \text{length } a$
abbreviation $n \equiv \text{length } b$

— The set of all solutions.

definition *Solutions* :: $(\text{nat list} \times \text{nat list}) \text{ set}$

where

$$\text{Solutions} = \{(x, y). a \cdot x = b \cdot y \wedge \text{length } x = m \wedge \text{length } y = n\}$$

lemma *in-Solutions-iff*:

$$(x, y) \in \text{Solutions} \longleftrightarrow \text{length } x = m \wedge \text{length } y = n \wedge a \cdot x = b \cdot y$$

<proof>

definition *Minimal-Solutions* :: $(\text{nat list} \times \text{nat list}) \text{ set}$

where

$$\text{Minimal-Solutions} = \{(x, y) \in \text{Solutions}. \text{nonzero } x \wedge \neg (\exists (u, v) \in \text{Solutions}. \text{nonzero } u \wedge u @ v <_v x @ y)\}$$

definition *dij* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$

where

$$\text{dij } i\ j = \text{lcm } (a ! i) (b ! j) \text{ div } (a ! i)$$

definition *eij* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$

where

$$\text{eij } i\ j = \text{lcm } (a ! i) (b ! j) \text{ div } (b ! j)$$

definition *sij* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat list} \times \text{nat list})$

where

$$\text{sij } i\ j = ((\text{zeroes } m)[i := \text{dij } i\ j], (\text{zeroes } n)[j := \text{eij } i\ j])$$

2.1 Further Constraints on Minimal Solutions

definition *Ej* :: $\text{nat} \Rightarrow \text{nat list} \Rightarrow \text{nat set}$

where

$$\text{Ej } j\ x = \{ \text{eij } i\ j - 1 \mid i. i < \text{length } x \wedge x ! i \geq \text{dij } i\ j \}$$

definition *Di* :: $\text{nat} \Rightarrow \text{nat list} \Rightarrow \text{nat set}$

where

$$\text{Di } i\ y = \{ \text{dij } i\ j - 1 \mid j. j < \text{length } y \wedge y ! j \geq \text{eij } i\ j \}$$

definition *Di'* :: $\text{nat} \Rightarrow \text{nat list} \Rightarrow \text{nat set}$

where

$$\text{Di}'\ i\ y = \{ \text{dij } i\ (j + \text{length } b - \text{length } y) - 1 \mid j. j < \text{length } y \wedge y ! j \geq \text{eij } i\ (j + \text{length } b - \text{length } y) \}$$

lemma *Ej-take-subset*:

$$\text{Ej } j\ (\text{take } k\ x) \subseteq \text{Ej } j\ x$$

<proof>

lemma *Di-take-subset:*

$Di\ i\ (take\ l\ y) \subseteq Di\ i\ y$

<proof>

lemma *Di'-drop-subset:*

$Di'\ i\ (drop\ l\ y) \subseteq Di'\ i\ y$

<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 $max-y :: nat\ list \Rightarrow nat \Rightarrow nat$

where

$max-y\ x\ j = (if\ j < n \wedge Ej\ j\ x \neq \{\} \ then\ Min\ (Ej\ j\ x) \ else\ Max\ (set\ a))$

definition $max-x :: nat\ list \Rightarrow nat \Rightarrow nat$

where

$max-x\ y\ i = (if\ i < m \wedge Di\ i\ y \neq \{\} \ then\ Min\ (Di\ i\ y) \ else\ Max\ (set\ b))$

definition $max-x' :: nat\ list \Rightarrow nat \Rightarrow nat$

where

$max-x'\ y\ i = (if\ i < m \wedge Di'\ i\ y \neq \{\} \ then\ Min\ (Di'\ i\ y) \ else\ Max\ (set\ b))$

lemma *Min-Ej-le:*

assumes $j < n$

and $e \in Ej\ j\ x$

and $length\ x \leq m$

shows $Min\ (Ej\ j\ x) \leq Max\ (set\ a)$ (**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 *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 $max-y\ x\ j \leq max-y\ (take\ k\ x)\ j$
 <proof>

lemma *max-x-le-take*:
assumes $length\ y \leq n$
shows $max-x\ y\ i \leq max-x\ (take\ l\ y)\ i$
 <proof>

lemma *max-x'-le-drop*:
assumes $length\ y \leq n$
shows $max-x'\ y\ i \leq max-x'\ (drop\ l\ y)\ i$
 <proof>

end

abbreviation $Solutions \equiv hlde-ops.Solutions$

abbreviation $Minimal-Solutions \equiv hlde-ops.Minimal-Solutions$

abbreviation $dij \equiv hlde-ops.dij$

abbreviation $eij \equiv hlde-ops.eij$

abbreviation $sij \equiv hlde-ops.sij$

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*: $(x, y) \in Minimal-Solutions\ a\ b \implies (x, y) \in Solutions\ a\ b$
 <proof>

lemma *Minimal-SolutionsI*:
assumes $(x, y) \in Solutions\ a\ b$
and $nonzero\ x$
and $\neg (\exists (u, v) \in Solutions\ a\ b. nonzero\ u \wedge u @ v <_v x @ y)$
shows $(x, y) \in Minimal-Solutions\ a\ b$
 <proof>

lemma *minimize-nonzero-solution*:

assumes $(x, y) \in \text{Solutions } a \text{ } b$ **and** *nonzero* x
obtains u **and** v **where** $u @ v \leq_v x @ y$ **and** $(u, v) \in \text{Minimal-Solutions } a \text{ } b$
 $\langle \text{proof} \rangle$

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

lemma *Minimal-Solutions-length*:
 $(x, y) \in \text{Minimal-Solutions } a \text{ } b \implies \text{length } x = \text{length } a \wedge \text{length } y = \text{length } b$
 $\langle \text{proof} \rangle$

lemma *Minimal-Solutions-gt0*:
 $(x, y) \in \text{Minimal-Solutions } a \text{ } b \implies \text{zeroes } (\text{length } x) <_v x$
 $\langle \text{proof} \rangle$

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

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

lemma *nonzero-Solutions-iff*:
assumes $(x, y) \in \text{Solutions}$
shows *nonzero* $x \longleftrightarrow$ *nonzero* y
 $\langle \text{proof} \rangle$

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

lemma *Solutions-snd-not-0*:
assumes $(x, y) \in \text{Solutions}$
and *nonzero* x
shows *nonzero* y
 $\langle \text{proof} \rangle$

end

2.2 Pointwise Restricting Solutions

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

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

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 $\text{inc} :: \text{nat list} \Rightarrow \text{nat} \Rightarrow \text{nat list} \Rightarrow \text{nat list}$

where

$\text{inc } y \ i \ u =$
 (if $i < \text{length } y$ then
 if $u ! i < y ! i$ then $u[i := u ! i + 1]$
 else $\text{inc } y \ (\text{Suc } i) \ u$
 else u)

$\langle \text{proof} \rangle$

termination inc

$\langle \text{proof} \rangle$

declare $\text{inc.simps} [\text{simp del}]$

Starting from the 0-vector produce us by iteratively incrementing with respect to y .

definition $\text{huets-us} :: \text{nat list} \Rightarrow \text{nat} \Rightarrow \text{nat list} \ (\langle \mathbf{u} \rangle \ 1000)$

where

$\mathbf{u} \ y \ i = ((\text{inc } y \ 0) \ \widehat{\sim} \ \text{Suc } i) \ (\text{zeroes } (\text{length } y))$

lemma $\text{huets-us-simps} [\text{simp}]$:

$\mathbf{u} \ y \ 0 = \text{inc } y \ 0 \ (\text{zeroes } (\text{length } y))$

$\mathbf{u} \ y \ (\text{Suc } i) = \text{inc } y \ 0 \ (\mathbf{u} \ y \ i)$

$\langle \text{proof} \rangle$

lemma $\text{length-inc} [\text{simp}]$: $\text{length } (\text{inc } y \ i \ u) = \text{length } u$

$\langle \text{proof} \rangle$

lemma $\text{length-us} [\text{simp}]$:

$\text{length } (\mathbf{u} \ y \ i) = \text{length } y$

$\langle \text{proof} \rangle$

inc produces vectors that are pointwise smaller than y

lemma inc-le :

assumes $\text{length } u = \text{length } y$ **and** $i < \text{length } y$ **and** $u \leq_v y$

shows $\text{inc } y \ i \ u \leq_v y$

$\langle \text{proof} \rangle$

lemma *us-le*:

assumes $\text{length } y > 0$

shows $\mathbf{u} \ y \ i \leq_v \ y$

<proof>

lemma *sum-list-inc-le*:

$u \leq_v \ y \implies \text{sum-list } (\text{inc } y \ i \ u) \leq \text{sum-list } y$

<proof>

lemma *sum-list-inc-gt0*:

assumes $\text{sum-list } u > 0$ **and** $\text{length } y = \text{length } u$

shows $\text{sum-list } (\text{inc } y \ i \ u) > 0$

<proof>

lemma *sum-list-inc-gt0'*:

assumes $\text{length } u = \text{length } y$ **and** $i < \text{length } y$ **and** $y \ ! \ i > 0$ **and** $j \leq i$

shows $\text{sum-list } (\text{inc } y \ j \ u) > 0$

<proof>

lemma *sum-list-us-gt0*:

assumes $\text{sum-list } y \neq 0$

shows $0 < \text{sum-list } (\mathbf{u} \ y \ i)$

<proof>

lemma *sum-list-inc-le'*:

assumes $\text{length } u = \text{length } y$

shows $\text{sum-list } (\text{inc } y \ i \ u) \leq \text{sum-list } u + 1$

<proof>

lemma *sum-list-us-le*:

$\text{sum-list } (\mathbf{u} \ y \ i) \leq i + 1$

<proof>

lemma *sum-list-us-bounded*:

assumes $i < k$

shows $\text{sum-list } (\mathbf{u} \ y \ i) \leq k$

<proof>

lemma *sum-list-inc-eq-sum-list-Suc*:

assumes $\text{length } u = \text{length } y$ **and** $i < \text{length } y$

and $\exists j \geq i. j < \text{length } y \wedge u \ ! \ j < y \ ! \ j$

shows $\text{sum-list } (\text{inc } y \ i \ u) = \text{Suc } (\text{sum-list } u)$

<proof>

lemma *sum-list-us-eq*:

assumes $i < \text{sum-list } y$

shows $\text{sum-list } (\mathbf{u} \ y \ i) = i + 1$

<proof>

lemma *inc-ge*: $\text{length } u = \text{length } y \implies u \leq_v \text{inc } y \ i \ u$
 ⟨*proof*⟩

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

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

context *hlde*
begin

lemma *max-coeff-bound-right*:
 assumes $(xs, ys) \in \text{Minimal-Solutions}$
 shows $\forall x \in \text{set } xs. x \leq \text{maxne0 } ys \ b$ (is $\forall x \in \text{set } xs. x \leq ?m$)
 ⟨*proof*⟩

Proof of Lemma 1 of Huet’s paper.

lemma *max-coeff-bound*:
 assumes $(xs, ys) \in \text{Minimal-Solutions}$
 shows $(\forall x \in \text{set } xs. x \leq \text{maxne0 } ys \ b) \wedge (\forall y \in \text{set } ys. y \leq \text{maxne0 } xs \ a)$
 ⟨*proof*⟩

lemma *max-coeff-bound'*:
 assumes $(x, y) \in \text{Minimal-Solutions}$
 shows $\forall i < \text{length } x. x \ ! \ i \leq \text{Max } (\text{set } b)$ and $\forall j < \text{length } y. y \ ! \ j \leq \text{Max } (\text{set } a)$
 ⟨*proof*⟩

lemma *Minimal-Solutions-alt-def*:
 $\text{Minimal-Solutions} = \{(x, y) \in \text{Solutions}.$
 $(x, y) \neq (\text{zeroes } m, \text{zeroes } n) \wedge$
 $x \leq_v \text{replicate } m \ (\text{Max } (\text{set } b)) \wedge$
 $y \leq_v \text{replicate } n \ (\text{Max } (\text{set } a)) \wedge$
 $\neg (\exists (u, v) \in \text{Solutions}. \text{nonzero } u \wedge u \ @ \ v <_v x \ @ \ y)\}$
 ⟨*proof*⟩

2.3 Special Solutions

definition *Special-Solutions* :: $(\text{nat list} \times \text{nat list}) \text{ set}$
where
 $\text{Special-Solutions} = \{sij \ i \ j \mid i \ j. i < m \wedge j < n\}$

lemma *dij-neq-0*:
 assumes $i < m$
 and $j < n$

shows $d_{ij} \neq 0$
<proof>

lemma *eij-neq-0*:
assumes $i < m$
and $j < n$
shows $e_{ij} \neq 0$
<proof>

lemma *Special-Solutions-in-Solutions*:
 $x \in \text{Special-Solutions} \implies x \in \text{Solutions}$
<proof>

lemma *Special-Solutions-in-Minimal-Solutions*:
assumes $(x, y) \in \text{Special-Solutions}$
shows $(x, y) \in \text{Minimal-Solutions}$
<proof>

lemma *non-special-solution-non-minimal*:
assumes $(x, y) \in \text{Solutions} - \text{Special-Solutions}$
and $ij: i < m \ j < n$
and $x ! i \geq d_{ij} \ i \ j$ **and** $y ! j \geq e_{ij} \ i \ j$
shows $(x, y) \notin \text{Minimal-Solutions}$
<proof>

2.4 Huet's conditions

definition *cond-A* $xs \ ys \longleftrightarrow (\forall x \in \text{set } xs. x \leq \text{maxne0 } ys \ b)$

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

definition *boundr* $x \ y \longleftrightarrow (\forall j < n. y ! j \leq \text{max-y } x \ j)$

definition *cond-D* $x \ y \longleftrightarrow (\forall l \leq n. \text{take } l \ b \cdot \text{take } l \ y \leq a \cdot x)$

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

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

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

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

lemma *boundr*:

assumes $\text{min}: (x, y) \in \text{Minimal-Solutions}$
and $(x, y) \notin \text{Special-Solutions}$
shows $\text{boundr } x \ y$

<proof>

lemma *boundl*:

assumes $\text{min}: (x, y) \in \text{Minimal-Solutions}$
and $(x, y) \notin \text{Special-Solutions}$
shows $\text{boundl } x \ y$

<proof>

lemma *Solution-imp-cond-D*:

assumes $(x, y) \in \text{Solutions}$
shows $\text{cond-D } x \ y$

<proof>

lemma *Solution-imp-subprodl*:

assumes $(x, y) \in \text{Solutions}$
shows $\text{subprodl } x \ y$

<proof>

theorem *conds*:

assumes $\text{min}: (x, y) \in \text{Minimal-Solutions}$
shows $\text{cond-A}: \text{cond-A } x \ y$
and $\text{cond-B}: (x, y) \notin \text{Special-Solutions} \implies \text{cond-B } x$
and $(x, y) \notin \text{Special-Solutions} \implies \text{boundr } x \ y$
and $\text{cond-D}: \text{cond-D } x \ y$
and $\text{subprodr}: (x, y) \notin \text{Special-Solutions} \implies \text{subprodr } y$
and $\text{subprodl}: \text{subprodl } x \ y$

<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 $\text{length } x \leq m$
shows $\text{max-y } u \ j \geq \text{max-y } x \ j$

<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 $\text{length } y \leq n$

shows $\text{max-x } v \ i \geq \text{max-x } y \ i$

<proof>

end

end

theory *Sorted-Wrt*

imports *Main*

begin

lemma *sorted-wrt-filter*:

$\text{sorted-wrt } P \ xs \implies \text{sorted-wrt } P \ (\text{filter } Q \ xs)$

<proof>

lemma *sorted-wrt-map-mono*:

assumes *sorted-wrt* $Q \ xs$

and $\bigwedge x \ y. Q \ x \ y \implies P \ (f \ x) \ (f \ y)$

shows *sorted-wrt* $P \ (\text{map } f \ xs)$

<proof>

lemma *sorted-wrt-concat-map-map*:

assumes *sorted-wrt* $Q \ xs$

and *sorted-wrt* $Q \ ys$

and $\bigwedge a \ x \ y. Q \ x \ y \implies P \ (f \ x \ a) \ (f \ y \ a)$

and $\bigwedge x \ y \ u \ v. x \in \text{set } xs \implies y \in \text{set } xs \implies Q \ u \ v \implies P \ (f \ x \ u) \ (f \ y \ v)$

shows *sorted-wrt* $P \ [f \ x \ y \ . \ y \leftarrow ys, \ x \leftarrow xs]$

<proof>

lemma *sorted-wrt-concat-map*:

assumes *sorted-wrt* $P \ (\text{map } h \ xs)$

and $\bigwedge x. x \in \text{set } xs \implies \text{sorted-wrt } P \ (\text{map } h \ (f \ x))$

and $\bigwedge x \ y \ u \ v. P \ (h \ x) \ (h \ y) \implies x \in \text{set } xs \implies y \in \text{set } xs \implies u \in \text{set } (f \ x)$

$\implies v \in \text{set } (f \ y) \implies P \ (h \ u) \ (h \ v)$

shows *sorted-wrt* $P \ (\text{concat } (\text{map } (\text{map } h \circ f) \ xs))$

<proof>

lemma *sorted-wrt-map-distr*:

assumes *sorted-wrt* $(\lambda x \ y. P \ x \ y) \ (\text{map } f \ xs)$

shows *sorted-wrt* $(\lambda x \ y. P \ (f \ x) \ (f \ y)) \ xs$

<proof>

lemma *sorted-wrt-tl*:

$xs \neq [] \implies \text{sorted-wrt } P \ xs \implies \text{sorted-wrt } P \ (tl \ xs)$
 ⟨proof⟩

end

3 Minimization

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

fun *minimize-wrt*
where

$\text{minimize-wrt } P \ [] = []$
 | $\text{minimize-wrt } P \ (x \# \ xs) = x \# \ \text{filter } (P \ x) \ (\text{minimize-wrt } P \ xs)$

lemma *minimize-wrt-subset*: $\text{set } (\text{minimize-wrt } P \ xs) \subseteq \text{set } xs$
 ⟨proof⟩

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

lemma *sorted-wrt-minimize-wrt*:
 $\text{sorted-wrt } P \ (\text{minimize-wrt } P \ xs)$
 ⟨proof⟩

lemma *sorted-wrt-imp-sorted-wrt-minimize-wrt*:
 $\text{sorted-wrt } Q \ xs \implies \text{sorted-wrt } Q \ (\text{minimize-wrt } P \ xs)$
 ⟨proof⟩

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

lemma *in-minimize-wrtI*:
assumes $x \in \text{set } xs$
and $\forall y \in \text{set } xs. \ P \ y \ x$
shows $x \in \text{set } (\text{minimize-wrt } P \ xs)$
 ⟨proof⟩

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

lemma *minimize-wrt-ni*:

assumes $x \in \text{set } xs$

and $x \notin \text{set } (\text{minimize-wrt } Q \text{ } xs)$

shows $\exists y \in \text{set } xs. (\neg Q \ y \ x) \wedge x \neq y$

<proof>

lemma *in-minimize-wrtD*:

assumes $\bigwedge x \ y. Q \ x \ y \implies \neg Q \ y \ x$

and *sorted-wrt* $Q \ xs$

and $x \in \text{set } (\text{minimize-wrt } P \text{ } xs)$

and $\bigwedge x \ y. \neg P \ x \ y \implies Q \ x \ y$

and $\bigwedge x. P \ x \ x$

shows $x \in \text{set } xs \wedge (\forall y \in \text{set } xs. P \ y \ x)$

<proof>

lemma *in-minimize-wrt-iff*:

assumes $\bigwedge x \ y. Q \ x \ y \implies \neg Q \ y \ x$

and *sorted-wrt* $Q \ xs$

and $\bigwedge x \ y. \neg P \ x \ y \implies Q \ x \ y$

and $\bigwedge x. P \ x \ x$

shows $x \in \text{set } (\text{minimize-wrt } P \text{ } xs) \iff x \in \text{set } xs \wedge (\forall y \in \text{set } xs. P \ y \ x)$

<proof>

lemma *set-minimize-wrt*:

assumes $\bigwedge x \ y. Q \ x \ y \implies \neg Q \ y \ x$

and *sorted-wrt* $Q \ xs$

and $\bigwedge x \ y. \neg P \ x \ y \implies Q \ x \ y$

and $\bigwedge x. P \ x \ x$

shows $\text{set } (\text{minimize-wrt } P \text{ } xs) = \{x \in \text{set } xs. \forall y \in \text{set } xs. P \ y \ x\}$

<proof>

lemma *minimize-wrt-append*:

assumes $\forall x \in \text{set } xs. \forall y \in \text{set } (xs \ @ \ ys). P \ y \ x$

shows $\text{minimize-wrt } P \ (xs \ @ \ ys) = xs \ @ \ \text{filter } (\lambda y. \forall x \in \text{set } xs. P \ x \ y) \ (\text{minimize-wrt } P \ ys)$

<proof>

end

theory *Simple-Algorithm*

imports

Linear-Diophantine-Equations

Minimize-Wrt

begin

lemma *concat-map-nth0*: $xs \neq [] \implies f \ (xs \ ! \ 0) \neq [] \implies \text{concat } (\text{map } f \ xs) \ ! \ 0 = f \ (xs \ ! \ 0) \ ! \ 0$

<proof>

3.1 Reverse-Lexicographic Enumeration of Potential Minimal Solutions

fun *rlex2* :: (nat list × nat list) ⇒ (nat list × nat list) ⇒ bool (**infix** <*rlex2*>
50)

where

(*xs*, *ys*) <*rlex2* (*us*, *vs*) ↔ *xs* @ *ys* <*rlex* *us* @ *vs*

lemma *rlex2-irrefl*:

¬ *x* <*rlex2* *x*

<proof>

lemma *rlex2-not-sym*: *x* <*rlex2* *y* ⇒ ¬ *y* <*rlex2* *x*

<proof>

lemma *less-imp-rlex2*: ¬ (case *x* of (*x*, *y*) ⇒ λ(*u*, *v*). ¬ *x* @ *y* <_{*v*} *u* @ *v*) *y* ⇒
x <*rlex2* *y*

<proof>

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

fun *gen* :: nat ⇒ nat ⇒ nat list list

where

gen *B* 0 = [[]]

| *gen* *B* (*Suc* *n*) = [*x*#*xs* . *xs* ← *gen* *B* *n*, *x* ← [0 ..< *B* + 1]]

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

definition *check* *a* *b* = filter (λ(*x*, *y*). *a* · *x* = *b* · *y*)

definition *minimize* = minimize-wrt (λ(*x*, *y*) (*u*, *v*). ¬ *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* ∧ (∀ *i*<*n*. *xs* ! *i* ≤ *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*)

$\langle \text{proof} \rangle$

lemma *gen-ne* [simp]: $\text{gen } B \ n \neq [] \ \langle \text{proof} \rangle$

lemma *gen2-ne*: $\text{gen2 } A \ B \ m \ n \neq [] \ \langle \text{proof} \rangle$

lemma *sorted-wrt-generate*: $\text{sorted-wrt } (<_{\text{rlEx2}}) \ (\text{generate } A \ B \ m \ n) \ \langle \text{proof} \rangle$

abbreviation *check-generate* $a \ b \equiv \text{check } a \ b \ (\text{generate } (\text{Max } (\text{set } b)) \ (\text{Max } (\text{set } a)) \ (\text{length } a) \ (\text{length } b))$

lemma *sorted-wrt-check-generate*: $\text{sorted-wrt } (<_{\text{rlEx2}}) \ (\text{check-generate } a \ b) \ \langle \text{proof} \rangle$

lemma *in-tl-gen2*: $x \in \text{set } (\text{tl } (\text{gen2 } A \ B \ m \ n)) \implies x \in \text{set } (\text{gen2 } A \ B \ m \ n) \ \langle \text{proof} \rangle$

lemma *gen-nth0* [simp]: $\text{gen } B \ n \ ! \ 0 = \text{zeroes } n \ \langle \text{proof} \rangle$

lemma *gen2-nth0* [simp]:
 $\text{gen2 } A \ B \ m \ n \ ! \ 0 = (\text{zeroes } m, \text{zeroes } n) \ \langle \text{proof} \rangle$

lemma *set-gen2*:
 $\text{set } (\text{gen2 } A \ B \ m \ n) = \{(x, y). \text{length } x = m \wedge \text{length } y = n \wedge (\forall i < m. x \ ! \ i \leq A) \wedge (\forall j < n. y \ ! \ j \leq B)\} \ \langle \text{proof} \rangle$

lemma *gen2-unique*:
assumes $i < j$
and $j < \text{length } (\text{gen2 } A \ B \ m \ n)$
shows $\text{gen2 } A \ B \ m \ n \ ! \ i \neq \text{gen2 } A \ B \ m \ n \ ! \ j \ \langle \text{proof} \rangle$

lemma *zeroes-ni-tl-gen2*:
 $(\text{zeroes } m, \text{zeroes } n) \notin \text{set } (\text{tl } (\text{gen2 } A \ B \ m \ n)) \ \langle \text{proof} \rangle$

lemma *set-generate*:
 $\text{set } (\text{generate } A \ B \ m \ n) = \{(x, y). (x, y) \neq (\text{zeroes } m, \text{zeroes } n) \wedge (x, y) \in \text{set } (\text{gen2 } A \ B \ m \ n)\} \ \langle \text{proof} \rangle$

lemma *set-check-generate*:
 $\text{set } (\text{check-generate } a \ b) = \{(x, y). (x, y) \neq (\text{zeroes } (\text{length } a), \text{zeroes } (\text{length } b)) \wedge \text{length } x = \text{length } a \wedge \text{length } y = \text{length } b \wedge a \cdot x = b \cdot y \wedge$

$(\forall i < \text{length } a. x ! i \leq \text{Max } (\text{set } b)) \wedge (\forall j < \text{length } b. y ! j \leq \text{Max } (\text{set } a))\}$
 <proof>

lemma *set-minimize-check-generate*:

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

lemma *set-solutions-iff*:

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

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

lemma (in *hlde*) *solutions-complete*:

$\text{Minimal-Solutions} \subseteq \text{set } (\text{solutions } a \ b)$
 <proof>

3.1.2 Correctness: *solutions* generates only minimal solutions.

lemma (in *hlde*) *solutions-sound*:

$\text{set } (\text{solutions } a \ b) \subseteq \text{Minimal-Solutions}$
 <proof>

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

end

4 Computing Minimal Complete Sets of Solutions

theory *Algorithm*

imports *Simple-Algorithm*

begin

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

lemma *concat-map-filter-filter*:

assumes $\bigwedge x. x \in \text{set } xs \implies \neg Q \ x \implies \text{filter } P \ (f \ x) = []$

shows $\text{concat } (\text{map } (\text{filter } P \circ f) \ (\text{filter } Q \ xs)) = \text{concat } (\text{map } (\text{filter } P \circ f) \ xs)$

<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*:

$sorted-wrt (<_{rie x}) (map fst (alls B xs))$

<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*:

$set (alls2 A B a b) = \{((x, s), (y, t)). length x = length a \wedge length y = length b$
 \wedge

$(\forall i < length a. x ! i \leq A) \wedge (\forall j < length b. y ! j \leq B) \wedge s = a \cdot x \wedge t = b \cdot y\}$

<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*: $alls2\ A\ B\ as\ bs =$
 $((zeroes\ (length\ as),\ 0),\ (zeroes\ (length\ bs),\ 0))\ \# \ tl\ (alls2\ A\ B\ as\ bs)$
 $\langle proof \rangle$

abbreviation *gen2*

where

$gen2\ A\ B\ a\ b \equiv map\ (\lambda(x,\ y).\ (fst\ x,\ fst\ y))\ (alls2\ A\ B\ a\ b)$

lemma *sorted-wrt-gen2*:

$sorted-wrt\ (<_{rllex2})\ (gen2\ A\ B\ a\ b)$
 $\langle proof \rangle$

definition *generate'*

where

$generate'\ A\ B\ a\ b = tl\ (map\ (\lambda(x,\ y).\ (fst\ x,\ fst\ y))\ (alls2\ A\ B\ a\ b))$

lemma *sorted-wrt-generate'*:

$sorted-wrt\ (<_{rllex2})\ (generate'\ A\ B\ a\ b)$
 $\langle proof \rangle$

lemma *gen2-nth0* [simp]:

$gen2\ A\ B\ a\ b\ !\ 0 = (zeroes\ (length\ a),\ zeroes\ (length\ b))$
 $\langle proof \rangle$

lemma *gen2-ne* [simp, intro]: $gen2\ m\ n\ b\ c \neq [] \langle proof \rangle$

lemma *in-generate'*: $x \in set\ (generate'\ m\ n\ c\ b) \implies x \in set\ (gen2\ m\ n\ c\ b)$
 $\langle proof \rangle$

definition *cond-cons* $P = (\lambda(ys,\ s).\ case\ ys\ of\ [] \Rightarrow True \mid ys \Rightarrow P\ ys\ s)$

lemma *cond-cons-simp* [simp]:

$cond-cons\ P\ ([],\ s) = True$
 $cond-cons\ P\ (x\ \# \ xs,\ s) = P\ (x\ \# \ xs)\ s$
 $\langle proof \rangle$

fun *suffs*

where

$suffs\ P\ as\ (xs,\ s) \longleftrightarrow$
 $length\ xs = length\ as \wedge$
 $s = as \cdot xs \wedge$
 $(\forall i \leq length\ xs.\ cond-cons\ P\ (drop\ i\ xs,\ drop\ i\ as \cdot drop\ i\ xs))$

declare *suffs.simps* [simp del]

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

lemma *suffs-Cons*:

$\text{suffs } P (a \# as) (x \# xs, s) \longleftrightarrow$
 $s = a * x + as \cdot xs \wedge \text{cond-cons } P (x \# xs, s) \wedge \text{suffs } P as (xs, as \cdot xs)$
 ⟨proof⟩

4.1 The Algorithm

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

lemma *maxne0-impl*:
assumes $\text{length } x = \text{length } a$
shows $\text{maxne0-impl } x a = \text{maxne0 } x a$
 ⟨proof⟩

lemma *maxne0-impl-le*:
 $\text{maxne0-impl } x a \leq \text{Max } (\text{set } (a::\text{nat list}))$
 ⟨proof⟩

context
fixes $a b :: \text{nat list}$
begin

definition *special-solutions* :: $(\text{nat list} \times \text{nat list}) \text{ list}$
where
 $\text{special-solutions} = [\text{sij } a b i j . i \leftarrow [0 \dots < \text{length } a], j \leftarrow [0 \dots < \text{length } b]]$

definition *big-e* :: $\text{nat list} \Rightarrow \text{nat} \Rightarrow \text{nat list}$
where
 $\text{big-e } x j = \text{map } (\lambda i. \text{eij } a b i j - 1) (\text{filter } (\lambda i. x ! i \geq \text{dij } a b i j) [0 \dots < \text{length } x])$

definition *big-d* :: $\text{nat list} \Rightarrow \text{nat} \Rightarrow \text{nat list}$
where
 $\text{big-d } y i = \text{map } (\lambda j. \text{dij } a b i j - 1) (\text{filter } (\lambda j. y ! j \geq \text{eij } a b i j) [0 \dots < \text{length } y])$

definition *big-d'* :: $\text{nat list} \Rightarrow \text{nat} \Rightarrow \text{nat list}$
where
 $\text{big-d}' y i =$
 $(\text{let } l = \text{length } y; n = \text{length } b \text{ in}$
 $\text{if } l > n \text{ then } [] \text{ else}$
 $(\text{let } k = n - l \text{ in}$
 $\text{map } (\lambda j. \text{dij } a b i (j + k) - 1) (\text{filter } (\lambda j. y ! j \geq \text{eij } a b i (j + k)) [0 \dots < \text{length } y])))$

definition $max\text{-}y\text{-}impl :: nat\ list \Rightarrow nat \Rightarrow nat$

where

$max\text{-}y\text{-}impl\ x\ j =$
(if $j < length\ b \wedge big\text{-}e\ x\ j \neq []$ then $Min\ (set\ (big\text{-}e\ x\ j))$
else $Max\ (set\ a)$)

definition $max\text{-}x\text{-}impl :: nat\ list \Rightarrow nat \Rightarrow nat$

where

$max\text{-}x\text{-}impl\ y\ i =$
(if $i < length\ a \wedge big\text{-}d\ y\ i \neq []$ then $Min\ (set\ (big\text{-}d\ y\ i))$
else $Max\ (set\ b)$)

definition $max\text{-}x\text{-}impl' :: nat\ list \Rightarrow nat \Rightarrow nat$

where

$max\text{-}x\text{-}impl'\ y\ i =$
(if $i < length\ a \wedge big\text{-}d'\ y\ i \neq []$ then $Min\ (set\ (big\text{-}d'\ y\ i))$
else $Max\ (set\ b)$)

definition $cond\text{-}a :: nat\ list \Rightarrow nat\ list \Rightarrow bool$

where

$cond\text{-}a\ xs\ ys \longleftrightarrow (\forall x \in set\ xs. x \leq maxne0\ ys\ b)$

definition $cond\text{-}b :: nat\ list \Rightarrow bool$

where

$cond\text{-}b\ xs \longleftrightarrow (\forall k \leq length\ a.$
 $take\ k\ a \cdot take\ k\ xs \leq b \cdot (map\ (max\text{-}y\text{-}impl\ (take\ k\ xs))\ [0 ..< length\ b]))$

definition $boundr\text{-}impl :: nat\ list \Rightarrow nat\ list \Rightarrow bool$

where

$boundr\text{-}impl\ x\ y \longleftrightarrow (\forall j < length\ b. y\ !\ j \leq max\text{-}y\text{-}impl\ x\ j)$

definition $cond\text{-}d :: nat\ list \Rightarrow nat\ list \Rightarrow bool$

where

$cond\text{-}d\ xs\ ys \longleftrightarrow (\forall l \leq length\ b. take\ l\ b \cdot take\ l\ ys \leq a \cdot xs)$

definition $subdprodr\text{-}impl :: nat\ list \Rightarrow bool$

where

$subdprodr\text{-}impl\ ys \longleftrightarrow (\forall l \leq length\ b.$
 $take\ l\ b \cdot take\ l\ ys \leq a \cdot map\ (max\text{-}x\text{-}impl\ (take\ l\ ys))\ [0 ..< length\ a])$

definition $subdprodl\text{-}impl :: nat\ list \Rightarrow nat\ list \Rightarrow bool$

where

$subdprodl\text{-}impl\ x\ y \longleftrightarrow (\forall k \leq length\ a. take\ k\ a \cdot take\ k\ x \leq b \cdot y)$

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

definition $static\text{-}bounds$

where

$static\text{-}bounds\ x\ y \longleftrightarrow$

(let $mx = \text{maxne0-impl } y \ b$; $my = \text{maxne0-impl } x \ a$ in
 $(\forall x \in \text{set } x. x \leq mx) \wedge (\forall y \in \text{set } y. y \leq my)$)

definition *check-cond* =

$(\lambda(x, y). \text{static-bounds } x \ y \wedge a \cdot x = b \cdot y \wedge \text{boundr-impl } x \ y \wedge \text{subdprodl-impl } x \ y \wedge \text{subdprodr-impl } y)$

definition *check'* = *filter check-cond*

definition *non-special-solutions* =

(let $A = \text{Max } (\text{set } b)$; $B = \text{Max } (\text{set } a)$
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'*:

$\text{length } ys \leq \text{length } b \implies \text{set } (\text{big-d}' \ a \ b \ ys \ i) = \text{hlde-ops.Di}' \ a \ b \ i \ ys$
<proof>

lemma *max-y-impl*:

max-y-impl $a \ b \ x \ j = \text{hlde-ops.max-y}$ $a \ b \ x \ j$
<proof>

lemma *max-x-impl*:

max-x-impl $a \ b \ y \ i = \text{hlde-ops.max-x}$ $a \ b \ y \ i$
<proof>

lemma *max-x-impl'*:

assumes $\text{length } y \leq \text{length } b$
shows *max-x-impl'* $a \ b \ y \ i = \text{hlde-ops.max-x}'$ $a \ b \ y \ i$
<proof>

lemma (in *hlde*) *cond-a* [*simp*]: *cond-a* $b \ x \ y = \text{cond-A}$ $x \ y$

<proof>

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

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

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

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

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

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

lemma (in *hlde*) *check* [*simp*]:
 $\text{check}' \ a \ b =$
 $\text{filter } (\lambda(x, y). \text{static-bounds } a \ b \ x \ y \wedge a \cdot x = b \cdot y \wedge \text{boundr } x \ y \wedge$
 $\text{subdprodl } x \ y \wedge$
 $\text{subdprodr } y)$
 ⟨*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*: $u \leq_v x \ v \leq_v y$
and *len*: $\text{length } x = m \ \text{length } y = n$
shows $\text{cond-B } x \implies \text{cond-B } u$
and $\text{boundr } x \ y \implies \text{boundr } u \ v$
and $a \cdot u = b \cdot v \implies \text{cond-D } x \ y \implies \text{cond-D } u \ v$
and $a \cdot u = b \cdot v \implies \text{subdprodl } x \ y \implies \text{subdprodl } u \ v$
and $\text{subdprodr } y \implies \text{subdprodr } v$
 ⟨*proof*⟩

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

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

lemma *set-gen2'*:
 $(\lambda(x, y). (\text{fst } x, \text{fst } y)) \text{ ' set } (\text{alls2 } A \ B \ a \ b) =$

$\{(x, y). x \leq_v \text{replicate}(\text{length } a) A \wedge y \leq_v \text{replicate}(\text{length } b) B\}$
 ⟨proof⟩

lemma (in *hlde*) *in-non-special-solutions*:
 assumes $(x, y) \in \text{set}(\text{non-special-solutions } a \ b)$
 shows $(x, y) \in \text{Solutions}$
 ⟨proof⟩

lemma *generate-unique*:
 assumes $i < j$
 and $j < \text{length}(\text{generate } A \ B \ a \ b)$
 shows $\text{generate } A \ B \ a \ b ! i \neq \text{generate } A \ B \ a \ b ! j$
 ⟨proof⟩

lemma *gen2-unique*:
 assumes $i < j$
 and $j < \text{length}(\text{gen2 } A \ B \ a \ b)$
 shows $\text{gen2 } A \ B \ a \ b ! i \neq \text{gen2 } A \ B \ a \ b ! j$
 ⟨proof⟩

lemma *zeroes-ni-generate'*:
 $(\text{zeroes}(\text{length } a), \text{zeroes}(\text{length } b)) \notin \text{set}(\text{generate}' \ A \ B \ a \ b)$
 ⟨proof⟩

lemma *set-generate'*:
 $\text{set}(\text{generate}' \ A \ B \ a \ b) =$
 $\{(x, y). (x, y) \neq (\text{zeroes}(\text{length } a), \text{zeroes}(\text{length } b)) \wedge (x, y) \in \text{set}(\text{gen2 } A \ B \ a \ b)\}$
 ⟨proof⟩

lemma *set-generate''*:
 $\text{set}(\text{generate}' \ A \ B \ a \ b) =$
 $\{(x, y). (x, y) \neq (\text{zeroes}(\text{length } a), \text{zeroes}(\text{length } b)) \wedge x \leq_v \text{replicate}(\text{length } a) \ A \wedge y \leq_v \text{replicate}(\text{length } b) \ B\}$
 ⟨proof⟩

lemma (in *hlde*) *zeroes-ni-non-special-solutions*:
 shows $(\text{zeroes } m, \text{zeroes } n) \notin \text{set}(\text{non-special-solutions } a \ b)$
 ⟨proof⟩

4.1.1 Correctness: *solve* generates only minimal solutions.

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

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

lemma (in *hlde*) *Minimal-Solutions-subset-solve*:
 shows $\text{Minimal-Solutions} \subseteq \text{set}(\text{solve } a \ b)$

<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* \Rightarrow *nat* \Rightarrow *bool*
 and *B* :: *nat*
 assumes *bound*: $\bigwedge x \ xs \ s. \ x > B \Longrightarrow C \ (x \# \ xs) \ s = False$
 and *cond-antimono*: $\bigwedge x \ x' \ xs \ s \ s'. \ C \ (x \# \ xs) \ s \Longrightarrow x' \leq x \Longrightarrow s' \leq s \Longrightarrow C \ (x' \# \ xs) \ s'$
begin

function *incs* :: *nat* \Rightarrow *nat* \Rightarrow (*nat list* \times *nat*) \Rightarrow (*nat list* \times *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*) \in *set (incs a x (xs, s))*
 shows *length ys = length xs + 1* \wedge *t = s + hd ys * a* \wedge *tl ys = xs* \wedge *C ys t*
 <proof>

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

lemma *incs-filter*:

assumes $x \leq B$
 shows *incs a x =* ($\lambda(xs, s). \text{filter } (\text{cond-cons } C) (\text{map } (\lambda x. (x \# xs, s + a * x))$
 $[x ..< B + 1])$)
 <proof>

fun *gen-check* :: *nat list* \Rightarrow (*nat list* \times *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*) \in *set (gen-check as)*
 shows *length ys = length as*

<proof>

lemma *in-gen-check*:

assumes $(xs, s) \in \text{set } (\text{gen-check } as)$

shows $\text{length } xs = \text{length } as \wedge s = as \cdot xs$

<proof>

lemma *gen-check-filter*:

$\text{gen-check } as = \text{filter } (\text{suffs } C \ as) \ (\text{alls } B \ as)$

<proof>

lemma *in-gen-check-cond*:

assumes $(xs, s) \in \text{set } (\text{gen-check } as)$

shows $\forall j \leq \text{length } xs. \text{drop } j \ xs \neq [] \longrightarrow C \ (\text{drop } j \ xs) \ (s - \text{take } j \ as \cdot \text{take } j \ xs)$

<proof>

lemma *sorted-gen-check*:

$\text{sorted-wrt } (<_{\text{rlex}}) \ (\text{map } \text{fst } (\text{gen-check } xs))$

<proof>

end

locale *bounded-generate-check* =

$c2: \text{bounded-gen-check } C_2 \ B_2 \ \text{for } C_2 \ B_2 +$

fixes C_1 **and** B_1

assumes $\text{cond1}: \bigwedge b \ ys. \ ys \in \text{fst } ' \ \text{set } (c2.\text{gen-check } b) \implies \text{bounded-gen-check } (C_1 \ b \ ys) \ (B_1 \ b)$

begin

definition *generate-check* $a \ b =$

$[(xs, ys). \ ys \leftarrow c2.\text{gen-check } b, \ xs \leftarrow \text{bounded-gen-check}.\text{gen-check } (C_1 \ b \ (\text{fst } ys)) \ a]$

lemma *generate-check-filter-conv*:

$\text{generate-check } a \ b = [(xs, ys).$

$\ ys \leftarrow \text{filter } (\text{suffs } C_2 \ b) \ (\text{alls } B_2 \ b),$

$\ xs \leftarrow \text{filter } (\text{suffs } (C_1 \ b \ (\text{fst } ys)) \ a) \ (\text{alls } (B_1 \ b) \ a)]$

<proof>

lemma *generate-check-filter*:

$\text{generate-check } a \ b = [(xs, ys) \leftarrow \text{alls2 } (B_1 \ b) \ B_2 \ a \ b. \ \text{suffs } (C_1 \ b \ (\text{fst } ys)) \ a \ xs \wedge \text{suffs } C_2 \ b \ ys]$

<proof>

lemma *tl-generate-check-filter*:

assumes $\text{suffs } (C_1 \ b \ (\text{zeroes } (\text{length } b))) \ a \ (\text{zeroes } (\text{length } a), \ 0)$

and $\text{suffs } C_2 \ b \ (\text{zeroes } (\text{length } b), \ 0)$

shows $\text{tl } (\text{generate-check } a \ b) = [(xs, ys) \leftarrow \text{tl } (\text{alls2 } (B_1 \ b) \ B_2 \ a \ b). \ \text{suffs } (C_1 \ b \ (\text{fst } ys)) \ a \ xs \wedge \text{suffs } C_2 \ b \ ys]$

```

    <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 ≤v y
    and i < length a
  shows max-x-impl' a b v i ≥ max-x-impl' a b y i
  <proof>

end

global-interpretation c12: bounded-generate-check (cond2 a b) Max (set a) cond1

```

$\lambda b. \text{Max } (\text{set } b)$

defines $c2\text{-gen-check} = c12.c2.\text{gen-check}$ **and** $c2\text{-incs} = c12.c2.\text{incs}$
and $c12\text{-generate-check} = c12.\text{generate-check}$
 $\langle \text{proof} \rangle$

definition $\text{post-cond } a \ b = (\lambda(x, y). \text{static-bounds } a \ b \ x \ y \wedge a \cdot x = b \cdot y \wedge \text{boundr-impl } a \ b \ x \ y)$

definition $\text{fast-filter } a \ b =$
 $\text{filter } (\text{post-cond } a \ b) (\text{map } (\lambda(x, y). (\text{fst } x, \text{fst } y)) (\text{tl } (c12\text{-generate-check } a \ b \ a \ b)))$

lemma $\text{cond1-cond2-zeroes}$:

shows $\text{suffs } (\text{cond1 } b \ (\text{zeroes } (\text{length } b))) \ a \ (\text{zeroes } (\text{length } a), \ 0)$
and $\text{suffs } (\text{cond2 } a \ b) \ b \ (\text{zeroes } (\text{length } b), \ 0)$
 $\langle \text{proof} \rangle$

lemma suffs-cond1I :

assumes $\forall y \in \text{set } aa. y \leq \text{maxne0-impl } aaa \ b$
and $\text{length } aa = \text{length } a$
and $a \cdot aa = b \cdot aaa$
shows $\text{suffs } (\text{cond1 } b \ aaa) \ a \ (aa, \ b \cdot aaa)$
 $\langle \text{proof} \rangle$

lemma suffs-cond2-conv :

assumes $\text{length } ys = \text{length } b$
shows $\text{suffs } (\text{cond2 } a \ b) \ b \ (ys, \ b \cdot ys) \longleftrightarrow$
 $(\forall y \in \text{set } ys. y \leq \text{Max } (\text{set } a)) \wedge \text{subdprodr-impl } a \ b \ ys$
(is $?L \longleftrightarrow ?R$
 $\langle \text{proof} \rangle$

lemma suffs-cond2I :

assumes $\forall y \in \text{set } aaa. y \leq \text{Max } (\text{set } a)$
and $\text{length } aaa = \text{length } b$
and $\text{subdprodr-impl } a \ b \ aaa$
shows $\text{suffs } (\text{cond2 } a \ b) \ b \ (aaa, \ b \cdot aaa)$
 $\langle \text{proof} \rangle$

lemma check-cond-conv :

assumes $(x, y) \in \text{set } (\text{alls2 } (\text{Max } (\text{set } b)) (\text{Max } (\text{set } a)) \ a \ b)$
shows $\text{check-cond } a \ b \ (\text{fst } x, \ \text{fst } y) \longleftrightarrow$
 $\text{static-bounds } a \ b \ (\text{fst } x) \ (\text{fst } y) \wedge a \cdot \text{fst } x = b \cdot \text{fst } y \wedge \text{boundr-impl } a \ b \ (\text{fst } x)$
 $(\text{fst } y) \wedge$
 $\text{suffs } (\text{cond1 } b \ (\text{fst } y)) \ a \ x \wedge$
 $\text{suffs } (\text{cond2 } a \ b) \ b \ y$
 $\langle \text{proof} \rangle$

lemma tune :

$\text{check}' \ a \ b \ (\text{generate}' \ (\text{Max } (\text{set } b)) (\text{Max } (\text{set } a)) \ a \ b) = \text{fast-filter } a \ b$

```

    <proof>

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

function incs :: nat  $\Rightarrow$  nat  $\Rightarrow$  (nat list  $\times$  nat)  $\Rightarrow$  (nat list  $\times$  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)  $\in$  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]:  $x > B \implies incs a x (xs, s) = []$ 
    <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  $\leftarrow$  c2-gen-check a b b, xs  $\leftarrow$  c1-gen-check b (fst ys) a]

lemma c1-gen-check-conv:
  assumes (ys, s)  $\in$  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.