A Formally Verified Solver for Homogeneous Linear Diophantine Equations*

Florian Meßner   Julian Parsert   Jonas Schöpf   **Christian Sternagel**

Master Seminar 1

November 29, 2017

* Supported by the Austrian Science Fund (FWF): project P27502

Quiz – How many unifiers?

$$\mathsf{f}(x, y) \approx \mathsf{f}(z, z)$$

#unifiers: ?

Quiz – How many unifiers?

$$\mathsf{f}(x, y) \;\approx\; \mathsf{f}(z, z)$$

#unifiers: 1

$$\{x \mapsto z, y \mapsto z\}$$

Quiz – How many AC unifiers?

$$x \cdot y \approx z \cdot z$$

#unifiers: ?

Quiz – How many AC unifiers?

$$x \cdot y \approx z \cdot z$$

#unifiers: 5

minimal complete set of AC unifiers:

$$\{x \mapsto z_3, \quad y \mapsto z_3, \quad z \mapsto z_3\}$$
$$\{x \mapsto z_1 \cdot z_1, \quad y \mapsto z_2 \cdot z_2, \quad z \mapsto z_1 \cdot z_2\}$$
$$\{x \mapsto z_1 \cdot z_1 \cdot z_3, \quad y \mapsto z_3, \quad z \mapsto z_1 \cdot z_3\}$$
$$\{x \mapsto z_3, \quad y \mapsto z_2 \cdot z_2 \cdot z_3, \quad z \mapsto z_2 \cdot z_3\}$$
$$\{x \mapsto z_1 \cdot z_1 \cdot z_3, \quad y \mapsto z_2 \cdot z_2 \cdot z_3, \quad z \mapsto z_1 \cdot z_2 \cdot z_3\}$$

Quiz – How many AC unifiers?

$$x \cdot y \ \approx \ z \cdot z \cdot z$$

#unifiers: ?

Quiz – How many AC unifiers?

$$x \cdot y \approx z \cdot z \cdot z$$

#unifiers: 13

Quiz – How many AC unifiers?

$$v \cdot x \cdot y \;\approx\; z \cdot z \cdot z$$

#unifiers: ?

Quiz – How many AC unifiers?

$$v \cdot x \cdot y \;\approx\; z \cdot z \cdot z$$

#unifiers: 981

Bibliography

Michael Clausen and Albrecht Fortenbacher.
Efficient solution of linear diophantine equations.
Journal of Symbolic Computation, 8(1):201–216, 1989.
doi:10.1016/S0747-7171(89)80025-2.

Gérard Huet.
An algorithm to generate the basis of solutions to homogeneous linear diophantine equations.
Information Processing Letters, 7(3):144–147, 1978.
doi:10.1016/0020-0190(78)90078-9.

Florian Meßner, Julian Parsert, Jonas Schöpf, and Christian Sternagel.
Homogeneous Linear Diophantine Equations.
The Archive of Formal Proofs, October 2017.
https://www.isa-afp.org/entries/Diophantine_Eqns_Lin_Hom.shtml, Formal proof development.

Homogeneous Linear Diophantine Equations (HLDEs)

$$a_1 x_1 + a_2 x_2 + \cdots + a_m x_m \quad = \quad b_1 y_1 + b_2 y_2 + \cdots + b_n y_n$$

Homogeneous Linear Diophantine Equations (HLDEs)

$$a_1 x_1 + a_2 x_2 + \cdots + a_m x_m \quad = \quad b_1 y_1 + b_2 y_2 + \cdots + b_n y_n$$

left-hand side coefficients

Homogeneous Linear Diophantine Equations (HLDEs)

$$a_1 x_1 + a_2 x_2 + \cdots + a_m x_m \quad = \quad b_1 y_1 + b_2 y_2 + \cdots + b_n y_n$$

left-hand side variables

Homogeneous Linear Diophantine Equations (HLDEs)

$$a_1 x_1 + a_2 x_2 + \cdots + a_m x_m \quad = \quad b_1 y_1 + b_2 y_2 + \cdots + b_n y_n$$

right-hand side coefficients

Homogeneous Linear Diophantine Equations (HLDEs)

$$a_1 x_1 + a_2 x_2 + \cdots + a_m x_m \quad = \quad b_1 y_1 + b_2 y_2 + \cdots + b_n y_n$$

right-hand side variables

Homogeneous Linear Diophantine Equations (HLDEs)

$$a_1 x_1 + a_2 x_2 + \cdots + a_m x_m \quad = \quad b_1 y_1 + b_2 y_2 + \cdots + b_n y_n$$

Example

$$x_1 + x_2 = 2 y_1$$

Homogeneous Linear Diophantine Equations (HLDEs)

$$a \bullet x \quad = \quad b \bullet y$$

Example

$$x_1 + x_2 = 2\, y_1$$

Homogeneous Linear Diophantine Equations (HLDEs)

$$a \bullet x \quad = \quad b \bullet y$$

Example

$$[1, 1] \bullet [x_1, x_2] = [2] \bullet [y_1]$$

Homogeneous Linear Diophantine Equations (HLDEs)

$$a \bullet x \quad = \quad b \bullet y$$

Example

$$[1, 1] \bullet [x_1, x_2] = [2] \bullet [y_1]$$

Remark

we represent HLDEs by lists of coefficients, e.g., `([1,1],[2])`

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$
- set of (pointwise) minimal solutions
  $\mathcal{M} = \{(x, y) \in \mathcal{S} \mid x \neq 0, \nexists (u, v) \in \mathcal{S}.\ u \neq 0 \land u \mathbin{+\!+} v <_{\mathsf{v}} x \mathbin{+\!+} y\}$

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$
- set of (pointwise) minimal solutions
  $\mathcal{M} = \{(x, y) \in \mathcal{S} \mid x \neq 0, \nexists (u, v) \in \mathcal{S}.\ u \neq 0 \wedge u +\!\!+\ v <_{\mathsf{v}} x +\!\!+\ y\}$

$x <_{\mathsf{v}} y$ iff $x_i \leq y_i$ and $x \neq y$

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$
- set of (pointwise) minimal solutions
  $\mathcal{M} = \{(x, y) \in \mathcal{S} \mid x \neq 0, \nexists (u, v) \in \mathcal{S}.\ u \neq 0 \wedge u \mathbin{++} v <_{\mathsf{v}} x \mathbin{++} y\}$

Searching for Solutions

- given $x_1 + x_2 = 2\, y_1$, represented by [1,1] and [2]

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$
- set of (pointwise) minimal solutions
  $\mathcal{M} = \{(x, y) \in \mathcal{S} \mid x \neq 0, \nexists (u, v) \in \mathcal{S}.\ u \neq 0 \wedge u \mathbin{+\!+} v <_{\mathsf{v}} x \mathbin{+\!+} y\}$

Searching for Solutions

- given $x_1 + x_2 = 2 y_1$, represented by [1,1] and [2]
- consider potential solutions

x_1 x_2 y_1

[0,0], [0] ?

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$
- set of (pointwise) minimal solutions
  $\mathcal{M} = \{(x, y) \in \mathcal{S} \mid x \neq 0, \nexists (u, v) \in \mathcal{S}.\ u \neq 0 \wedge u \mathbin{++} v <_{\mathsf{v}} x \mathbin{++} y\}$

Searching for Solutions

- given $x_1 + x_2 = 2 y_1$, represented by [1,1] and [2]
- consider potential solutions

  x_1 x_2 y_1

  [0,0], [0] ✔ (trivial solution, not minimal)

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$
- set of (pointwise) minimal solutions
  $\mathcal{M} = \{(x, y) \in \mathcal{S} \mid x \neq 0, \nexists(u, v) \in \mathcal{S}.\ u \neq 0 \wedge u \mathbin{+\!+} v <_{\mathsf{v}} x \mathbin{+\!+} y\}$

Searching for Solutions

- given $x_1 + x_2 = 2 y_1$, represented by `[1,1]` and `[2]`
- consider potential solutions

  x_1 x_2 y_1
  `[0,0]`, `[0]` ✔ (trivial solution, not minimal)
  `[0,0]`, `[1]` ?

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$
- set of (pointwise) minimal solutions
  $\mathcal{M} = \{(x, y) \in \mathcal{S} \mid x \neq 0, \nexists(u, v) \in \mathcal{S}.\ u \neq 0 \wedge u \mathbin{++} v <_{\mathsf{v}} x \mathbin{++} y\}$

Searching for Solutions

- given $x_1 + x_2 = 2\,y_1$, represented by [1,1] and [2]
- consider potential solutions

  $x_1\ x_2 \qquad y_1$
  [0,0], [0] ✔ (trivial solution, not minimal)
  [0,0], [1] ✘

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$
- set of (pointwise) minimal solutions
  $\mathcal{M} = \{(x, y) \in \mathcal{S} \mid x \neq 0, \nexists (u, v) \in \mathcal{S}.\ u \neq 0 \land u \mathbin{++} v <_{\mathsf{v}} x \mathbin{++} y\}$

Searching for Solutions

- given $x_1 + x_2 = 2 y_1$, represented by [1,1] and [2]
- consider potential solutions

```
x1 x2    y1
[0,0],  [0] ✔ (trivial solution, not minimal)
[0,0],  [1] ✘
   ⋮
[1,1],  [1] ?
```

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$
- set of (pointwise) minimal solutions
  $\mathcal{M} = \{(x, y) \in \mathcal{S} \mid x \neq 0, \nexists (u, v) \in \mathcal{S}.\ u \neq 0 \wedge u \mathbin{++} v <_{\mathsf{v}} x \mathbin{++} y\}$

Searching for Solutions

- given $x_1 + x_2 = 2y_1$, represented by [1,1] and [2]
- consider potential solutions

x_1 x_2	y_1	
[0,0],	[0]	✔ (trivial solution, not minimal)
[0,0],	[1]	✘
⋮		
[1,1],	[1]	✔

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$
- set of (pointwise) minimal solutions
  $\mathcal{M} = \{(x, y) \in \mathcal{S} \mid x \neq 0, \nexists (u, v) \in \mathcal{S}.\ u \neq 0 \wedge u \mathbin{++} v <_{\mathsf{v}} x \mathbin{++} y\}$

Searching for Solutions

- given $x_1 + x_2 = 2y_1$, represented by [1,1] and [2]
- consider potential solutions

```
 x1 x2    y1
[0,0], [0] ✔ (trivial solution, not minimal)
[0,0], [1] ✘
   ⋮
[1,1], [1] ✔
   ⋮
[2,0], [1] ?
```

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$
- set of (pointwise) minimal solutions
  $\mathcal{M} = \{(x, y) \in \mathcal{S} \mid x \neq 0, \nexists(u, v) \in \mathcal{S}.\ u \neq 0 \wedge u +\!\!+\ v <_{\mathsf{v}} x +\!\!+\ y\}$

Searching for Solutions

- given $x_1 + x_2 = 2y_1$, represented by `[1,1]` and `[2]`
- consider potential solutions

$x_1\ x_2 \quad y_1$

`[0,0], [0]` ✔ (trivial solution, not minimal)

`[0,0], [1]` ✘

⋮

`[1,1], [1]` ✔

⋮

`[2,0], [1]` ✔

Solutions of HLDEs

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$
- set of (pointwise) minimal solutions
  $\mathcal{M} = \{(x, y) \in \mathcal{S} \mid x \neq 0, \nexists (u, v) \in \mathcal{S}.\ u \neq 0 \wedge u \mathbin{++} v <_{\mathsf{v}} x \mathbin{++} y\}$

Searching for Solutions

- given $x_1 + x_2 = 2 y_1$, represented by [1,1] and [2]
- consider potential solutions

x_1 x_2 y_1

[0,0], [0] ✔ (trivial solution, not minimal)
[0,0], [1] ✘
⋮
[1,1], [1] ✔
⋮
[2,0], [1] ✔
⋮
[3,1], [2] ?

- given lists of non-zero coefficients a and b (of lengths m and n)
- set of solutions $\mathcal{S} = \{(x, y) \mid a \bullet x = b \bullet y, |x| = m, |y| = n\}$
- set of (pointwise) minimal solutions
  $\mathcal{M} = \{(x, y) \in \mathcal{S} \mid x \neq 0, \nexists (u, v) \in \mathcal{S}.\ u \neq 0 \wedge u \mathbin{+\!\!+} v <_{\mathsf{v}} x \mathbin{+\!\!+} y\}$

Searching for Solutions

- given $x_1 + x_2 = 2y_1$, represented by [1,1] and [2]
- consider potential solutions

```
 x1 x2    y1
[0,0],  [0] ✔ (trivial solution, not minimal)
[0,0],  [1] ✘
    ⋮
[1,1],  [1] ✔
    ⋮
[2,0],  [1] ✔
    ⋮
[3,1],  [2] ✔ (but not minimal)
```

Bounding Minimal Solutions

Lemma (Huet)
if $(x, y) \in \mathcal{M}(a, b)$ then $x_i \leq \max(b)$ and $y_j \leq \max(a)$

Bounding Minimal Solutions

Lemma (Huet)
if $(x, y) \in \mathcal{M}(a, b)$ then $x_i \leq \max(b)$ and $y_j \leq \max(a)$

Example

- for $a =$ `[1,1]` and $b =$ `[2]`

Bounding Minimal Solutions

Lemma (Huet)

if $(x, y) \in \mathcal{M}(a, b)$ then $x_i \leq \max(b)$ and $y_j \leq \max(a)$

Example

- for $a = $ `[1,1]` and $b = $ `[2]`
- 18 potential solutions $(3^2 \cdot 2^1)$

```
[([0,0],[0]), ([1,0],[0]), ([2,0],[0]), ([0,1],[0]),
 ([1,1],[0]), ([2,1],[0]), ([0,2],[0]), ([1,2],[0]),
 ([2,2],[0]), ([0,0],[1]), ([1,0],[1]), ([2,0],[1]),
 ([0,1],[1]), ([1,1],[1]), ([2,1],[1]), ([0,2],[1]),
 ([1,2],[1]), ([2,2],[1])]
```

Bounding Minimal Solutions

Lemma (Huet)

if $(x, y) \in \mathcal{M}(a, b)$ then $x_i \leq \max(b)$ and $y_j \leq \max(a)$

Example

- for $a =$ `[1,1]` and $b =$ `[2]`
- 18 potential solutions $(3^2 \cdot 2^1)$

```
[([0,0],[0]), ([1,0],[0]), ([2,0],[0]), ([0,1],[0]),
 ([1,1],[0]), ([2,1],[0]), ([0,2],[0]), ([1,2],[0]),
 ([2,2],[0]), ([0,0],[1]), ([1,0],[1]), ([2,0],[1]),
 ([0,1],[1]), ([1,1],[1]), ([2,1],[1]), ([0,2],[1]),
 ([1,2],[1]), ([2,2],[1])]
```

- containing 4 actual solutions $(a \bullet x = b \bullet y)$

```
[([0,0],[0]),([2,0],[1]),([1,1],[1]),([0,2],[1])]
```

Bounding Minimal Solutions

Lemma (Huet)
if $(x, y) \in \mathcal{M}(a, b)$ then $x_i \leq \max(b)$ and $y_j \leq \max(a)$

Example

- for $a =$ [1,1] and $b =$ [2]
- 18 potential solutions $(3^2 \cdot 2^1)$

```
[([0,0],[0]), ([1,0],[0]), ([2,0],[0]), ([0,1],[0]),
 ([1,1],[0]), ([2,1],[0]), ([0,2],[0]), ([1,2],[0]),
 ([2,2],[0]), ([0,0],[1]), ([1,0],[1]), ([2,0],[1]),
 ([0,1],[1]), ([1,1],[1]), ([2,1],[1]), ([0,2],[1]),
 ([1,2],[1]), ([2,2],[1])]
```

- containing 4 actual solutions $(a \bullet x = b \bullet y)$

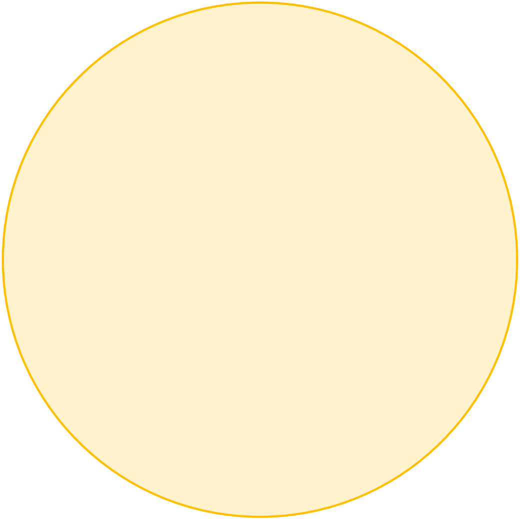
```
[([0,0],[0]),([2,0],[1]),([1,1],[1]),([0,2],[1])]
```

- of which 3 are minimal (w.r.t. $<_{\mathsf{v}}$)

```
[([2,0],[1]), ([0,2],[1]), ([1,1],[1])]
```

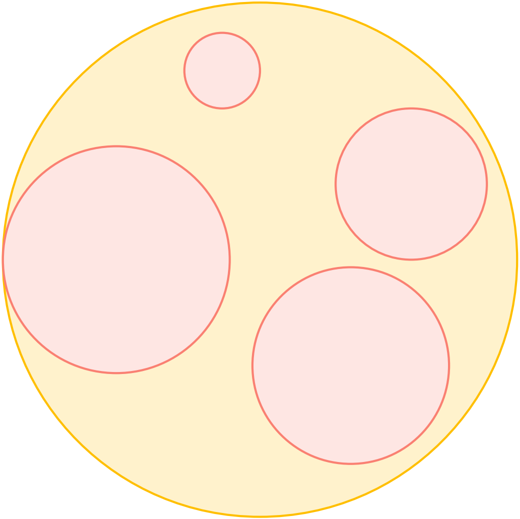
Computing Minimal Complete Sets of Solutions

1. generate potential solutions (crude overapproximation)



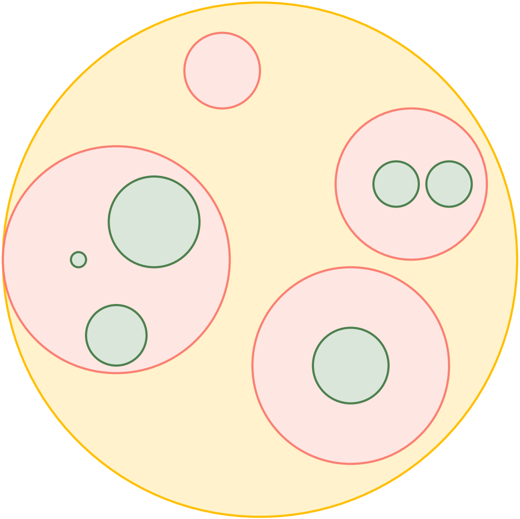
Computing Minimal Complete Sets of Solutions

1. generate potential solutions (crude overapproximation)
2. check for actual solutions



Computing Minimal Complete Sets of Solutions

1. generate potential solutions (crude overapproximation)
2. check for actual solutions
3. minimize remaining set of candidates



Phase 1 – Generate

- given bound `b` and coefficients `cs`

Phase 1 – Generate

- given bound `b` and coefficients `cs`
- compute all vectors of length equal to `length cs` within bound

```
gen b [] = [[]]
gen b (c:cs) = [x:xs | xs <- gen b cs, x <- [0 .. b]]
```

Phase 1 – Generate

- given bound `b` and coefficients `cs`
- compute all vectors of length equal to `length cs` within bound

```
gen b [] = [[]]
gen b (c:cs) = [x:xs | xs <- gen b cs, x <- [0 .. b]]
```

- given bounds `a`, `b` and coefficients `as`, `bs`

Example

- equation $x_1 + x_2 = 2y_1$, `a` $=$ `2`, `b` $=$ `1`, `as` $=$ `[1,1]`, `bs` $=$ `[2]`

Phase 1 – Generate

- given bound `b` and coefficients `cs`
- compute all vectors of length equal to `length cs` within bound

```
gen b [] = [[]]
gen b (c:cs) = [x:xs | xs <- gen b cs, x <- [0 .. b]]
```

- given bounds `a`, `b` and coefficients `as`, `bs`
- compute all potential solutions within bounds

```
generate a b as bs =
  tail [(x, y) | y <- gen b bs, x <- gen a as]
```

Example

- equation $x_1 + x_2 = 2y_1$, $\mathtt{a} = 2$, $\mathtt{b} = 1$, $\mathtt{as} = [1,1]$, $\mathtt{bs} = [2]$

Phase 1 – Generate

- given bound `b` and coefficients `cs`
- compute all vectors of length equal to `length cs` within bound

```
gen b [] = [[]]
gen b (c:cs) = [x:xs | xs <- gen b cs, x <- [0 .. b]]
```

- given bounds `a`, `b` and coefficients `as`, `bs`
- compute all potential solutions within bounds

```
generate a b as bs =
  tail [(x, y) | y <- gen b bs, x <- gen a as]
```

Example

- equation $x_1 + x_2 = 2y_1$, a = 2, b = 1, as = [1,1], bs = [2]

```
[0,0],    [1,0],    [2,0],    [0,1],    [1,1],
[2,1],    [0,2],    [1,2],    [2,2],
```

Phase 1 – Generate

- given bound `b` and coefficients `cs`
- compute all vectors of length equal to `length cs` within bound

```
gen b [] = [[]]
gen b (c:cs) = [x:xs | xs <- gen b cs, x <- [0 .. b]]
```

- given bounds `a`, `b` and coefficients `as`, `bs`
- compute all potential solutions within bounds

```
generate a b as bs =
  tail [(x, y) | y <- gen b bs, x <- gen a as]
```

- (solutions are generated in reverse lexicographic order $<_{\mathsf{rlex}}$)

Example

- equation $x_1 + x_2 = 2y_1$, `a` $=$ `2`, `b` $=$ `1`, `as` $=$ `[1,1]`, `bs` $=$ `[2]`

```
[0,0], [0]   [1,0], [0]   [2,0], [0]   [0,1], [0]   [1,1], [0]
[2,1], [0]   [0,2], [0]   [1,2], [0]   [2,2], [0]
[0,0], [1]   [1,0], [1]   [2,0], [1]   [0,1], [1]   [1,1], [1]
[2,1], [1]   [0,2], [1]   [1,2], [1]   [2,2], [1]
```

Phase 1 – Generate

- given bound b and coefficients cs
- compute all vectors of length equal to length cs within bound

```
gen b [] = [[]]
gen b (c:cs) = [x:xs | xs <- gen b cs, x <- [0 .. b]]
```

- given bounds a, b and coefficients as, bs
- compute all potential solutions within bounds

```
generate a b as bs =
  tail [(x, y) | y <- gen b bs, x <- gen a as]
```

- (solutions are generated in reverse lexicographic order $<_{\text{rlex}}$)

$x <_{\text{rlex}} y$ iff $\exists i.\, x_i < y_i \wedge \forall j > i.\, x_j = y_j$

Example

- equation $x_1 + x_2 = 2y_1$, a = 2, b = 1, as = [1,1], bs = [2]

```
            [1,0], [0]   [2,0], [0]   [0,1], [0]   [1,1], [0]
[2,1], [0]  [0,2], [0]   [1,2], [0]   [2,2], [0]
[0,0], [1]  [1,0], [1]   [2,0], [1]   [0,1], [1]   [1,1], [1]
[2,1], [1]  [0,2], [1]   [1,2], [1]   [2,2], [1]
```

Phase 2 – Check

- drop non-solutions

```
check as bs = filter (isSolution as bs)
```

Example

- equation $x_1 + x_2 = 2\,y_1$, a = 2, b = 1, as = [1,1], bs = [2]

```
              [1,0], [0]   [2,0], [0]   [0,1], [0]   [1,1], [0]
[2,1], [0]    [0,2], [0]   [1,2], [0]   [2,2], [0]
[0,0], [1]    [1,0], [1]   [2,0], [1]   [0,1], [1]   [1,1], [1]
[2,1], [1]    [0,2], [1]   [1,2], [1]   [2,2], [1]
```

Phase 2 – Check

- drop non-solutions

```
check as bs = filter (isSolution as bs)
```

Example

- equation $x_1 + x_2 = 2\,y_1$, a $=$ 2, b $=$ 1, as $=$ [1,1], bs $=$ [2]

[2,0], [1]   [1,1], [1]

[0,2], [1]

Phase 2 – Check

- drop non-solutions

```
check as bs = filter (isSolution as bs)
```

Phase 3 – Minimize

-
```
minimize [] = []
minimize ((x,y):xs) =
    (x,y) : filter (x ++ y ≮v ) (minimize xs)
```

Example

- equation $x_1 + x_2 = 2y_1$, a $=$ 2, b $=$ 1, as $=$ [1,1], bs $=$ [2]

[2,0], [1]   [1,1], [1]

[0,2], [1]

Phase 2 – Check

- drop non-solutions

```
check as bs = filter (isSolution as bs)
```

Phase 3 – Minimize

- ```
  minimize [] = []
  minimize ((x,y):xs) =
      (x,y) : filter (x ++ y ≮v ) (minimize xs)
  ```

Remark

if $x <_{\mathsf{v}} y$ then $x <_{\mathsf{rlex}} y$

Example

- equation $x_1 + x_2 = 2\,y_1$, a $=$ 2, b $=$ 1, as $=$ [1,1], bs $=$ [2]

[2,0], [1] [1,1], [1]

[0,2], [1]
  ```

A Simple Algorithm

```
solutions as bs =
    minimize (check as bs (generate a b as bs))
  where
    a = maximum bs
    b = maximum as
```

A Simple Algorithm

```
solutions as bs =
    minimize (check as bs (generate a b as bs))
  where
    a = maximum bs
    b = maximum as
```

Lemma

algorithm is sound and complete, that is, `solutions` $a\ b = \mathcal{M}(a, b)$

A Simple Algorithm

```
solutions as bs =
    minimize (check as bs (generate a b as bs))
  where
    a = maximum bs
    b = maximum as
```

Lemma

algorithm is sound and complete, that is, `solutions` $a\ b = \mathcal{M}(a, b)$

Examples

a	b	#solutions	time (s)
[1,1]	[2]	3	0.001

A Simple Algorithm

```
solutions as bs =
    minimize (check as bs (generate a b as bs))
  where
    a = maximum bs
    b = maximum as
```

Lemma

algorithm is sound and complete, that is, `solutions` $a\ b = \mathcal{M}(a, b)$

Examples

a	b	#solutions	time (s)
[1,1]	[2]	3	0.001
[1,1]	[3]	4	0.001

A Simple Algorithm

```
solutions as bs =
    minimize (check as bs (generate a b as bs))
  where
    a = maximum bs
    b = maximum as
```

Lemma

algorithm is sound and complete, that is, `solutions` $a\ b = \mathcal{M}(a, b)$

Examples

a	b	#solutions	time (s)
[1,1]	[2]	3	0.001
[1,1]	[3]	4	0.001
[1,1,1]	[3]	10	0.001

A Simple Algorithm

```
solutions as bs =
    minimize (check as bs (generate a b as bs))
  where
    a = maximum bs
    b = maximum as
```

Lemma

algorithm is sound and complete, that is, `solutions` $a\ b = \mathcal{M}(a, b)$

Examples

a	b	#solutions	time (s)
[1,1]	[2]	3	0.001
[1,1]	[3]	4	0.001
[1,1,1]	[3]	10	0.001
[1,2,5]	[1,2,3,4]	39	0.2

A Simple Algorithm

```
solutions as bs =
    minimize (check as bs (generate a b as bs))
  where
    a = maximum bs
    b = maximum as
```

Lemma

algorithm is sound and complete, that is, `solutions` $a\ b = \mathcal{M}(a, b)$

Examples

a	b	#solutions	time (s)
[1,1]	[2]	3	0.001
[1,1]	[3]	4	0.001
[1,1,1]	[3]	10	0.001
[1,2,5]	[1,2,3,4]	39	0.2
[1,1,1,2,3]	[1,1,2,2]	44	0.2

A Simple Algorithm

```
solutions as bs =
    minimize (check as bs (generate a b as bs))
  where
    a = maximum bs
    b = maximum as
```

Lemma

algorithm is sound and complete, that is, `solutions` $a\ b = \mathcal{M}(a, b)$

Examples

a	b	#solutions	time (s)
[1,1]	[2]	3	0.001
[1,1]	[3]	4	0.001
[1,1,1]	[3]	10	0.001
[1,2,5]	[1,2,3,4]	39	0.2
[1,1,1,2,3]	[1,1,2,2]	44	0.2
[2,5,9]	[1,2,3,7,8]	119	85.5

A Simple Algorithm

```
solutions as bs =
    minimize (check as bs (generate a b as bs))
  where
    a = maximum bs
    b = maximum as
```

Lemma

algorithm is sound and complete, that is, `solutions` $a\ b = \mathcal{M}(a, b)$

Examples

a	b	#solutions	time (s)
[1,1]	[2]	3	0.001
[1,1]	[3]	4	0.001
[1,1,1]	[3]	10	0.001
[1,2,5]	[1,2,3,4]	39	0.2
[1,1,1,2,3]	[1,1,2,2]	44	0.2
[2,5,9]	[1,2,3,7,8]	119	85.5
[2,2,2,3,3,3]	[2,2,2,3,3,3]	138	125.4

A Simple Algorithm

```
solutions as bs =
    minimize (check as bs (generate a b as bs))
  where
    a = maximum bs
    b = maximum as
```

Lemma

algorithm is sound and complete, that is, `solutions` a $b = \mathcal{M}(a, b)$

Examples

a	b	#solutions	time (s)
[1,1]	[2]	3	0.001
[1,1]	[3]	4	0.001
[1,1,1]	[3]	10	0.001
[1,2,5]	[1,2,3,4]	39	0.2
[1,1,1,2,3]	[1,1,2,2]	44	0.2
[2,5,9]	[1,2,3,7,8]	119	85.5
[2,2,2,3,3,3]	[2,2,2,3,3,3]	138	125.4
[1,2,2,5,9]	[1,2,3,7,8]	timeout (after 20 min)	

Special Solutions

- given i and j, unique special solution is

$$0 \cdots \mathsf{lcm}(a_i, b_j)/a_i \cdots 0, 0 \cdots \mathsf{lcm}(a_i, b_j)/b_j \cdots 0$$

Special Solutions

- given i and j, unique special solution is

$$0 \cdots \mathsf{lcm}(a_i, b_j)/a_i \cdots 0, 0 \cdots \mathsf{lcm}(a_i, b_j)/b_j \cdots 0$$

- only 1 non-zero x_i and y_j

Special Solutions

- given i and j, unique special solution is

$$0 \cdots \mathsf{lcm}(a_i, b_j)/a_i \cdots 0, 0 \cdots \mathsf{lcm}(a_i, b_j)/b_j \cdots 0$$

- only 1 non-zero x_i and y_j
- all special solutions are minimal

Special Solutions

- given i and j, unique special solution is

$$0 \cdots \mathsf{lcm}(a_i, b_j)/a_i \cdots 0, 0 \cdots \mathsf{lcm}(a_i, b_j)/b_j \cdots 0$$

- only 1 non-zero x_i and y_j
- all special solutions are minimal

Example

- equation $x_1 + x_2 = 2y_1$
- special solutions

```
specialSolutions [1,1] [2] = [([2,0],[1]),([0,2],[1])]
```

Special Solutions

- given i and j, unique special solution is

$$0 \cdots \mathsf{lcm}(a_i, b_j)/a_i \cdots 0, 0 \cdots \mathsf{lcm}(a_i, b_j)/b_j \cdots 0$$

- only 1 non-zero x_i and y_j
- all special solutions are minimal
- it remains to compute non-special solutions (that is, those minimal solutions that are not special)

Example

- equation $x_1 + x_2 = 2\,y_1$
- special solutions

```
specialSolutions [1,1] [2] = [([2,0],[1]),([0,2],[1])]
```

Non-Special Solutions

Lemma (Huet)

if (x, y) is non-special minimal solution then

- $y_j \leq \texttt{maxy}\ x\ j$
- $\texttt{take}\ a\ k \bullet \texttt{take}\ x\ k \leq b \bullet y$
- $\texttt{take}\ b\ l \bullet \texttt{take}\ y\ l \leq a \bullet \texttt{map}\ (\texttt{maxx}\ (\texttt{take}\ y\ l))\ [0..m-1]$

where

$$\texttt{maxx}\ y\ i = \textbf{if}\ i < m \wedge D_i\ y \neq 0\ \textbf{then}\ \mathsf{min}(D_i\ y)\ \textbf{else}\ \mathsf{max}(b)$$

$$\texttt{maxy}\ x\ j = \textbf{if}\ j < n \wedge E_j\ x \neq 0\ \textbf{then}\ \mathsf{min}(E_j\ x)\ \textbf{else}\ \mathsf{max}(a)$$

$$D_i\ y = \{\mathsf{lcm}(a_i, b_j)/a_i - 1 \mid j < |y| \wedge y_j \geq \mathsf{lcm}(a_i, b_j)/b_j\}$$

$$E_j\ x = \{\mathsf{lcm}(a_i, b_j)/b_j - 1 \mid i < |x| \wedge x_i \geq \mathsf{lcm}(a_i, b_j)/a_i\}$$

Non-Special Solutions

Lemma (Huet)

if (x, y) is non-special minimal solution then

- $y_j \leq \texttt{maxy}\ x\ j$
- $\texttt{take}\ a\ k \bullet \texttt{take}\ x\ k \leq b \bullet y$
- $\texttt{take}\ b\ l \bullet \texttt{take}\ y\ l \leq a \bullet \texttt{map}\ (\texttt{maxx}\ (\texttt{take}\ y\ l))\ [0..m-1]$

where

$$\texttt{maxx}\ y\ i = \textbf{if}\ i < m \wedge D_i\ y \neq 0\ \textbf{then}\ \min(D_i\ y)\ \textbf{else}\ \max(b)$$

$$\texttt{maxy}\ x\ j = \textbf{if}\ j < n \wedge E_j\ x \neq 0\ \textbf{then}\ \min(E_j\ x)\ \textbf{else}\ \max(a)$$

$$D_i\ y = \{\mathrm{lcm}(a_i, b_j)/a_i - 1 \mid j < |y| \wedge y_j \geq \mathrm{lcm}(a_i, b_j)/b_j\}$$

$$E_j\ x = \{\mathrm{lcm}(a_i, b_j)/b_j - 1 \mid i < |x| \wedge x_i \geq \mathrm{lcm}(a_i, b_j)/a_i\}$$

Improved Bounds on Minimal Solutions

(Clausen and Fortenbacher)

if (x, y) is minimal solution then $x_i \leq \max^{\neq 0}\ y\ b$ and $y_j \leq \max^{\neq 0}\ x\ a$

Merging Generate and Check

- compute all vectors of length equal to `length cs` whose elements and "partial sums" satisfy `p`

```
incs p c i (xs,s) =
    if p (i:xs) t then (i:xs,t) : incs p c (i+1) (xs,s)
    else []
  where
    t = s + c*i

genCheck p [] = [([],0)]
genCheck p (c:cs) =
  concat (map (incs p c 0) (genCheck p cs))
```

Merging Generate and Check

- compute all vectors of length equal to `length cs` whose elements and "partial sums" satisfy `p`

```
incs p c i (xs,s) =
    if p (i:xs) t then (i:xs,t) : incs p c (i+1) (xs,s)
    else []
  where
    t = s + c*i

genCheck p [] = [([],0)]
genCheck p (c:cs) =
  concat (map (incs p c 0) (genCheck p cs))
```

- compute potential solutions within bounds

```
generateCheck as bs =
    tail [(x, y) | (y, _) <- genCheck q bs,
                   (x, _) <- genCheck (p y) as]
  where
    p ys (x:_) s = s <= bs `dp` ys && x <= maxne0 ys bs
    ...
```

An Improved Algorithm

- additional check phase

```
check' as bs = filter (\(xs, ys) ->
  all (<= maxne0 xs as) ys &&
  isSolution as bs xs ys &&
  all (\j -> ys !! j <= maxy xs j) [0..length bs - 1])
```

An Improved Algorithm

- additional check phase

```
check' as bs = filter (\(xs, ys) ->
  all (<= maxne0 xs as) ys &&
  isSolution as bs xs ys &&
  all (\j -> ys !! j <= maxy xs j) [0..length bs - 1])
```

- computing non-special solutions

```
nonSpecialSolutions as bs =
    minimize (check' as bs (generateCheck as bs))
```

An Improved Algorithm

- additional check phase

```
check' as bs = filter (\(xs, ys) ->
  all (<= maxne0 xs as) ys &&
  isSolution as bs xs ys &&
  all (\j -> ys !! j <= maxy xs j) [0..length bs - 1])
```

- computing non-special solutions

```
nonSpecialSolutions as bs =
    minimize (check' as bs (generateCheck as bs))
```

- the algorithm

```
solutions' as bs =
  specialSolutions as bs ++ nonSpecialSolutions as bs
```

An Improved Algorithm

- additional check phase

```
check' as bs = filter (\(xs, ys) ->
  all (<= maxne0 xs as) ys &&
  isSolution as bs xs ys &&
  all (\j -> ys !! j <= maxy xs j) [0..length bs - 1])
```

- computing non-special solutions

```
nonSpecialSolutions as bs =
    minimize (check' as bs (generateCheck as bs))
```

- the algorithm

```
solutions' as bs =
  specialSolutions as bs ++ nonSpecialSolutions as bs
```

Lemma

`solutions'` and `solutions` compute the same results

Example

```
generateCheck [1,1] [2]
```

```
nonSpecialSolutions [1,1] [2]
```

```
solutions' [1,1] [2]
```

Example

```
generateCheck [1,1] [2]
```

```
nonSpecialSolutions [1,1] [2]
```

```
solutions' [1,1] [2]
 = specialSolutions [1,1] [2] ++ nonSpecialSolutions ...
```

Example

```
generateCheck [1,1] [2]
```

```
nonSpecialSolutions [1,1] [2]
 = minimize (check' [1,1] [2] (generateCheck [1,1] [2]))
```

```
solutions' [1,1] [2]
 = specialSolutions [1,1] [2] ++ nonSpecialSolutions ...
```

Example

```
generateCheck [1,1] [2]
 = tail [(x,y) | y <- genCheck q [2],
                 x <- genCheck (p y) [1,1]]
```

```
nonSpecialSolutions [1,1] [2]
 = minimize (check' [1,1] [2] (generateCheck [1,1] [2]))
```

```
solutions' [1,1] [2]
 = specialSolutions [1,1] [2] ++ nonSpecialSolutions ...
```

Example

```
generateCheck [1,1] [2]
 = tail [(x,y) | y <- genCheck q [2],
                 x <- genCheck (p y) [1,1]]
 = tail ([(x,[0]) | x <- genCheck (p [0]) [1,1]] ++ [(x,[1])
```

```
nonSpecialSolutions [1,1] [2]
 = minimize (check' [1,1] [2] (generateCheck [1,1] [2]))
```

```
solutions' [1,1] [2]
 = specialSolutions [1,1] [2] ++ nonSpecialSolutions ...
```

Example

```
generateCheck [1,1] [2]
 = tail [(x,y) | y <- genCheck q [2],
                 x <- genCheck (p y) [1,1]]
 = tail ([(x,[0]) | x <- genCheck (p [0]) [1,1]] ++ [(x,[1])
 = tail ([(x,[0]) | x <- [[0,0]]] ++ [(x,[1]) | ...])
```

```
nonSpecialSolutions [1,1] [2]
 = minimize (check' [1,1] [2] (generateCheck [1,1] [2]))
```

```
solutions' [1,1] [2]
 = specialSolutions [1,1] [2] ++ nonSpecialSolutions ...
```

Example

```
generateCheck [1,1] [2]
 = tail [(x,y) | y <- genCheck q [2],
                 x <- genCheck (p y) [1,1]]
 = tail ([(x,[0]) | x <- genCheck (p [0]) [1,1]] ++ [(x,[1])
 = tail ([(x,[0]) | x <- [[0,0]]] ++ [(x,[1]) | ...])
 = tail (([0,0],[0]) : [(x,[1]) | ...])
```

```
nonSpecialSolutions [1,1] [2]
 = minimize (check' [1,1] [2] (generateCheck [1,1] [2]))
```

```
solutions' [1,1] [2]
 = specialSolutions [1,1] [2] ++ nonSpecialSolutions ...
```

Example

```
generateCheck [1,1] [2]
 = tail [(x,y) | y <- genCheck q [2],
                 x <- genCheck (p y) [1,1]]
 = tail ([(x,[0]) | x <- genCheck (p [0]) [1,1]] ++ [(x,[1])
 = tail ([(x,[0]) | x <- [[0,0]]] ++ [(x,[1]) | ...])
 = tail (([0,0],[0]) : [(x,[1]) | ...])
 = [(x,[1]) | x <- genCheck (p [1]) [1,1]]
```

```
nonSpecialSolutions [1,1] [2]
 = minimize (check' [1,1] [2] (generateCheck [1,1] [2]))
```

```
solutions' [1,1] [2]
 = specialSolutions [1,1] [2] ++ nonSpecialSolutions ...
```

Example

```
generateCheck [1,1] [2]
 = tail [(x,y) | y <- genCheck q [2],
                 x <- genCheck (p y) [1,1]]
 = tail ([(x,[0]) | x <- genCheck (p [0]) [1,1]] ++ [(x,[1])
 = tail ([(x,[0]) | x <- [[0,0]]] ++ [(x,[1]) | ...])
 = tail (([0,0],[0]) : [(x,[1]) | ...])
 = [(x,[1]) | x <- genCheck (p [1]) [1,1]]
 = [(x,[1]) | x <- [[0,0],[1,0],[2,0],[0,1],[1,1],[0,2]] ]
```

```
nonSpecialSolutions [1,1] [2]
 = minimize (check' [1,1] [2] (generateCheck [1,1] [2]))
```

```
solutions' [1,1] [2]
 = specialSolutions [1,1] [2] ++ nonSpecialSolutions ...
```

Example

```
generateCheck [1,1] [2]
 = tail [(x,y) | y <- genCheck q [2],
                 x <- genCheck (p y) [1,1]]
 = tail ([(x,[0]) | x <- genCheck (p [0]) [1,1]] ++ [(x,[1])
 = tail ([(x,[0]) | x <- [[0,0]]] ++ [(x,[1]) | ...])
 = tail (([0,0],[0]) : [(x,[1]) | ...])
 = [(x,[1]) | x <- genCheck (p [1]) [1,1]]
 = [(x,[1]) | x <- [[0,0],[1,0],[2,0],[0,1],[1,1],[0,2]] ]
 = [([0,0],[1]),([1,0],[1]),([2,0],[1]),([0,1],[1]),
    ([1,1],[1]),([0,2],[1])]

nonSpecialSolutions [1,1] [2]
 = minimize (check' [1,1] [2] (generateCheck [1,1] [2]))



solutions' [1,1] [2]
 = specialSolutions [1,1] [2] ++ nonSpecialSolutions ...
```

Example

```
generateCheck [1,1] [2]
 = tail [(x,y) | y <- genCheck q [2],
                 x <- genCheck (p y) [1,1]]
 = tail ([(x,[0]) | x <- genCheck (p [0]) [1,1]] ++ [(x,[1])
 = tail ([(x,[0]) | x <- [[0,0]]] ++ [(x,[1]) | ...])
 = tail (([0,0],[0]) : [(x,[1]) | ...])
 = [(x,[1]) | x <- genCheck (p [1]) [1,1]]
 = [(x,[1]) | x <- [[0,0],[1,0],[2,0],[0,1],[1,1],[0,2]] ]
 = [([0,0],[1]),([1,0],[1]),([2,0],[1]),([0,1],[1]),
    ([1,1],[1]),([0,2],[1])]

nonSpecialSolutions [1,1] [2]
 = minimize (check' [1,1] [2] (generateCheck [1,1] [2]))
 = minimize [([1,1],[1])]


solutions' [1,1] [2]
 = specialSolutions [1,1] [2] ++ nonSpecialSolutions ...
```

Example

```
generateCheck [1,1] [2]
 = tail [(x,y) | y <- genCheck q [2],
                 x <- genCheck (p y) [1,1]]
 = tail ([(x,[0]) | x <- genCheck (p [0]) [1,1]] ++ [(x,[1])
 = tail ([(x,[0]) | x <- [[0,0]]] ++ [(x,[1]) | ...])
 = tail (([0,0],[0]) : [(x,[1]) | ...])
 = [(x,[1]) | x <- genCheck (p [1]) [1,1]]
 = [(x,[1]) | x <- [[0,0],[1,0],[2,0],[0,1],[1,1],[0,2]] ]
 = [([0,0],[1]),([1,0],[1]),([2,0],[1]),([0,1],[1]),
    ([1,1],[1]),([0,2],[1])]

nonSpecialSolutions [1,1] [2]
 = minimize (check' [1,1] [2] (generateCheck [1,1] [2]))
 = minimize [([1,1],[1])]
 = [([1,1],[1])]

solutions' [1,1] [2]
 = specialSolutions [1,1] [2] ++ nonSpecialSolutions ...
```

Example

```
generateCheck [1,1] [2]
 = tail [(x,y) | y <- genCheck q [2],
                 x <- genCheck (p y) [1,1]]
 = tail ([(x,[0]) | x <- genCheck (p [0]) [1,1]] ++ [(x,[1])
 = tail ([(x,[0]) | x <- [[0,0]]] ++ [(x,[1]) | ...])
 = tail (([0,0],[0]) : [(x,[1]) | ...])
 = [(x,[1]) | x <- genCheck (p [1]) [1,1]]
 = [(x,[1]) | x <- [[0,0],[1,0],[2,0],[0,1],[1,1],[0,2]] ]
 = [([0,0],[1]),([1,0],[1]),([2,0],[1]),([0,1],[1]),
    ([1,1],[1]),([0,2],[1])]

nonSpecialSolutions [1,1] [2]
 = minimize (check' [1,1] [2] (generateCheck [1,1] [2]))
 = minimize [([1,1],[1])]
 = [([1,1],[1])]

solutions' [1,1] [2]
 = specialSolutions [1,1] [2] ++ nonSpecialSolutions ...
 = [([2,0],[1]),([0,2],[1])] ++ [([1,1],[1])]
```

Examples

a	b	#solutions	time (s)
[1,1]	[2]	3	0.001
[1,1]	[3]	4	0.001
[1,1,1]	[3]	10	0.001
[1,2,5]	[1,2,3,4]	39	0.1
[1,1,1,2,3]	[1,1,2,2]	44	0.1
[2,5,9]	[1,2,3,7,8]	119	85.5
[2,2,2,3,3,3]	[2,2,2,3,3,3]	138	125.4
[1,2,2,5,9]	[1,2,3,7,8]	timeout (after 20 min)	

Examples

a	b	#solutions	time (s)
[1,1]	[2]	3	0.001
[1,1]	[3]	4	0.001
[1,1,1]	[3]	10	0.001
[1,2,5]	[1,2,3,4]	39	0.05
[1,1,1,2,3]	[1,1,2,2]	44	0.01
[2,5,9]	[1,2,3,7,8]	119	8.6
[2,2,2,3,3,3]	[2,2,2,3,3,3]	138	0.06
[1,2,2,5,9]	[1,2,3,7,8]	345	517.4
[1,4,4,8,12]	[3,6,9,12,20]	232	67.4

Summary

- first formalization of HLDEs (we used Isabelle/HOL)

Summary

- first formalization of HLDEs (we used Isabelle/HOL)
- and of simple solver computing minimal complete sets of solutions

Summary

- first formalization of HLDEs (we used Isabelle/HOL)
- and of simple solver computing minimal complete sets of solutions
- clear separation of 3 phases: generate, check, and minimize

Summary

- first formalization of HLDEs (we used Isabelle/HOL)
- and of simple solver computing minimal complete sets of solutions
- clear separation of 3 phases: generate, check, and minimize
- which greatly simplifies proofs

Summary

- first formalization of HLDEs (we used Isabelle/HOL)
- and of simple solver computing minimal complete sets of solutions
- clear separation of 3 phases: generate, check, and minimize
- which greatly simplifies proofs
- basis for computing minimal complete sets of AC unifiers

Summary

- first formalization of HLDEs (we used Isabelle/HOL)
- and of simple solver computing minimal complete sets of solutions
- clear separation of 3 phases: generate, check, and minimize
- which greatly simplifies proofs
- basis for computing minimal complete sets of AC unifiers
- improved efficiency by partially merging generate and check phases