A Formalization of Convex Polyhedra based on the Simplex Method

Séminaire Francilien de Géométrie Algorithmique et Combinatoire

Xavier Allamigeon[1]   Ricardo D. Katz[2]

March 15th, 2018

[1] INRIA and Ecole polytechnique
[2] CIFACIS-CONICET

Motivation

Formal proving

Informal definition

Formal proving = "checking" proof using computers
… by implementing the proof in a **proof assistant**

Formal proving

Informal definition

Formal proving $=$ "checking" proof using computers
... by implementing the proof in a **proof assistant**

Remark

Formal proving $\neq$ proving mathematical statements **automatically**.

Formal proving

Informal definition

Formal proving $=$ “checking” proof using computers
… by implementing the proof in a **proof assistant**

Remark

Formal proving $\neq$ proving mathematical statements **automatically**.

Many proof assistants: Agda, Coq, Isabelle, HOL-Light, Lean, Mizar, PVS, etc
↳ underlying logic, proof kernel, automation, libraries, etc

Formal proving

Informal definition

Formal proving = "checking" proof using computers
... by implementing the proof in a **proof assistant**

Remark

Formal proving ≠ proving mathematical statements **automatically**.

Many proof assistants: Agda, Coq, Isabelle, HOL-Light, Lean, Mizar, PVS, etc
↳ underlying logic, proof kernel, automation, libraries, etc

Recent achievements

- completion of Kepler conjecture [Hales et al., 2017]
- Feit–Thompson theorem [Gonthier et al., 2013]
- a collection of 100 theorems, see `http://www.cs.ru.nl/~freek/100/`

Why formalizing convex polyhedra?

Convex polyhedra are universal objects:

Pure maths discrete mathematics, combinatorics, algebraic geometry, etc

Applied maths optimization, operations research, control theory, etc

CS computational geometry, software verification, compilation and program optimization, constraint solving, etc.

Why formalizing convex polyhedra?

Convex polyhedra are universal objects:

Pure maths discrete mathematics, combinatorics, algebraic geometry, etc

Applied maths optimization, operations research, control theory, etc

CS computational geometry, software verification, compilation and program optimization, constraint solving, etc.

Some reasons to formalize convex polyhedra

- increase the level of trust in polyhedral computation (and their critical applications)

Why formalizing convex polyhedra?

Convex polyhedra are universal objects:

Pure maths discrete mathematics, combinatorics, algebraic geometry, etc

Applied maths optimization, operations research, control theory, etc

CS computational geometry, software verification, compilation and program optimization, constraint solving, etc.

Some reasons to formalize convex polyhedra

- increase the level of trust in polyhedral computation (and their critical applications)
- get rid of flaws in complicated proofs

Why formalizing convex polyhedra?

Convex polyhedra are universal objects:

Pure maths discrete mathematics, combinatorics, algebraic geometry, etc

Applied maths optimization, operations research, control theory, etc

CS computational geometry, software verification, compilation and program optimization, constraint solving, etc.

Some reasons to formalize convex polyhedra

- increase the level of trust in polyhedral computation (and their critical applications)
- get rid of flaws in complicated proofs
- provide rigorous proof of theorems relying on informal computations ("formal experimental maths")

Our contribution

Our contribution

Summary of our contribution

First steps of the formalization of the theory of convex polyhedra, in Coq.

Our contribution

Summary of our contribution

First steps of the formalization of the theory of convex polyhedra, in Coq.

Main characteristics

It is carried out in an **effective** way:

1. relies on a complete implementation of the simplex method (correctness + termination)
2. basic predicates (emptiness, boundedness, etc) of polyhedra are defined by means of Coq programs, calling the simplex method

Our contribution

Summary of our contribution

First steps of the formalization of the theory of convex polyhedra, in Coq.

Main characteristics

It is carried out in an **effective** way:

1. relies on a complete implementation of the simplex method (correctness + termination)
2. basic predicates (emptiness, boundedness, etc) of polyhedra are defined by means of Coq programs, calling the simplex method

Outcome

- predicates naturally come with certificates

Our contribution

Summary of our contribution

First steps of the formalization of the theory of convex polyhedra, in Coq.

Main characteristics

It is carried out in an **effective** way:

1. relies on a complete implementation of the simplex method (correctness + termination)
2. basic predicates (emptiness, boundedness, etc) of polyhedra are defined by means of Coq programs, calling the simplex method

Outcome

- predicates naturally come with certificates

  Example: a polyhedron is empty iff there is a certificate of inconsistency of the defining system (Farkas Lemma)

Our contribution

Summary of our contribution

First steps of the formalization of the theory of convex polyhedra, in Coq.

Main characteristics

It is carried out in an **effective** way:

1. relies on a complete implementation of the simplex method (correctness + termination)
2. basic predicates (emptiness, boundedness, etc) of polyhedra are defined by means of Coq programs, calling the simplex method

Outcome

- predicates naturally come with certificates
- this easily provides several essential results on polyhedra (Farkas, Minkowski, strong duality Th., etc)

Our contribution (2)

Implementation

The development is gathered in the library Coq-Polyhedra:

`github.com/nhojem/Coq-Polyhedra`

Our contribution (2)

Implementation

The development is gathered in the library Coq-Polyhedra:

`github.com/nhojem/Coq-Polyhedra`

It is based on the Mathematical Components Library [Gonthier et al., 2016]:

Our contribution (2)

Implementation

The development is gathered in the library Coq-Polyhedra:

`github.com/nhojem/Coq-Polyhedra`

It is based on the Mathematical Components Library [Gonthier et al., 2016]:

- we extensively use its Boolean reflection methodology

predicates over polyhedra $\equiv$ Coq programs		usual statements $\equiv$ logical formula
`bool`	$\xleftrightarrow{\texttt{reflect}}$	`Prop`

Our contribution (2)

Implementation

The development is gathered in the library Coq-Polyhedra:

`github.com/nhojem/Coq-Polyhedra`

It is based on the Mathematical Components Library [Gonthier et al., 2016]:

- we extensively use its Boolean reflection methodology

predicates over polyhedra $\equiv$ Coq programs	`reflect`	usual statements $\equiv$ logical formula
`bool`	$\longleftrightarrow$	`Prop`

- we exploit some of its components (mainly linear algebra) to formalize the simplex method.

Related work

Existing formalizations of polyhedra

HOL-Light very complete formalization of convex polyhedra, including several important results [Harrison, 2013]

Isabelle implementation of a simplex-based satisfiability procedure [Spasić and Marić, 2012]

Goal: obtain a practical and executable code for SMT solving

Coq implementation of Fourier–Motzkin elimination on linear inequalities [Sakaguchi, 2016]

In comparison,

- our approach is effective, based on certificates
- we use the simplex method as a mathematical tool

Remark

Polyhedra are also used in formal proving as "informal backend".

A quick overview at Coq and Mathematical Components

The proof assistant Coq

Main features

- developed since $\sim$ 30 years, first implementation by Coquand and Huet
- written in OCaml
- relies on the **Calculus of Inductive Constructions** (CIC)
- GUIs: CoqIDE, Emacs/Proof General, etc

The proof assistant Coq

Main features

- developed since $\sim$ 30 years, first implementation by Coquand and Huet
- written in OCaml
- relies on the **Calculus of Inductive Constructions** (CIC)
- GUIs: CoqIDE, Emacs/Proof General, etc

CIC = very rich typed functional programming language,
well-suited for implementing mathematical objects and statements

Main features

- developed since $\sim$ 30 years, first implementation by Coquand and Huet
- written in OCaml
- relies on the **Calculus of Inductive Constructions** (CIC)
- GUIs: CoqIDE, Emacs/Proof General, etc

CIC = very rich typed functional programming language,
well-suited for implementing mathematical objects and statements

CIC is an intuitionistic (constructive) logic

- no excluded middle law $P \vee \neg P$
- no double negation elimination $P \Leftrightarrow \neg\neg P$, no reductio ad absurdum
- to show $\exists x.\, P(x)$, you need to **construct** an x such that $P(x)$ holds.

Digression on functional languages and types

Functional = functions are terms like any other (constants, variables, etc)

Digression on functional languages and types

Functional = functions are terms like any other (constants, variables, etc)

Notation

- `f x` stands for the application of `f` to `x` (compared with `f(x)` in most imperative languages);
- `fun x => t` is the function which maps `x` to `t`;
- **curryfied** form: `fun x => fun y => t`, compared with `fun (x,y) => t`

Digression on functional languages and types

Functional = functions are terms like any other (constants, variables, etc)

Notation

- `f x` stands for the application of `f` to `x` (compared with `f(x)` in most imperative languages);
- `fun x => t` is the function which maps `x` to `t`;
- **curryfied** form: `fun x => fun y => t`, compared with `fun (x,y) => t`

Example

- if `f` and `g` are two functions, `f g` stands for their composition.
- lots of **pattern matching**, loops implemented via **recursion**, etc

Digression on functional languages and types

Functional = functions are terms like any other (constants, variables, etc)

Notation

- `f x` stands for the application of `f` to `x` (compared with `f(x)` in most imperative languages);
- `fun x => t` is the function which maps `x` to `t`;
- **curryfied** form: `fun x => fun y => t`, compared with `fun (x,y) => t`

Example

- if `f` and `g` are two functions, `f g` stands for their composition.
- lots of **pattern matching**, loops implemented via **recursion**, etc

Types

Every term comes with a type, for instance:

- `nat`, `bool`
- `A -> B`: functions from `A` to `B`

Checking proof: Curry–Howard isomorphism

General correspondence principle between logical formulas and types

Formula	Type	
A	`A`	the type `A` lives in the sort `Prop`

Checking proof: Curry–Howard isomorphism

General correspondence principle between logical formulas and types

Formula	Type	
A	`A`	the type `A` lives in the sort `Prop`
proof x of A	`x:A`	the term `x` has type `A`

Checking proof: Curry–Howard isomorphism

General correspondence principle between logical formulas and types

Formula	Type	
A	`A`	the type `A` lives in the sort `Prop`
proof x of A	`x:A`	the term `x` has type `A`
$A \wedge B$	`A * B`	the type of pairs `(a,b)`, where `a:A`, `b:B`

General correspondence principle between logical formulas and types

Formula	Type	
A	`A`	the type `A` lives in the sort `Prop`
proof x of A	`x:A`	the term `x` has type `A`
$A \wedge B$	`A * B`	the type of pairs `(a,b)`, where `a:A`, `b:B`
$A \vee B$	`A + B`	"sum type", either `a:A` or `b:B`

Checking proof: Curry–Howard isomorphism

General correspondence principle between logical formulas and types

Formula	Type	
A	`A`	the type `A` lives in the sort `Prop`
proof x of A	`x:A`	the term `x` has type `A`
$A \wedge B$	`A * B`	the type of pairs `(a,b)`, where `a:A`, `b:B`
$A \vee B$	`A + B`	"sum type", either `a:A` or `b:B`
$A \implies B$	`A -> B`	function which returns a proof of B from a proof of A

Checking proof: Curry–Howard isomorphism

General correspondence principle between logical formulas and types

Formula	Type	
A	`A`	the type `A` lives in the sort `Prop`
proof x of A	`x:A`	the term `x` has type `A`
$A \wedge B$	`A * B`	the type of pairs `(a,b)`, where `a:A`, `b:B`
$A \vee B$	`A + B`	"sum type", either `a:A` or `b:B`
$A \implies B$	`A -> B`	function which returns a proof of B from a proof of A
$\forall x. A(x)$	$\Pi_{\mathtt{x}}$`A(x)`	dependent product

Checking proof: Curry–Howard isomorphism

General correspondence principle between logical formulas and types

Formula	Type	
A	`A`	the type `A` lives in the sort `Prop`
proof x of A	`x:A`	the term `x` has type `A`
$A \wedge B$	`A * B`	the type of pairs `(a,b)`, where `a:A`, `b:B`
$A \vee B$	`A + B`	"sum type", either `a:A` or `b:B`
$A \implies B$	`A -> B`	function which returns a proof of B
		from a proof of A
$\forall x. A(x)$	$\Pi_{\mathtt{x}}$`A(x)`	dependent product
False	`False`	empty type

Checking proof: Curry–Howard isomorphism

General correspondence principle between logical formulas and types

Formula	Type	
A	`A`	the type `A` lives in the sort `Prop`
proof x of A	`x:A`	the term `x` has type `A`
$A \wedge B$	`A * B`	the type of pairs `(a,b)`, where `a:A`, `b:B`
$A \vee B$	`A + B`	"sum type", either `a:A` or `b:B`
$A \implies B$	`A -> B`	function which returns a proof of B from a proof of A
$\forall x. A(x)$	Π_x`A(x)`	dependent product
False	`False`	empty type

Coq is a typechecker!

The kernel of Coq checks that the type of terms is correct w.r.t. a set of inference rules.

Checking proof: Curry–Howard isomorphism

General correspondence principle between logical formulas and types

Formula	Type	
A	`A`	the type `A` lives in the sort `Prop`
proof x of A	`x:A`	the term `x` has type `A`
$A \wedge B$	`A * B`	the type of pairs `(a,b)`, where `a:A`, `b:B`
$A \vee B$	`A + B`	"sum type", either `a:A` or `b:B`
$A \implies B$	`A -> B`	function which returns a proof of B from a proof of A
$\forall x. A(x)$	$\Pi_{\mathtt{x}}$`A(x)`	dependent product
False	`False`	empty type

Coq is a typechecker!

The kernel of Coq checks that the type of terms is correct w.r.t. a set of inference rules.

Example

$$\frac{\mathtt{f : A \to B} \qquad \mathtt{a : A}}{\mathtt{f\ a : B}} \equiv \text{if } A \implies B \text{ and } A \text{ hold, then } B \text{ holds}$$

The Mathematical Components library

Called MathComp for short.

The Mathematical Components library

Called MathComp for short.

- developed by Gonthier et al. for the formal proof of the Four Color Theorem and Feit–Thompson Theorem

The Mathematical Components library

Called MathComp for short.

- developed by Gonthier et al. for the formal proof of the Four Color Theorem and Feit–Thompson Theorem
- relies on **small scale reflection** principle: proof by computation rather than by logical derivation

The Mathematical Components library

Called MathComp for short.

- developed by Gonthier et al. for the formal proof of the Four Color Theorem and Feit–Thompson Theorem
- relies on **small scale reflection** principle: proof by computation rather than by logical derivation
- large **hierarchy** of mathematical objects: linear algebra, finite groups, algebraic numbers, representation theory, etc

The Mathematical Components library

Called MathComp for short.

- developed by Gonthier et al. for the formal proof of the Four Color Theorem and Feit–Thompson Theorem
- relies on **small scale reflection** principle: proof by computation rather than by logical derivation
- large **hierarchy** of mathematical objects: linear algebra, finite groups, algebraic numbers, representation theory, etc
- powerful set of **tactics**

The Mathematical Components library

Called MathComp for short.

- developed by Gonthier et al. for the formal proof of the Four Color Theorem and Feit–Thompson Theorem
- relies on **small scale reflection** principle: proof by computation rather than by logical derivation
- large **hierarchy** of mathematical objects: linear algebra, finite groups, algebraic numbers, representation theory, etc
- powerful set of **tactics**
- freely distributed, available at `https://github.com/math-comp/math-comp`

The Mathematical Components library

Called MathComp for short.

- developed by Gonthier et al. for the formal proof of the Four Color Theorem and Feit–Thompson Theorem
- relies on **small scale reflection** principle: proof by computation rather than by logical derivation
- large **hierarchy** of mathematical objects: linear algebra, finite groups, algebraic numbers, representation theory, etc
- powerful set of **tactics**
- freely distributed, available at `https://github.com/math-comp/math-comp`
- a book (Mahboubi and Tassi), under development `https://math-comp.github.io/mcb/`

Called MathComp for short.

- developed by Gonthier et al. for the formal proof of the Four Color Theorem and Feit–Thompson Theorem
- relies on **small scale reflection** principle: proof by computation rather than by logical derivation
- large **hierarchy** of mathematical objects: linear algebra, finite groups, algebraic numbers, representation theory, etc
- powerful set of **tactics**
- freely distributed, available at `https://github.com/math-comp/math-comp`
- a book (Mahboubi and Tassi), under development `https://math-comp.github.io/mcb/`

Examples!

Boolean reflection

Two different worlds:

- logical formulas of CIC, which live in the sort `Prop`
- Boolean algebra, i.e., the type `bool := true | false`.

Boolean reflection

Two different worlds:

- logical formulas of CIC, which live in the sort `Prop`
- Boolean algebra, i.e., the type `bool := true | false`.

`Prop`	`bool`
`True, False`	`true, false`
`A /\ B`	`a && b`
`A \/ B`	`a || b`
`~ A`	`~~ a`
`forall x, P x`	`[forall i, p i]`

Boolean reflection

Two different worlds:

- logical formulas of CIC, which live in the sort `Prop`
- Boolean algebra, i.e., the type `bool := true | false`.

`Prop`	`bool`
`True, False`	`true, false`
`A /\ B`	`a && b`
`A \/ B`	`a || b`
`~ A`	`~~ a`
`forall x, P x`	`[forall i, p i]`

Pros of `bool`

`bool` behaves like the **classical logic**!

$\Longrightarrow$ overcome the intuitionistic restriction of CIC
allows case analysis, reductio ad absurdum, etc

Boolean reflection (2)

But using of `bool`

- requires to implement a decision procedure in the CIC:
  Booleans are **computed** by Coq functions

Boolean reflection (2)

But using of `bool`

- requires to implement a decision procedure in the CIC: Booleans are **computed** by Coq functions

Example

Decidable equality over naturals (type `nat`)

```
Fixpoint eqn m n {struct m} :=
  match m, n with
  | 0, 0 => true
  | m'.+1, n'.+1 => eqn m' n'
  | _, _ => false
  end.
```

```
About eqn.
  eqn : nat -> nat -> bool
```

Boolean reflection (2)

But using of `bool`

- requires to implement a decision procedure in the CIC:
  Booleans are **computed** by Coq functions
- is less convenient to manipulate the logical part of formulas.

Boolean reflection (2)

But using of `bool`

- requires to implement a decision procedure in the CIC:
  Booleans are **computed** by Coq functions
- is less convenient to manipulate the logical part of formulas.

$\Longrightarrow$ MathComp combines the best of the two worlds `Prop` and `bool`, via the **Boolean reflection methodology**:

Boolean reflection (2)

But using of `bool`

- requires to implement a decision procedure in the CIC: Booleans are **computed** by Coq functions
- is less convenient to manipulate the logical part of formulas.

$\Longrightarrow$ MathComp combines the best of the two worlds `Prop` and `bool`, via the **Boolean reflection methodology**:

Reflection predicate

`reflect P b` essentially means that `P:Prop` and `b:bool` are equivalent:

- either `P` holds and `b = true`,
- or `~ P` holds and `b = false`.

Boolean reflection (2)

But using of `bool`

- requires to implement a decision procedure in the CIC:
  Booleans are **computed** by Coq functions
- is less convenient to manipulate the logical part of formulas.

$\Longrightarrow$ MathComp combines the best of the two worlds `Prop` and `bool`, via the **Boolean reflection methodology**:

Reflection predicate

`reflect P b` essentially means that `P:Prop` and `b:bool` are equivalent:

- either `P` holds and `b = true`,
- or `~ P` holds and `b = false`.

+ MathComp provides **reflection views** to pass from `bool` to `Prop`, and vice versa.

Examples!

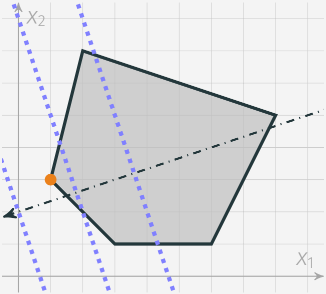
Formalizing the simplex method

Linear programming

$$\begin{aligned} \text{minimize} \quad & \langle c, x \rangle \\ \text{subject to} \quad & Ax \geqslant b\,,\ x \in \mathbb{R}^n \end{aligned}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, and $\langle c, x \rangle := \sum_{i=1}^n c_i x_i$.

Example



$$\begin{aligned} \text{minimize} \quad & 3x_1 + x_2 \\ \text{subject to} \quad & x_1 + x_2 \geqslant 4 \\ & -x_1 - 3x_2 \geqslant -23 \\ & 4x_1 - x_2 \geqslant 1 \\ & -2x_1 + x_2 \geqslant -11 \\ & x_2 \geqslant 1 \end{aligned}$$

The purpose of the simplex method

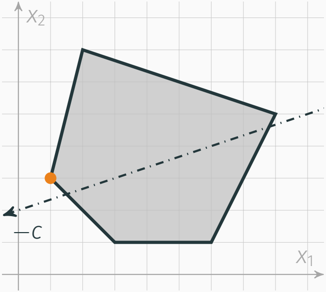
Linear programming

$$\begin{aligned} \text{minimize} \quad & \langle c, x \rangle \\ \text{subject to} \quad & Ax \geqslant b\,,\ x \in \mathbb{R}^n \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, and $\langle c, x \rangle := \sum_{i=1}^{n} c_i x_i$.

The **value** of the linear program can be

- finite (optimal point)



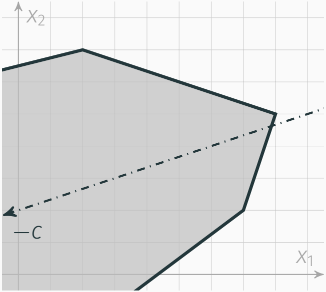
Linear programming

$$\begin{aligned} \text{minimize} \quad & \langle c, x \rangle \\ \text{subject to} \quad & Ax \geqslant b\,,\ x \in \mathbb{R}^n \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, and $\langle c, x \rangle := \sum_{i=1}^{n} c_i x_i$.

The **value** of the linear program can be

- finite (optimal point)
- equal to $-\infty$ (no lower bound)
  $\Longrightarrow$ the LP is **unbounded**



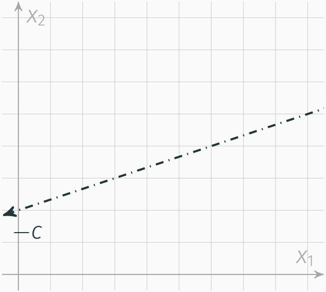
Linear programming

$$\begin{aligned} \text{minimize} \quad & \langle c, x \rangle \\ \text{subject to} \quad & Ax \geqslant b\,,\ x \in \mathbb{R}^n \end{aligned}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, and $\langle c, x \rangle := \sum_{i=1}^n c_i x_i$.

The **value** of the linear program can be

- finite (optimal point)
- equal to $-\infty$ (no lower bound)
  $\Longrightarrow$ the LP is **unbounded**
- equal to $+\infty$ (empty feasible set)
  $\Longrightarrow$ the LP is **infeasible**



The purpose of the simplex method (2)

The simplex method can be thought of as a **decision procedure**.

A **linear program**

$$\begin{array}{rl} \text{minimize} & \langle c, x\rangle \\ \text{subject to} & Ax \geqslant b\,,\ x \in \mathbb{R}^n \end{array}$$

The **dual** LP

$$\begin{array}{rl} \text{maximize} & \langle b, u\rangle \\ \text{subject to} & A^T u = c\,,\ u \geqslant 0\,,\ u \in \mathbb{R}^m \end{array}$$

The purpose of the simplex method (2)

The simplex method can be thought of as a **decision procedure**.

A **linear program**	The **dual** LP
minimize $\langle c, x\rangle$ subject to $Ax \geqslant b\,,\ x \in \mathbb{R}^n$	maximize $\langle b, u\rangle$ subject to $A^T u = c\,,\ u \geqslant 0\,,\ u \in \mathbb{R}^m$

Theorem (Strong duality)

If one of the two LPs is feasible, then they have the ***same optimal value****.*

In addition, when both are feasible, the optimal value is simultaneously attained by a primal feasible point x^* and a dual feasible point u^*.

The purpose of the simplex method (2)

The simplex method can be thought of as a **decision procedure**.

A **linear program**	The **dual** LP
minimize $\langle c, x\rangle$ subject to $Ax \geqslant b\,,\ x \in \mathbb{R}^n$	maximize $\langle b, u\rangle$ subject to $A^T u = c\,,\ u \geqslant 0\,,\ u \in \mathbb{R}^m$

Theorem (Strong duality)

*If one of the two LPs is feasible, then they have the **same optimal value**. (...)*

Corollary (Farkas Lemma)

The polyhedron $\{x \in \mathbb{R}^n : Ax \geqslant b\}$ is empty if and only if the value of the following LP is $+\infty$:

$$\textit{maximize} \quad \langle b, u\rangle \quad \textit{subject to} \quad A^T u = 0\,,\ u \geqslant 0\,,\ u \in \mathbb{R}^m$$

The purpose of the simplex method (2)

The simplex method can be thought of as a **decision procedure**.

A **linear program**	The **dual** LP
minimize $\langle c, x\rangle$ subject to $Ax \geqslant b\,,\ x \in \mathbb{R}^n$	maximize $\langle b, u\rangle$ subject to $A^T u = c\,,\ u \geqslant 0\,,\ u \in \mathbb{R}^m$

Theorem (Strong duality)

If one of the two LPs is feasible, then they have the ***same optimal value****. (...)*

Corollary (Farkas Lemma)

The polyhedron $\{x \in \mathbb{R}^n : Ax \geqslant b\}$ is empty if and only if the value of the following LP is $+\infty$:

$$\textit{maximize} \quad \langle b, u\rangle \quad \textit{subject to} \quad A^T u = 0\,,\ u \geqslant 0\,,\ u \in \mathbb{R}^m$$

Fit the Boolean reflection framework

Emptiness of polyhedra can be defined as a **Boolean predicate**, relying on the simplex method.

Linear programming

$$\begin{array}{rl} \text{minimize} & \langle c, x \rangle \\ \text{subject to} & Ax \geqslant b\,,\ x \in \mathbb{R}^n \end{array}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, and $\langle c, x \rangle := \sum_{i=1}^n c_i x_i$.

Global variables:

```
Variable A: 'M_(m,n). (* matrix of size m*n *)
Variable b: 'cV_m. (* col. vector of size m *)
Variable c: 'cV_n. (* col. vector of size n *)
```

The purpose of the simplex method (3)

Linear programming

$$\begin{aligned} \text{minimize} \quad & \langle c, x \rangle \\ \text{subject to} \quad & Ax \geqslant b\,,\ x \in \mathbb{R}^n \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, and $\langle c, x \rangle := \sum_{i=1}^{n} c_i x_i$.

Global variables:

```
Variable A: 'M_(m,n). (* matrix of size m*n *)
Variable b: 'cV_m. (* col. vector of size m *)
Variable c: 'cV_n. (* col. vector of size n *)
```

The feasible set is formalized via a Boolean predicate

```
Definition polyhedron A b := [pred x:'cV_n | (A *m x) >=m b].
```

- `*m` is the matrix product
- `y >=m z` is a notation for `[forall i, y i 0 >= z i 0]`

$\Longrightarrow$ `x \in polyhedron A b` reduces to `A *m x >=m b`.

The purpose of the simplex method (3)

Linear programming

$$\begin{aligned} \text{minimize} \quad & \langle c, x \rangle \\ \text{subject to} \quad & Ax \geqslant b\,,\ x \in \mathbb{R}^n \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, and $\langle c, x \rangle := \sum_{i=1}^{n} c_i x_i$.

Global variables:

```
Variable A: 'M_(m,n). (* matrix of size m*n *)
Variable b: 'cV_m. (* col. vector of size m *)
Variable c: 'cV_n. (* col. vector of size n *)
```

The feasible set is formalized via a Boolean predicate

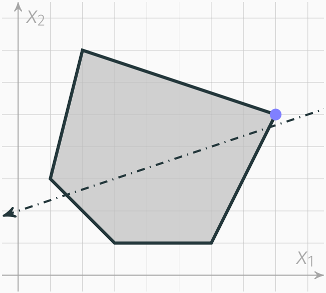
```
Definition polyhedron A b := [pred x:'cV_n | (A *m x) >=m b].
```

- `*m` is the matrix product
- `y >=m z` is a notation for `[forall i, y i 0 >= z i 0]`

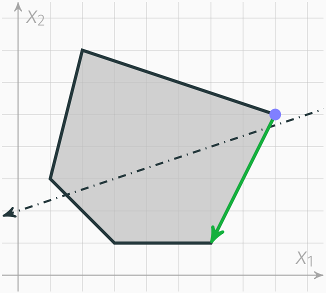
$\Longrightarrow$ `x \in polyhedron A b` reduces to `A *m x >=m b`.

Objective function $\equiv$ `'[c,x]`, notation for `\sum_(i < n) c_i * x_i`.

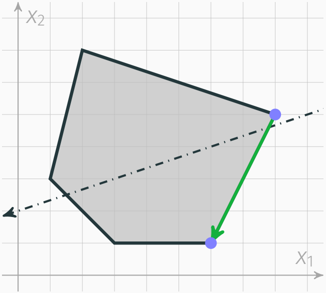
1. starting from an initial vertex



1. starting from an initial vertex
2. iterate over the vertex-edge graph while decreasing the objective function

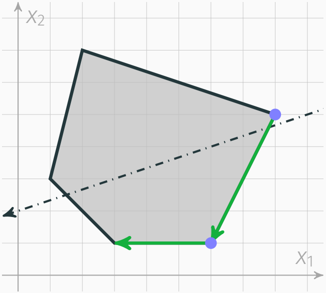


1. starting from an initial vertex
2. iterate over the vertex-edge graph while decreasing the objective function

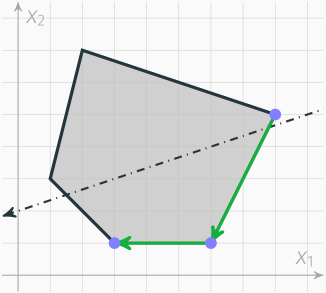


Geometric interpretation of the simplex method

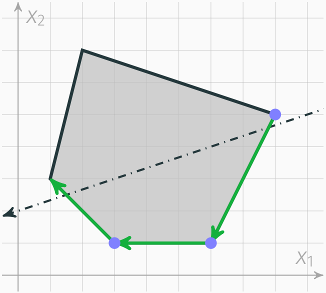
1. starting from an initial vertex
2. iterate over the vertex-edge graph
   while decreasing the objective function



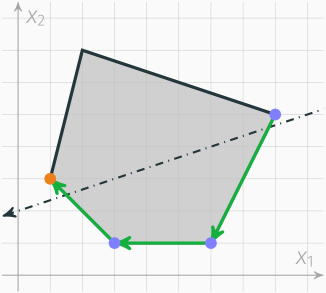
1. starting from an initial vertex
2. iterate over the vertex-edge graph
   while decreasing the objective function



1. starting from an initial vertex
2. iterate over the vertex-edge graph
   while decreasing the objective function



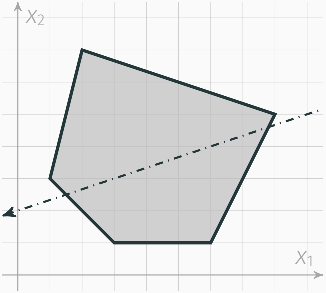
1. starting from an initial vertex
2. iterate over the vertex-edge graph
   while decreasing the objective function
3. up to finding an optimal vertex



Geometric interpretation of the simplex method

1. starting from an initial vertex
2. iterate over the vertex-edge graph while decreasing the objective function
3. up to finding an optimal vertex

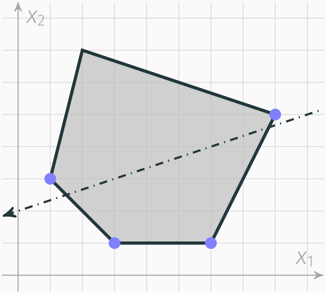
Three ingredients:



1. starting from an initial vertex
2. iterate over the vertex-edge graph
   while decreasing the objective function
3. up to finding an optimal vertex

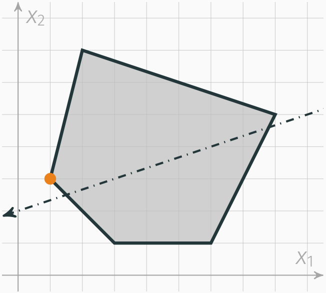
Three ingredients:

- bases encode vertices



Geometric interpretation of the simplex method

1. starting from an initial vertex
2. iterate over the vertex-edge graph while decreasing the objective function
3. up to finding an optimal vertex



Three ingredients:

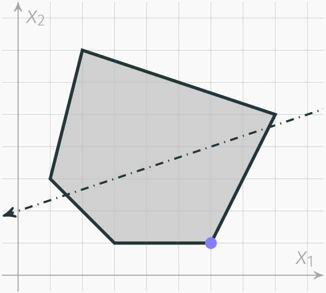
- bases encode vertices
- reduced costs determine optimality

Geometric interpretation of the simplex method

1. starting from an initial vertex
2. iterate over the vertex-edge graph while decreasing the objective function
3. up to finding an optimal vertex

Three ingredients:

- bases encode vertices
- reduced costs determine optimality
- pivoting switches from a vertex to another

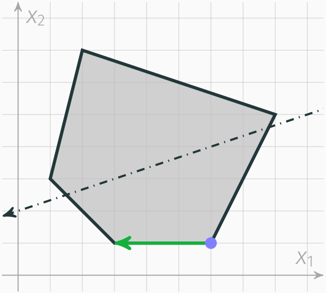


Geometric interpretation of the simplex method

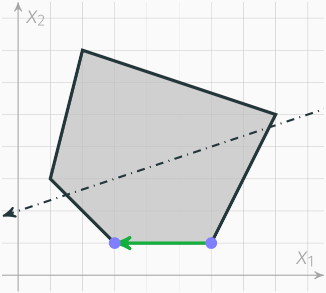
1. starting from an initial vertex
2. iterate over the vertex-edge graph while decreasing the objective function
3. up to finding an optimal vertex

Three ingredients:

- bases encode vertices
- reduced costs determine optimality
- pivoting switches from a vertex to another



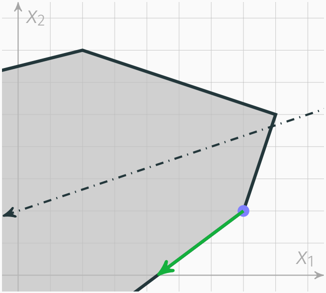
1. starting from an initial vertex
2. iterate over the vertex-edge graph while decreasing the objective function
3. up to finding an optimal vertex



Three ingredients:

- bases encode vertices
- reduced costs determine optimality
- pivoting switches from a vertex to another

1. starting from an initial vertex
2. iterate over the vertex-edge graph
   while decreasing the objective function
3. up to finding an optimal vertex



Three ingredients:

- bases encode vertices
- reduced costs determine optimality
- pivoting switches from a vertex to another
  or determines if the LP is unbounded

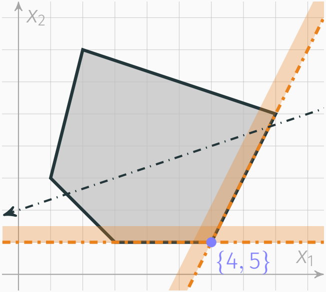
Definition

A **basis** is a subset $\mathcal{I} \subset \{1, \ldots, m\}$ of cardinality n such that the system

$$A_i x = b_i \,, \quad i \in \mathcal{I}$$

has a unique solution, called the **basic point**.

The basis is **feasible** when the basic point belongs to the polyhedron.



$$\begin{array}{lrl}
\text{minimize} & 3x_1 + x_2 & \\
\text{subject to} & x_1 + x_2 \geqslant 4 & ① \\
& -x_1 - 3x_2 \geqslant -23 & ② \\
& 4x_1 - x_2 \geqslant 1 & ③ \\
& -2x_1 + x_2 \geqslant -11 & ④ \\
& x_2 \geqslant 1 & ⑤
\end{array}$$

Bases

Definition

A **basis** is a subset $\mathcal{I} \subset \{1, \ldots, m\}$ of cardinality n such that the system

$$A_i x = b_i \,, \quad i \in \mathcal{I}$$

has a unique solution, called the **basic point**.

The basis is **feasible** when the basic point belongs to the polyhedron.

Bases are formalized via three layers of types:

Bases

Definition

A **basis** is a subset $\mathcal{I} \subset \{1, \dots, m\}$ of cardinality n such that the system

$$A_i x = b_i \,, \quad i \in \mathcal{I}$$

has a unique solution, called the **basic point**.

The basis is **feasible** when the basic point belongs to the polyhedron.

Bases are formalized via three layers of types:

```
Inductive prebasis := Prebasis (I: {set 'I_m}) of (#|I| == n).
```

Definition

A **basis** is a subset $\mathcal{I} \subset \{1, \ldots, m\}$ of cardinality n such that the system

$$A_i x = b_i \,, \quad i \in \mathcal{I}$$

has a unique solution, called the **basic point**.

The basis is **feasible** when the basic point belongs to the polyhedron.

Bases are formalized via three layers of types:

```
Inductive prebasis := Prebasis (I: {set 'I_m}) of (#|I| == n).
```

```
Inductive basis :=
  Basis (I:prebasis) of row_free (row_submx A I).
```

= submatrix $A_{\mathcal{I}}$

Bases

Definition

A **basis** is a subset $\mathcal{I} \subset \{1, \ldots, m\}$ of cardinality n such that the system

$$A_i x = b_i \,, \quad i \in \mathcal{I}$$

has a unique solution, called the **basic point**.

The basis is **feasible** when the basic point belongs to the polyhedron.

Bases are formalized via three layers of types:

```
Inductive prebasis := Prebasis (I: {set 'I_m}) of (#|I| == n).

Inductive basis :=
  Basis (I:prebasis) of row_free (row_submx A I).

Inductive feasible_basis :=
  FeasibleBasis (I:basis) of point_of_basis I \in polyhedron A b
```

where we have defined:

```
Definition point_of_basis (I:basis) :=
  (qinvmx [...] (row_submx A I)) *m (row_submx b I).
```

Reduced costs: optimality certificate

```
Variable I : feasible_basis.
```

Reduced costs: optimality certificate

```
Variable I : feasible_basis.
```

Definition

The *reduced cost vector* at basis $\mathcal{I}$ is defined as the unique solution $u \in \mathbb{R}^{\mathcal{I}}$ of the system

$$(A_{\mathcal{I}})^T u = c\,.$$

Reduced costs: optimality certificate

```
Variable I : feasible_basis.
```

Definition

The *reduced cost vector* at basis $\mathcal{I}$ is defined as the unique solution $u \in \mathbb{R}^{\mathcal{I}}$ of the system

$$(A_{\mathcal{I}})^T u = c\,.$$

In Coq, this simply writes as:

```
Definition reduced_cost :=
  (qinvmx [...] (row_submx A I))^T *m c.
```

Reduced costs: optimality certificate

```
Variable I : feasible_basis.
```

Definition

The *reduced cost vector* at basis $\mathcal{I}$ is defined as the unique solution $u \in \mathbb{R}^{\mathcal{I}}$ of the system

$$(A_{\mathcal{I}})^T u = c\,.$$

In COQ, this simply writes as:

```
Definition reduced_cost :=
  (qinvmx [...] (row_submx A I))^T *m c.
```

```
Lemma optimality_certificate :
  reduced_cost >=m 0 ->
    forall x, x \in polyhedron A b ->
      '[c, point_of_basis I] <= '[c, x].
```

Reduced costs: optimality certificate

```
Variable I : feasible_basis.
```

Definition

The *reduced cost vector* at basis $\mathcal{I}$ is defined as the unique solution $u \in \mathbb{R}^{\mathcal{I}}$ of the system

$$(A_{\mathcal{I}})^T u = c\,.$$

In Coq, this simply writes as:

```
Definition reduced_cost :=
  (qinvmx [...] (row_submx A I))^T *m c.
```

```
Lemma optimality_certificate :
  reduced_cost >=m 0 ->
    forall x, x \in polyhedron A b ->
      '[c, point_of_basis I] <= '[c, x].
```

Reduced costs: optimality certificate

```
Variable I : feasible_basis.
```

Definition

The *reduced cost vector* at basis $\mathcal{I}$ is defined as the unique solution $u \in \mathbb{R}^{\mathcal{I}}$ of the system

$$(A_{\mathcal{I}})^T u = c\,.$$

In Coq, this simply writes as:

```
Definition reduced_cost :=
  (qinvmx [...] (row_submx A I))^T *m c.
```

```
Lemma optimality_certificate :
  reduced_cost >=m 0 ->
    forall x, x \in polyhedron A b ->
      '[c, point_of_basis I] <= '[c, x].
```

Proof sketch

$$\langle c, x \rangle = \langle u, A_{\mathcal{I}} x \rangle = \langle \underbrace{u}_{\geqslant 0}, \underbrace{A_{\mathcal{I}} x - b_{\mathcal{I}}}_{\leqslant 0} \rangle + \langle u, b_{\mathcal{I}} \rangle$$

Reduced costs: optimality certificate

```
Variable I : feasible_basis.
```

Definition

The *reduced cost vector* at basis $\mathcal{I}$ is defined as the unique solution $u \in \mathbb{R}^{\mathcal{I}}$ of the system

$$(A_{\mathcal{I}})^T u = c\,.$$

In COQ, this simply writes as:

```
Definition reduced_cost :=
  (qinvmx [...] (row_submx A I))^T *m c.
```

```
Lemma optimality_certificate :
  reduced_cost >=m 0 ->
    forall x, x \in polyhedron A b ->
      '[c, point_of_basis I] <= '[c, x].
```

Proof sketch

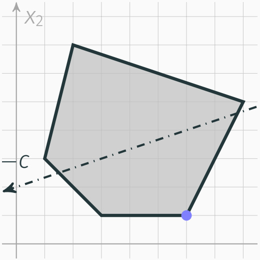
$$\langle c, x\rangle = \langle u, A_{\mathcal{I}} x\rangle = \langle \underbrace{u}_{\geqslant 0}, \underbrace{A_{\mathcal{I}} x - b_{\mathcal{I}}}_{\leqslant 0}\rangle + \langle u, b_{\mathcal{I}}\rangle \leqslant \langle u, b_{\mathcal{I}}\rangle$$

$\square$

Pivoting

If the reduced cost vector has some negative entries:

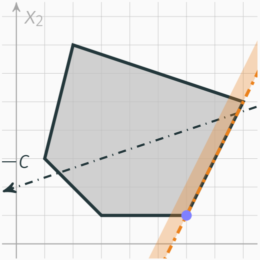
```
Variable i : 'I_#|I|.
Hypothesis [...] : reduced_cost i 0 < 0.
```



Pivoting

If the reduced cost vector has some negative entries:

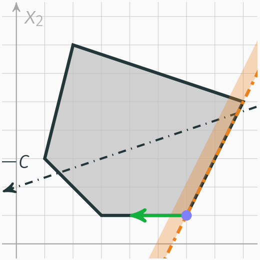
```
Variable i : 'I_#|I|.
Hypothesis [...] : reduced_cost i 0 < 0.
```



Pivoting

If the reduced cost vector has some negative entries:

```
Variable i : 'I_#|I|.
Hypothesis [...] : reduced_cost i 0 < 0.
```



we can build a direction vector which

- follows an incident edge
- decreases the objective function

Pivoting

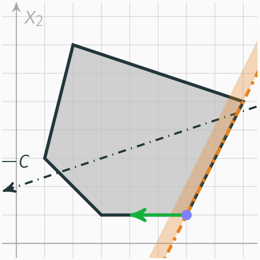
If the reduced cost vector has some negative entries:

```
Variable i : 'I_#|I|.
Hypothesis [...] : reduced_cost i 0 < 0.
```

we can build a direction vector which

- follows an incident edge
- decreases the objective function

```
Definition direction :=
  let: ei := (delta_mx i 0) in
  (qinvmx [...] (row_submx A I)) *m ei.
Lemma direction_improvement :
  '[c, direction] < 0.
```



Pivoting

If the reduced cost vector has some negative entries:

```
Variable i : 'I_#|I|.
Hypothesis [...] : reduced_cost i 0 < 0.
```

we can build a direction vector which

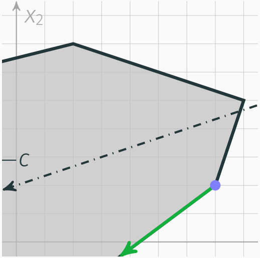
- follows an incident edge
- decreases the objective function

At this stage, two possibilities:

Pivoting

If the reduced cost vector has some negative entries:

```
Variable i : 'I_#|I|.
Hypothesis [...] : reduced_cost i 0 < 0.
```



we can build a direction vector which

- follows an incident edge
- decreases the objective function

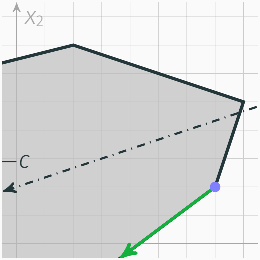
At this stage, two possibilities:

- the direction is *feasible*: `(A *m direction) >=m 0`
  i.e., the halfline is contained in the polyhedron
  $\Longrightarrow$ the LP is unbounded

Pivoting

If the reduced cost vector has some negative entries:

```
Variable i : 'I_#|I|.
Hypothesis [...] : reduced_cost i 0 < 0.
```



we can build a direction vector which

- follows an incident edge
- decreases the objective function

At this stage, two possibilities:

- the direction is *feasible*: `(A *m direction) >=m 0` i.e., the halfline is contained in the polyhedron
  $\Longrightarrow$ the LP is unbounded

```
Lemma unbounded_certificate : (A *m direction) >=m 0 ->
  forall M, exists x, (x \in polyhedron A b) /\ ('[c,x] < M)
```

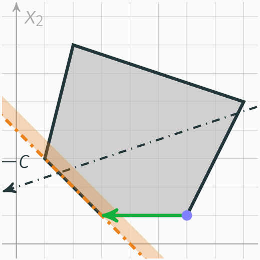
Pivoting

If the reduced cost vector has some negative entries:

```
Variable i : 'I_#|I|.
Hypothesis [...] : reduced_cost i 0 < 0.
```

we can build a direction vector which

- follows an incident edge
- decreases the objective function



At this stage, two possibilities:

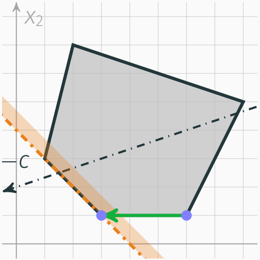
- the direction is *feasible*: `(A *m direction) >=m 0`
  $\Longrightarrow$ the LP is unbounded
- or, the halfline hits the boundary of a new halfspace

```
Definition new_halfspace := [...]
```

Pivoting

If the reduced cost vector has some negative entries:

```
Variable i : 'I_#|I|.
Hypothesis [...] : reduced_cost i 0 < 0.
```



we can build a direction vector which

- follows an incident edge
- decreases the objective function

At this stage, two possibilities:

- the direction is *feasible*: `(A *m direction) >=m 0`

  $\Longrightarrow$ the LP is unbounded

- or, the halfline hits the boundary of a new halfspace

```
Definition new_halfspace := [...]
```

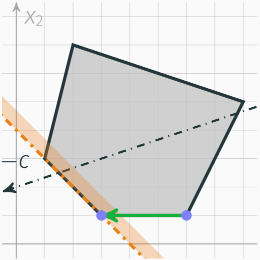
$\Longrightarrow$ the index `new_halfspace` is used to build the next basis:

```
Definition next_I :=
  new_halfspace |: (I :\ (enum_val [...] i)).
```

Pivoting

If the reduced cost vector has some negative entries:

```
Variable i : 'I_#|I|.
Hypothesis [...] : reduced_cost i 0 < 0.
```



we can build a direction vector which

- follows an incident edge
- decreases the objective function

At this stage, two possibilities:

- the direction is *feasible*: `(A *m direction) >=m 0`

  $\Longrightarrow$ the LP is unbounded

- or, the halfline hits the boundary of a new halfspace

```
Definition new_halfspace := [...]
```

$\Longrightarrow$ the index `new_halfspace` is used to build the next basis:

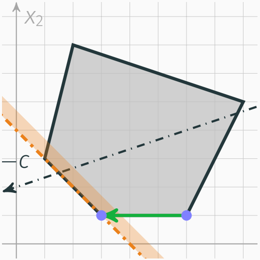
```
Definition next_I :=
  new_halfspace |: (I :\ (enum_val [...] i)).
```

= removes `i` from `I`

Pivoting

If the reduced cost vector has some negative entries:

```
Variable i : 'I_#|I|.
Hypothesis [...] : reduced_cost i 0 < 0.
```



we can build a direction vector which

- follows an incident edge
- decreases the objective function

At this stage, two possibilities:

- the direction is *feasible*: `(A *m direction) >=m 0`
  $\implies$ the LP is unbounded
- or, the halfline hits the boundary of a new halfspace

```
Definition new_halfspace := [...]
```

$\implies$ the index `new_halfspace` is used to build the next basis:

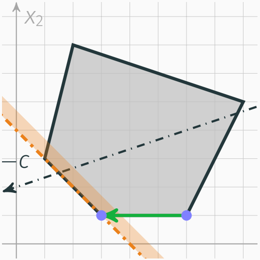
```
Definition next_I :=
  new_halfspace |: (I :\ (enum_val [...] i)).
```

= adds `new_halfspace`

Pivoting

If the reduced cost vector has some negative entries:

```
Variable i : 'I_#|I|.
Hypothesis [...] : reduced_cost i 0 < 0.
```



we can build a direction vector which

- follows an incident edge
- decreases the objective function

At this stage, two possibilities:

- the direction is *feasible*: `(A *m direction) >=m 0`

  $\Longrightarrow$ the LP is unbounded

- or, the halfline hits the boundary of a new halfspace

```
Definition new_halfspace := [...]
```

$\Longrightarrow$ the index `new_halfspace` is used to build the next basis:

```
Definition next_I :=
  new_halfspace |: (I :\ (enum_val [...] i)).
Fact [...]:
  '[c, point_of_basis next_I] <= '[c, point_of_basis I].
```

Termination

```
Fact [...]:
  '[c, point_of_basis next_I] <= '[c, point_of_basis I].
```

Termination

```
Fact [...]:
  '[c, point_of_basis next_I] < '[c, point_of_basis I].
```

If at every step the inequality is **strict**, termination follows from the fact that there are finitely many bases:

Termination

```
Fact [...]:
  '[c, point_of_basis next_I] < '[c, point_of_basis I].
```

If at every step the inequality is **strict**, termination follows from the fact that there are finitely many bases:

```
Definition basis_height I :=
  #|[ set J: feasible_bases |
     '[c, point_of_basis I] > '[c, point_of_basis J] ]|.
Function simplex_phase2 I {measure basis_height I} := [...].
```

Termination

```
Fact [...]:
  '[c, point_of_basis next_I] < '[c, point_of_basis I].
```

If at every step the inequality is **strict**, termination follows from the fact that there are finitely many bases:

```
Definition basis_height I :=
  #|[ set J: feasible_bases |
     '[c, point_of_basis I] > '[c, point_of_basis J] ]|.
Function simplex_phase2 I {measure basis_height I} := [...].
```

The inequality may not be strict because of **degenerate bases**

= several bases correspond to the same basic point.

Termination

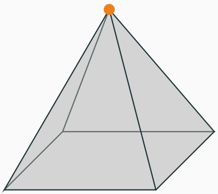
```
Fact [...]:
  '[c, point_of_basis next_I] <= '[c, point_of_basis I].
```

If at every step the inequality is **strict**, termination follows from the fact that there are finitely many bases:

```
Definition basis_height I :=
  #|[ set J: feasible_bases |
     '[c, point_of_basis I] > '[c, point_of_basis J] ]|.
Function simplex_phase2 I {measure basis_height I} := [...].
```

The inequality may not be strict because of **degenerate bases**

= several bases correspond to the same basic point.



Termination

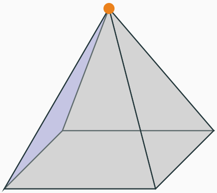
```
Fact [...]:
  '[c, point_of_basis next_I] <= '[c, point_of_basis I].
```

If at every step the inequality is **strict**, termination follows from the fact that there are finitely many bases:

```
Definition basis_height I :=
  #|[ set J: feasible_bases |
     '[c, point_of_basis I] > '[c, point_of_basis J] ]|.
Function simplex_phase2 I {measure basis_height I} := [...].
```

The inequality may not be strict because of **degenerate bases**

= several bases correspond to the same basic point.



Termination

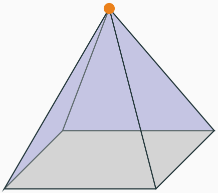
```
Fact [...]:
  '[c, point_of_basis next_I] <= '[c, point_of_basis I].
```

If at every step the inequality is **strict**, termination follows from the fact that there are finitely many bases:

```
Definition basis_height I :=
  #|[ set J: feasible_bases |
     '[c, point_of_basis I] > '[c, point_of_basis J] ]|.
Function simplex_phase2 I {measure basis_height I} := [...].
```

The inequality may not be strict because of **degenerate bases**

= several bases correspond to the same basic point.



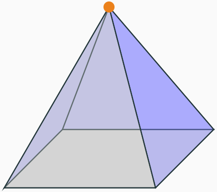
```
Fact [...]:
  '[c, point_of_basis next_I] <= '[c, point_of_basis I].
```

If at every step the inequality is **strict**, termination follows from the fact that there are finitely many bases:

```
Definition basis_height I :=
  #|[ set J: feasible_bases |
     '[c, point_of_basis I] > '[c, point_of_basis J] ]|.
Function simplex_phase2 I {measure basis_height I} := [...].
```

The inequality may not be strict because of **degenerate bases**

= several bases correspond to the same basic point.



Termination

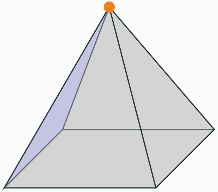
```
Fact [...]:
  '[c, point_of_basis next_I] <= '[c, point_of_basis I].
```

If at every step the inequality is **strict**, termination follows from the fact that there are finitely many bases:

```
Definition basis_height I :=
  #|[ set J: feasible_bases |
     '[c, point_of_basis I] > '[c, point_of_basis J] ]|.
Function simplex_phase2 I {measure basis_height I} := [...].
```

The inequality may not be strict because of **degenerate bases**

= several bases correspond to the same basic point.



Termination

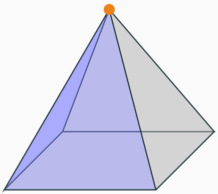
```
Fact [...]:
  '[c, point_of_basis next_I] <= '[c, point_of_basis I].
```

If at every step the inequality is **strict**, termination follows from the fact that there are finitely many bases:

```
Definition basis_height I :=
  #|[ set J: feasible_bases |
     '[c, point_of_basis I] > '[c, point_of_basis J] ]|.
Function simplex_phase2 I {measure basis_height I} := [...].
```

The inequality may not be strict because of **degenerate bases**

= several bases correspond to the same basic point.



Termination

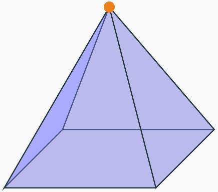
```
Fact [...]:
  '[c, point_of_basis next_I] <= '[c, point_of_basis I].
```

If at every step the inequality is **strict**, termination follows from the fact that there are finitely many bases:

```
Definition basis_height I :=
  #|[ set J: feasible_bases |
     '[c, point_of_basis I] > '[c, point_of_basis J] ]|.
Function simplex_phase2 I {measure basis_height I} := [...].
```

The inequality may not be strict because of **degenerate bases**

= several bases correspond to the same basic point.



Termination: dealing with degenerate bases

Our solution: Dantzig's lexicographic rule

- we consider a slightly **perturbed** LP, involving $0 < \varepsilon \ll 1$:

$$\text{minimize} \quad \langle c, x \rangle \quad \text{subject to} \quad Ax \geqslant \tilde{b}\,, \qquad \text{where } \tilde{b}_i := b_i - \varepsilon^i$$

Termination: dealing with degenerate bases

Our solution: Dantzig's lexicographic rule

- we consider a slightly **perturbed** LP, involving $0 < \varepsilon \ll 1$:

$$\text{minimize} \quad \langle c, x\rangle \quad \text{subject to} \quad Ax \geqslant \tilde{b}\,, \qquad \text{where } \tilde{b}_i := b_i - \varepsilon^i$$

- the perturbation of b into $\tilde{b}$ is done **symbolically**:
  every real v is now a polynomial in ε of degree $\leqslant m$

$$v + v_1\varepsilon + v_2\varepsilon^2 + \cdots + v_m\varepsilon^m$$

  encoded as a row vector $(v, v_1, \ldots, v_m)$.

Termination: dealing with degenerate bases

Our solution: Dantzig's lexicographic rule

- we consider a slightly **perturbed** LP, involving $0 < \varepsilon \ll 1$:

$$\text{minimize} \quad \langle c, x\rangle \quad \text{subject to} \quad Ax \geqslant \tilde{b}\,, \qquad \text{where } \tilde{b}_i := b_i - \varepsilon^i$$

- the perturbation of b into $\tilde{b}$ is done **symbolically**:
  every real v is now a polynomial in ε of degree $\leqslant m$

$$v + v_1\varepsilon + v_2\varepsilon^2 + \cdots + v_m\varepsilon^m$$

  encoded as a row vector $(v, v_1, \ldots, v_m)$.
- the usual order is replaced by the lexicographic order

Termination: dealing with degenerate bases

Our solution: Dantzig's lexicographic rule

- we consider a slightly **perturbed** LP, involving $0 < \varepsilon \ll 1$:

$$\text{minimize} \quad \langle c, x \rangle \quad \text{subject to} \quad Ax \geqslant \tilde{b}\,, \qquad \text{where } \tilde{b}_i := b_i - \varepsilon^i$$

- the perturbation of b into $\tilde{b}$ is done **symbolically**:
  every real v is now a polynomial in ε of degree $\leqslant m$

$$v + v_1\varepsilon + v_2\varepsilon^2 + \cdots + v_m\varepsilon^m$$

  encoded as a row vector $(v, v_1, \ldots, v_m)$.
- the usual order is replaced by the lexicographic order

```
Definition b_pert := (row_mx b -(1%:M)) : 'M_(m,1+m).
Definition point_of_basis_pert (I:basis) : 'M_(n,1+m) :=
  (qinvmx [...] (row_submx A I)) *m (row_submx b_pert I).
Lemma rel_points_of_basis (I:basis):
  point_of_basis I = col 0 (point_of_basis_pert I).
```

Termination: dealing with degenerate bases

Our solution: Dantzig's lexicographic rule

- we consider a slightly **perturbed** LP, involving $0 < \varepsilon \ll 1$:

$$\text{minimize} \quad \langle c, x\rangle \quad \text{subject to} \quad Ax \geqslant \tilde{b}\,, \qquad \text{where } \tilde{b}_i := b_i - \varepsilon^i$$

- the perturbation of b into $\tilde{b}$ is done **symbolically**:
  every real v is now a polynomial in ε of degree $\leqslant m$

$$v + v_1\varepsilon + v_2\varepsilon^2 + \cdots + v_m\varepsilon^m$$

  encoded as a row vector $(v, v_1, \ldots, v_m)$.
- the usual order is replaced by the lexicographic order

```
Definition b_pert := (row_mx b -(1%:M)) : 'M_(m,1+m).
Definition point_of_basis_pert (I:basis) : 'M_(n,1+m) :=
  (qinvmx [...] (row_submx A I)) *m (row_submx b_pert I).
Lemma rel_points_of_basis (I:basis):
  point_of_basis I = col 0 (point_of_basis_pert I).
```

$\Longrightarrow$ type `lex_feasible_basis` $\equiv$ feasible basis in the lex-setting:

```
Lemma lex_feasible_basis_is_feasible (I:lex_feasible_basis):
  is_feasible I.
```

Termination: dealing with degenerate bases (2)

No degenerate bases anymore!

```
Lemma eq_pert_point_imp_eq_bas (I I':basis) :
  point_of_basis_pert I = point_of_basis_pert I' -> I = I'.
```

Termination: dealing with degenerate bases (2)

No degenerate bases anymore!

```
Lemma eq_pert_point_imp_eq_bas (I I':basis) :
  point_of_basis_pert I = point_of_basis_pert I' -> I = I'.
```

This is based on the fact that the `(1+j)`th column of `point_of_basis_pert I` is nonzero if, and only if, `j` belongs to `I`:

```
Lemma col_point_of_basis_pert (I:basis) (j:'I_m):
  (col (rshift 1 j) (point_of_basis_pert I) != 0) = (j \in I).
```

Termination: dealing with degenerate bases (2)

No degenerate bases anymore!

```
Lemma eq_pert_point_imp_eq_bas (I I':basis) :
  point_of_basis_pert I = point_of_basis_pert I' -> I = I'.
```

This is based on the fact that the `(1+j)`th column of `point_of_basis_pert I` is nonzero if, and only if, `j` belongs to `I`:

```
Lemma col_point_of_basis_pert (I:basis) (j:'I_m):
  (col (rshift 1 j) (point_of_basis_pert I) != 0) = (j \in I).
```

which follows from

```
Lemma col_b_pert (I:prebasis) (j:'I_m):
  (col (rshift 1 j) (row_submx b_pert I) != 0) = (j \in I).
```

No degenerate bases anymore!

```
Lemma eq_pert_point_imp_eq_bas (I I':basis) :
  point_of_basis_pert I = point_of_basis_pert I' -> I = I'.
```

This is based on the fact that the `(1+j)`th column of `point_of_basis_pert I` is nonzero if, and only if, `j` belongs to `I`:

```
Lemma col_point_of_basis_pert (I:basis) (j:'I_m):
  (col (rshift 1 j) (point_of_basis_pert I) != 0) = (j \in I).
```

which follows from

```
Lemma col_b_pert (I:prebasis) (j:'I_m):
  (col (rshift 1 j) (row_submx b_pert I) != 0) = (j \in I).
```

This finally ensure that:

```
Fact [...] : (c^T *m point_of_basis_pert next_I)
    <lex (c^T *m point_of_basis_pert I).
```

Phase II simplex method

We arrive at a complete implementation of **Phase II** simplex method:

```
simplex_phase2 : feasible_basis -> result
```

where

```
Inductive result :=
| Optimal_basis of feasible_basis.
| Unbounded_cert (I: feasible_basis) of 'I_#|I|
```

Phase II simplex method

We arrive at a complete implementation of **Phase II** simplex method:

```
simplex_phase2 : feasible_basis -> result
```

where

```
Inductive result :=
| Optimal_basis of feasible_basis.
| Unbounded_cert (I: feasible_basis) of 'I_#|I|
```

Correction is specified via an inductive predicate (*a la* MathComp):

```
Lemma phase2P : forall I_0, phase2_spec (simplex_phase2 I_0).
```

where

```
Inductive phase2_spec : result -> Type :=
```

We arrive at a complete implementation of **Phase II** simplex method:

```
simplex_phase2 : feasible_basis -> result
```

where

```
Inductive result :=
| Optimal_basis of feasible_basis.
| Unbounded_cert (I: feasible_basis) of 'I_#|I|
```

Correction is specified via an inductive predicate (*a la* MathComp):

```
Lemma phase2P : forall I_0, phase2_spec (simplex_phase2 I_0).
```

where

```
Inductive phase2_spec : result -> Type :=
| Optimal (I: feasible_basis) of
  (reduced_cost I) >=m 0 : phase2_spec (Optimal_basis I).
```

Phase II simplex method

We arrive at a complete implementation of **Phase II** simplex method:

```
simplex_phase2 : feasible_basis -> result
```

where

```
Inductive result :=
| Optimal_basis of feasible_basis.
| Unbounded_cert (I: feasible_basis) of 'I_#|I|
```

Correction is specified via an inductive predicate (*a la* MathComp):

```
Lemma phase2P : forall I_0, phase2_spec (simplex_phase2 I_0).
```

where

```
Inductive phase2_spec : result -> Type :=
| Optimal (I: feasible_basis) of
  (reduced_cost I) >=m 0 : phase2_spec (Optimal_basis I).
| Unbounded (I: feasible_basis) (i: 'I_#|I|) of
  (reduced_cost I) i 0 < 0 /\ (A *m direction I i) >=m 0 :
    phase2_spec (Unbounded_cert i).
```

Feasibility as a Coq program

Corollary (of duality)

$\{x \in \mathbb{R}^n : Ax \geqslant b\}$ is empty if, and only if, the dual LP is ***unbounded***:

$$\text{maximize} \quad \langle b, u \rangle \quad \text{subject to} \quad A^T u = 0\,,\ u \geqslant 0\,,\ u \in \mathbb{R}^m$$

Feasibility as a Coq program

Corollary (of duality)

$\{x \in \mathbb{R}^n : Ax \geqslant b\}$ is empty if, and only if, the dual LP is ***unbounded****:*

$$\textit{maximize} \quad \langle b, u \rangle \quad \textit{subject to} \quad A^T u = 0\,,\; u \geqslant 0\,,\; u \in \mathbb{R}^m$$

```
Definition feasible :=
  let dualA := col_mx (col_mx A^T (-A^T)) (1%:M) in
  match simplex_phase2 dualA 0 (-b) with
  | Optimal_basis _ => true | Unbounded_cert _ => false.
```

Feasibility as a Coq program

Corollary (of duality)

*$\{x \in \mathbb{R}^n : Ax \geqslant b\}$ is empty if, and only if, the dual LP is **unbounded**:*

$$\text{maximize} \quad \langle b, u \rangle \quad \text{subject to} \quad A^T u = 0\,,\; u \geqslant 0\,,\; u \in \mathbb{R}^m$$

```
Definition feasible :=
  let dualA := col_mx (col_mx A^T (-A^T)) (1%:M) in
  match simplex_phase2 dualA 0 (-b) with
  | Optimal_basis _ => true | Unbounded_cert _ => false.
```

We relate this definition with the usual logical statements:

Feasibility as a Coq program

Corollary (of duality)

*$\{x \in \mathbb{R}^n : Ax \geqslant b\}$ is empty if, and only if, the dual LP is **unbounded**:*

$$\text{maximize} \quad \langle b, u \rangle \quad \text{subject to} \quad A^T u = 0\,,\ u \geqslant 0\,,\ u \in \mathbb{R}^m$$

```
Definition feasible :=
  let dualA := col_mx (col_mx A^T (-A^T)) (1%:M) in
  match simplex_phase2 dualA 0 (-b) with
  | Optimal_basis _ => true | Unbounded_cert _ => false.
```

We relate this definition with the usual logical statements:

```
Lemma feasibleP :
  reflect (exists x, x \in polyhedron A b) feasible.
```

Proof sketch

- the witness `x` is built from the reduced costs vector associated with the optimal basis

Feasibility as a Coq program

Corollary (of duality)

$\{x \in \mathbb{R}^n : Ax \geqslant b\}$ is empty if, and only if, the dual LP is ***unbounded****:*

$$\text{maximize} \quad \langle b, u\rangle \quad \text{subject to} \quad A^T u = 0\,,\ u \geqslant 0\,,\ u \in \mathbb{R}^m$$

```
Definition feasible :=
  let dualA := col_mx (col_mx A^T (-A^T)) (1%:M) in
  match simplex_phase2 dualA 0 (-b) with
  | Optimal_basis _ => true | Unbounded_cert _ => false.
```

We relate this definition with the usual logical statements:

```
Lemma feasibleP :
  reflect (exists x, x \in polyhedron A b) feasible.
```

```
Lemma infeasibleP : (* Farkas Lemma *)
  reflect (exists d, [/\ A^T *m d = 0, d >=m 0 & '[b,d] > 0])
    (~~ feasible).
```

Proof sketch

- the witness `x` is built from the reduced costs vector associated with the optimal basis
- inconsistency cert. `d` is built from the unboundedness certificate □

Phase II requires a feasible basis to start with... **Phase I finds it!**

Phase I: finding a feasible basis

Phase II requires a feasible basis to start with… **Phase I finds it!**

…by calling Phase II

Phase I: finding a feasible basis

Phase II requires a feasible basis to start with… **Phase I finds it!**

…by calling Phase II
…on a certain LP for which an initial feasible basis can be easily built.

Phase I: finding a feasible basis

Phase II requires a feasible basis to start with... **Phase I finds it!**

...by calling Phase II
...on a certain LP for which an initial feasible basis can be easily built.

Assumption: the polyhedron is **pointed**

```
Hypothesis Hpointed: (\rank A >= n)%N.
```

Phase I: finding a feasible basis

Phase II requires a feasible basis to start with... **Phase I finds it!**

...by calling Phase II
...on a certain LP for which an initial feasible basis can be easily built.

Assumption: the polyhedron is **pointed**

```
Hypothesis Hpointed: (\rank A >= n)%N.
```

Phase I Linear Program

$$\begin{aligned} \text{minimize} \quad & \langle e, y - A_K x \rangle \\ \text{subject to} \quad & A_K x \leqslant b_K + y \,,\; A_L x \geqslant b_L \\ & y \geqslant 0 \,,\; (x, y) \in \mathbb{R}^{n+p} \end{aligned}$$

where K and L are built from an (arbitrary) basis $\mathcal{I}$:

$$K := \{ i \in [m] : A_i x^{\mathcal{I}} < b_i \} \,, \qquad L := \{ i \in [m] : A_i x^{\mathcal{I}} \geqslant b_i \} \,.$$

Phase I: finding a feasible basis

Phase II requires a feasible basis to start with… **Phase I finds it!**

…by calling Phase II
…on a certain LP for which an initial feasible basis can be easily built.

Assumption: the polyhedron is **pointed**

```
Hypothesis Hpointed: (\rank A >= n)%N.
```

Phase I Linear Program

$$\begin{aligned} \text{minimize} \quad & \langle e, y - A_K x \rangle \\ \text{subject to} \quad & A_K x \leqslant b_K + y\,,\ A_L x \geqslant b_L \\ & y \geqslant 0\,,\ (x, y) \in \mathbb{R}^{n+p} \end{aligned}$$

where K and L are built from an (arbitrary) basis $\mathcal{I}$:

$$K := \{i \in [m] : A_i x^{\mathcal{I}} < b_i\}\,, \qquad L := \{i \in [m] : A_i x^{\mathcal{I}} \geqslant b_i\}\,.$$

$\Longrightarrow$ we finally obtain a **complete implementation** of the simplex method.

Phase I: finding a feasible basis

Phase II requires a feasible basis to start with… **Phase I finds it!**

…by calling Phase II
…on a certain LP for which an initial feasible basis can be easily built.

Assumption: the polyhedron is **pointed**

```
Hypothesis Hpointed: (\rank A >= n)%N.
```

Phase I Linear Program

$$\begin{aligned} \text{minimize} \quad & \langle e, y - A_K x \rangle \\ \text{subject to} \quad & A_K x \leqslant b_K + y\,, \; A_L x \geqslant b_L \\ & y \geqslant 0\,, \; (x, y) \in \mathbb{R}^{n+p} \end{aligned}$$

where K and L are built from an (arbitrary) basis $\mathcal{I}$:

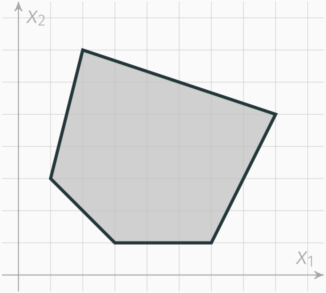
$$K := \{i \in [m] : A_i x^{\mathcal{I}} < b_i\}\,, \qquad L := \{i \in [m] : A_i x^{\mathcal{I}} \geqslant b_i\}\,.$$

$\Longrightarrow$ we finally obtain a **complete implementation** of the simplex method.

BUT this requires advanced manipulations of block matrices (> 500 lines).

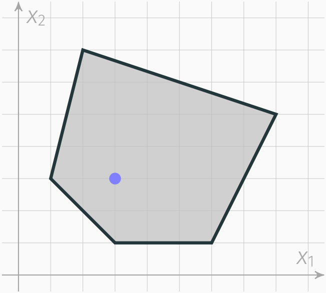
Phase I: finding a feasible basis (2)

Alternative algorithm :



Alternative algorithm :

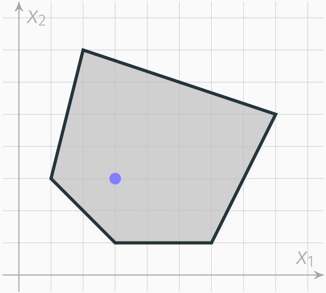
- start from an arbitrary $x \in \mathcal{P}(A, b)$



Phase I: finding a feasible basis (2)

Alternative algorithm :

- start from an arbitrary $x \in \mathcal{P}(A, b)$
- let $\mathcal{I}(x) := \{i \in [m] : A_i x = b_i\}$
- take $d \neq 0$ such that $d \in \ker A_{\mathcal{I}(x)}$

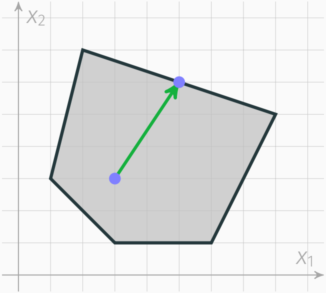


Phase I: finding a feasible basis (2)

Alternative algorithm :

- start from an arbitrary $x \in \mathcal{P}(A, b)$
- let $\mathcal{I}(x) := \{i \in [m] : A_i x = b_i\}$
- take $d \neq 0$ such that $d \in \ker A_{\mathcal{I}(x)}$
- move along $\pm d$ from x, up to the boundary
  $\implies$ new point $x' \in \mathcal{P}(A, b)$, s.t.

$$\operatorname{rank} A_{\mathcal{I}(x')} > \operatorname{rank} A_{\mathcal{I}(x)}$$

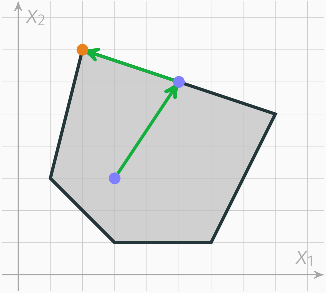


Phase I: finding a feasible basis (2)

Alternative algorithm :

- start from an arbitrary $x \in \mathcal{P}(A, b)$
- let $\mathcal{I}(x) := \{i \in [m] : A_i x = b_i\}$
- take $d \neq 0$ such that $d \in \ker A_{\mathcal{I}(x)}$
- move along $\pm d$ from x, up to the boundary
  $\implies$ new point $x' \in \mathcal{P}(A, b)$, s.t.

$$\operatorname{rank} A_{\mathcal{I}(x')} > \operatorname{rank} A_{\mathcal{I}(x)}$$



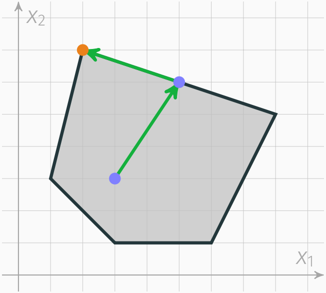
Phase I: finding a feasible basis (2)

Alternative algorithm :

- start from an arbitrary $x \in \mathcal{P}(A, b)$
- let $\mathcal{I}(x) := \{i \in [m] : A_i x = b_i\}$
- take $d \neq 0$ such that $d \in \ker A_{\mathcal{I}(x)}$
- move along $\pm d$ from x, up to the boundary
  $\Longrightarrow$ new point $x' \in \mathcal{P}(A, b)$, s.t.

$$\operatorname{rank} A_{\mathcal{I}(x')} > \operatorname{rank} A_{\mathcal{I}(x)}$$

- if $\ker A_{\mathcal{I}(x)} = \{0\}$, x is a feasible basic point.

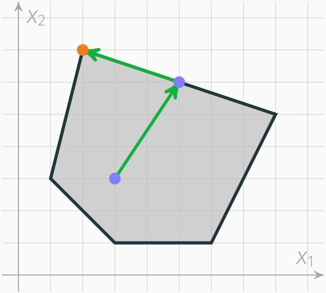


Alternative algorithm :

- start from an arbitrary $x \in \mathcal{P}(A, b)$
- let $\mathcal{I}(x) := \{i \in [m] : A_i x = b_i\}$
- take $d \neq 0$ such that $d \in \ker A_{\mathcal{I}(x)}$
- move along $\pm d$ from x, up to the boundary
  $\Longrightarrow$ new point $x' \in \mathcal{P}(A, b)$, s.t.

$$\operatorname{rank} A_{\mathcal{I}(x')} > \operatorname{rank} A_{\mathcal{I}(x)}$$

- if $\ker A_{\mathcal{I}(x)} = \{0\}$, x is a feasible basic point.



Remark

- the initial x is provided by the dual Phase II:

```
Lemma feasibleP :
  reflect (exists x, x \in polyhedron A b) feasible.
```

- share steps with the simplex method
- conceptually (and computationally) simpler

General case

Reduction to the **pointed** case:

$$\begin{aligned} \text{minimize} \quad & \langle c, v - w \rangle \\ \text{subject to} \quad & A(v - w) \geqslant b\,,\ v \geqslant 0\,,\ w \geqslant 0\,,\ (v, w) \in \mathbb{R}^{n+n} \end{aligned} \tag{1}$$

We arrive at the definition of the function **simplex**:

```
Inductive simplex_spec : simplex_final_result -> Type :=
| Infeasible d of ([/\ A^T *m d = 0, d >=m 0 & '[b, d] > 0):
                simplex_spec (Simplex_infeasible d)
| Unbounded p of
  [/\ (p.1 \in polyhedron A b), A *m p.2 >=m 0 &
   '[c,p.2] < 0]: simplex_spec (Simplex_unbounded p)
| Optimal_point p of
  [/\ (p.1 \in polyhedron A b), (p.2 \in dual_polyhedron A c) &
   '[c,p.1] = '[b,p.2]]: simplex_spec (Simplex_optimal_point p).

Theorem simplexP : simplex_spec simplex.
```

which completely solves an LP (from scratch).

Additional Boolean predicates on polyhedra

```
Definition unbounded :=
  if simplex is Simplex_unbounded _ then true else false.
Lemma unboundedP : reflect
  (forall M, exists y, y \in polyhedron A b /\ '[c,y] < M)
    unbounded.
```

Additional Boolean predicates on polyhedra

```
Definition unbounded :=
  if simplex is Simplex_unbounded _ then true else false.
Lemma unboundedP : reflect
  (forall M, exists y, y \in polyhedron A b /\ '[c,y] < M)
    unbounded.
```

Proof sketch

```
[...]
| Unbounded p of
  [/\ (p.1 \in polyhedron A b), (A *m p.2 >=m 0) &
   '[c,p.2] < 0]: simplex_spec (Simplex_unbounded p)
[...]
```

The certificate `y` is built by taking a point of the form $\texttt{p.1} + \lambda\texttt{p.2}$, where $\lambda \geqslant 0$ is sufficiently large.

Effective formalization of convex polyhedra

Strong duality

Primal LP

$$\begin{array}{ll} \text{minimize} & \langle c, x \rangle \\ \text{subject to} & Ax \geqslant b\,,\ x \in \mathbb{R}^n \end{array}$$

Dual LP

$$\begin{array}{ll} \text{maximize} & \langle b, u \rangle \\ \text{subject to} & A^T u = c\,,\ u \geqslant 0\,,\ u \in \mathbb{R}^m \end{array}$$

Theorem

- *the value of the primal LP is $\geqslant$ the value of the dual LP;*
- *both LP have the* ***same value****, unless both LP are infeasible.*

Strong duality

Primal LP

$$\begin{array}{ll} \text{minimize} & \langle c, x\rangle \\ \text{subject to} & Ax \geqslant b\,,\ x \in \mathbb{R}^n \end{array}$$

Dual LP

$$\begin{array}{ll} \text{maximize} & \langle b, u\rangle \\ \text{subject to} & A^T u = c\,,\ u \geqslant 0\,,\ u \in \mathbb{R}^m \end{array}$$

Theorem

- *the value of the primal LP is $\geqslant$ the value of the dual LP;*
- *both LP have the* ***same value****, unless both LP are infeasible.*

```
Fact weak_duality : forall x, forall u,
  x \in polyhedron A b -> u \in dual_polyhedron A c ->
    '[c,x] >= '[b,u].
Fact strong_duality : [...]
  exists x, exists u, x \in polyhedron A b ->
    u \in dual_polyhedron A c -> '[c,x] = '[b,u].
```

Strong duality

Primal LP		**Dual LP**	
minimize	$\langle c, x\rangle$	maximize	$\langle b, u\rangle$
subject to	$Ax \geqslant b\,,\ x \in \mathbb{R}^n$	subject to	$A^T u = c\,,\ u \geqslant 0\,,\ u \in \mathbb{R}^m$

Theorem

- *the value of the primal LP is $\geqslant$ the value of the dual LP;*
- *both LP have the* ***same value****, unless both LP are infeasible.*

```
Fact weak_duality : forall x, forall u,
  x \in polyhedron A b -> u \in dual_polyhedron A c ->
    '[c,x] >= '[b,u].
Fact strong_duality : [...]
  exists x, exists u, x \in polyhedron A b ->
    u \in dual_polyhedron A c -> '[c,x] = '[b,u].
```

Proof sketch

```
| Optimal_point p of
  [/\ (p.1 \in polyhedron A b), (p.2 \in dual_polyhedron A c) &
   '[c,p.1] = '[b,p.2]]: simplex_spec (Simplex_optimal_point p).
```

Convex hulls

Definition

A point $x \in \mathbb{R}^n$ belongs to the convex hull of the set $V = \{v^1, \dots, v^p\}$ if

$$\exists \lambda \in \mathbb{R}^p\,, \quad x = \sum_{i=1}^{p} \lambda_i v^i \quad \text{where } \lambda \geqslant 0\,,\ \sum_{i=1}^{p} \lambda_i = 1\,.$$

Convex hulls

Definition

A point $x \in \mathbb{R}^n$ belongs to the convex hull of the set $V = \{v^1, \dots, v^p\}$ if

$$\exists \lambda \in \mathbb{R}^p, \quad x = \sum_{i=1}^{p} \lambda_i v^i \quad \text{where } \lambda \geqslant 0, \sum_{i=1}^{p} \lambda_i = 1.$$

$\Longrightarrow$ membership amounts to the non-emptiness of a polyhedron in $\lambda \in \mathbb{R}^p$ (parametrized by x and V)

```
Let e := (const_mx 1):'cV_p. (* vector with constant entry 1 *)
Definition is_in_convex_hull (x:'cV_n) :=
  let Ax :=
    col_mx (col_mx (col_mx V (-V)) (col_mx e^T (-e^T))) 1%:M in
  let bx :=
    col_mx (col_mx (col_mx x (-x)) (col_mx 1 (-1))) (0:'cV_p) in
  feasible Ax bx.
```

Convex hulls

Definition

A point $x \in \mathbb{R}^n$ belongs to the convex hull of the set $V = \{v^1, \dots, v^p\}$ if

$$\exists \lambda \in \mathbb{R}^p \,, \quad x = \sum_{i=1}^{p} \lambda_i v^i \quad \text{where } \lambda \geqslant 0 \,, \sum_{i=1}^{p} \lambda_i = 1 \,.$$

$\Longrightarrow$ membership amounts to the non-emptiness of a polyhedron in $\lambda \in \mathbb{R}^p$ (parametrized by x and V)

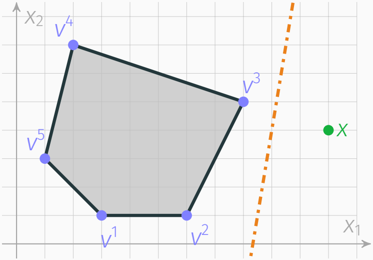
```
Let e := (const_mx 1):'cV_p. (* vector with constant entry 1 *)
Definition is_in_convex_hull (x:'cV_n) :=
  let Ax :=
    col_mx (col_mx (col_mx V (-V)) (col_mx e^T (-e^T))) 1%:M in
  let bx :=
    col_mx (col_mx (col_mx x (-x)) (col_mx 1 (-1))) (0:'cV_p) in
  feasible Ax bx.

Lemma is_in_convex_hullP (x:'cV_n) : reflect
  (exists lambda:'cV_p,
    [/\ (lambda >=m 0), '[e, lambda] = 1 & x = V *m lambda])
  (is_in_convex_hull x).
```

Convex hulls (2)

Theorem (Separation result)

*If x **does not** belong to the convex hull of V, there exists $c \in \mathbb{R}^n$ such that*

$$\langle c, v^i \rangle > \langle c, x \rangle \,, \quad i = 1, \ldots, p$$

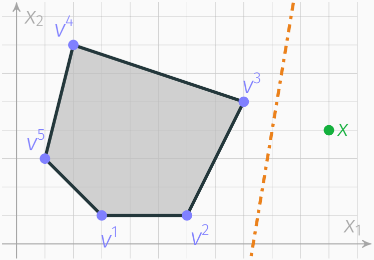


Convex hulls (2)

Theorem (Separation result)

*If x **does not** belong to the convex hull of V, there exists $c \in \mathbb{R}^n$ such that*

$$\langle c, v^i \rangle > \langle c, x \rangle \,, \quad i = 1, \dots, p$$




```
Theorem separation (x: 'cV_n) : ~~ (is_in_convex_hull x)
  -> exists c, [forall i, '[c, col i V] > '[c, x]].
```

Proof sketch

The certificate `c` is built as `(dsubmx (usubmx (usubmx d)))-(usubmx (usubmx (usubmx d)))` where `d` is the infeasibility certificate of the polyhedron over $\lambda \in \mathbb{R}^p$.

Minkowski Theorem

Theorem

Every bounded polyhedron is the convex hull of finitely many points.

Minkowski Theorem

Theorem

Every bounded polyhedron is the convex hull of finitely many points.

```
Theorem minkowski : bounded_polyhedron A b ->
  polyhedron A b =i is_in_convex_hull matrix_of_points.
```

where:

- `=i` is the extensional equality
- `matrix_of_points` is the matrix of the feasible basic points

Minkowski Theorem

Theorem

Every bounded polyhedron is the convex hull of finitely many points.

```
Theorem minkowski : bounded_polyhedron A b ->
  polyhedron A b =i is_in_convex_hull matrix_of_points.
```

where:

- `=i` is the extensional equality
- `matrix_of_points` is the matrix of the feasible basic points

Proof sketch

Suppose that x lies in the polyhedron.

If x does not belong to the convex hull of the basic points, there exists c such that

$$\langle c, x \rangle < \langle c, z \rangle \quad \text{for all feasible basic point } z$$

Minkowski Theorem

Theorem

Every bounded polyhedron is the convex hull of finitely many points.

```
Theorem minkowski : bounded_polyhedron A b ->
  polyhedron A b =i is_in_convex_hull matrix_of_points.
```

where:

- `=i` is the extensional equality
- `matrix_of_points` is the matrix of the feasible basic points

Proof sketch

Suppose that x lies in the polyhedron.

If x does not belong to the convex hull of the basic points, there exists c such that

$$\langle c, z^* \rangle \leqslant \langle c, x \rangle < \langle c, z \rangle \quad \text{for all feasible basic point } z$$

where z^* is the basic point found by the simplex method. □

Conclusion

Conclusion

Summary of the contributions

First steps of the formalization of the theory of convex polyhedra in Coq

- carried out in an effective way
- by exploiting a formalization of the simplex method

Conclusion

Summary of the contributions

First steps of the formalization of the theory of convex polyhedra in Coq

- carried out in an effective way
- by exploiting a formalization of the simplex method

Perspectives

- continue the development of the theory
  Example: faces → dimension → affine hull

Conclusion

Summary of the contributions

First steps of the formalization of the theory of convex polyhedra in Coq

- carried out in an effective way
- by exploiting a formalization of the simplex method

Perspectives

- continue the development of the theory
  Example: faces → dimension → affine hull
- formalize combinatorial enumeration algorithms
  Example: enumerate the vertices of a polyhedron

Conclusion

Summary of the contributions

First steps of the formalization of the theory of convex polyhedra in Coq

- carried out in an effective way
- by exploiting a formalization of the simplex method

Perspectives

- continue the development of the theory
  Example: faces → dimension → affine hull
- formalize combinatorial enumeration algorithms
  Example: enumerate the vertices of a polyhedron
- handle large-scale instances
  Example: formally disprove Hirsch conjecture? (e.g., $\sim 35\,000$ vertices)

Conclusion

Summary of the contributions

First steps of the formalization of the theory of convex polyhedra in Coq

- carried out in an effective way
- by exploiting a formalization of the simplex method

Perspectives

- continue the development of the theory
  Example: faces $\rightarrow$ dimension $\rightarrow$ affine hull
- formalize combinatorial enumeration algorithms
  Example: enumerate the vertices of a polyhedron
- handle large-scale instances
  Example: formally disprove Hirsch conjecture? (e.g., $\sim 35\,000$ vertices)
  $\Longrightarrow$ refine algorithms to work with low-level data structures

Thank you!

`github.com/nhojem/Coq-Polyhedra`

References

Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A machine-checked proof of the odd order theorem. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, pages 163–179, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39634-2.

Georges Gonthier, Assia Mahboubi, and Enrico Tassi. A Small Scale Reflection Extension for the Coq system. Research Report RR-6455, Inria Saclay Ile de France, 2016.

Thomas Hales, MARK ADAMS, GERTRUD BAUER, TAT DAT DANG, JOHN HARRISON, LE TRUONG HOANG, CEZARY KALISZYK, VICTOR MAGRON, SEAN MCLAUGHLIN, TAT THANG NGUYEN, and et al. A formal proof of the kepler conjecture. *Forum of Mathematics, Pi*, 5:e2, 2017. doi: 10.1017/fmp.2017.1.

John Harrison. The HOL Light theory of Euclidean space. *Journal of Automated Reasoning*, 50, 2013.

Kazuhiko Sakaguchi. Vass. `https://github.com/pi8027/vass`, 2016.

Mirko Spasić and Filip Marić. Formalization of incremental simplex algorithm by stepwise refinement. In Dimitra Giannakopoulou and Dominique Méry,

editors, *Proceedings of FM 2012*. Springer, 2012. ISBN 978-3-642-32759-9.
doi: 10.1007/978-3-642-32759-9_35.