Logic and binary variables

Outline

Propositions and binary variables

Literals, clauses, conjunctive normal form

From CNF to linear constraints

Logical implications

Splitting an implication: when it is allowed

Counting: at most one, at least one, exactly one

Logical operators and set operators

Checking the translation, not trusting it

Five exercises

Logical constraints inside an optimisation model

What this chapter has put in your hands

Learning objectives (1/2)

At the end of this chapter you will be able to:

- bring a logical condition between binary decisions to conjunctive normal form;
- translate every clause into a linear inequality and prove that the translation is exact;
- write implications, their contrapositive and the *if and only if*, and say when an implication can be split;

Learning objectives (2/2)

At the end of this chapter you will be able to:

- write cardinality constraints: at most one, at least one, exactly one;
- add logical constraints to an optimisation model and solve it in `gurobipy`.

Prerequisites

- From the module *Mathematical Analysis 1*, the chapter «Basics of logic and proof techniques»: propositions, logical connectives and their truth tables
- From this module, Chapter 1 «What is a MIP model»: binary variables and the set covering, packing and partitioning patterns

What this is about

- A binary variable is a "**yes/no**" answer
- In this chapter we learn to translate logical conditions between those answers into **linear constraints**
- And — above all — to **prove** that the translation is exact

Propositions and binary variables

Introduction: Propositions and binary variables

- We start from the link between a **proposition** (true or false) and a **binary variable**
- We introduce **boolean expressions**, built with `AND` , `OR` , `NOT`
- We look at the **satisfiability** problem: is there an assignment that makes the expression true?
- This is the language in which we will describe logical conditions in MIP models

One proposition, one variable

- A **boolean function** returns one of the two values `TRUE` and `FALSE`
- It is represented by a binary variable

$$x \in \{0, 1\}, \qquad \begin{aligned} x = 1 &\iff \text{the proposition is } \texttt{TRUE} \\ x = 0 &\iff \text{the proposition is } \texttt{FALSE} \end{aligned}$$

- A **boolean expression** is built from:
  - binary variables
  - the three operators `AND` (conjunction, $\wedge$), `OR` (disjunction, $\vee$) and `NOT` (negation, $\neg$)
  - parentheses

Example: two boolean expressions

$$\texttt{NOT}\ x_a \quad \texttt{OR} \quad \big((x_b \quad \texttt{OR} \quad x_c) \quad \texttt{AND} \quad (x_d \quad \texttt{OR} \quad x_e)\big)$$

$$(x_a \quad \texttt{OR} \quad x_b) \quad \texttt{AND} \quad (x_c \quad \texttt{OR} \quad x_d) \quad \texttt{AND} \quad \texttt{NOT}\ x_e$$

Satisfiability

- The **boolean satisfiability problem** asks whether there is an assignment of logical values to the variables that makes the expression true
- If there is, the expression is **satisfiable**
- Otherwise it is **unsatisfiable**

Example: satisfiable and not (1/2)

- The formula

$$\texttt{NOT}\ x_a\ \ \texttt{OR}\ \ ((x_b\ \ \texttt{OR}\ \ x_c)\ \ \texttt{AND}\ \ (x_d\ \ \texttt{OR}\ \ x_e))$$

  is satisfied, for instance, by $x_a = 0$ (the others arbitrary)

- The formula

$$(x_a\ \ \texttt{OR}\ \ x_b)\ \ \texttt{AND}\ \ (x_c\ \ \texttt{OR}\ \ x_d)\ \ \texttt{AND}\ \ \texttt{NOT}\ x_e$$

  is satisfied by $x_a = 1$, $x_c = 1$, $x_b = x_d = x_e = 0$

Example: satisfiable and not (2/2)

- The formula

$$(x_a \ \texttt{OR} \ x_b) \ \texttt{AND} \ (\texttt{NOT} \ x_a \ \texttt{OR} \ x_b)$$
$$\texttt{AND} \ (x_a \ \texttt{OR} \ \texttt{NOT} \ x_b) \ \texttt{AND} \ (\texttt{NOT} \ x_a \ \texttt{OR} \ \texttt{NOT} \ x_b)$$

  is **unsatisfiable**

- Each of the four possible assignments of (x_a, x_b) falsifies one of the four clauses

Summary

Propositions and binary variables:

- A true/false proposition $\leftrightarrow$ a variable $x \in \{0, 1\}$
- Boolean expressions use `AND` , `OR` , `NOT` and parentheses

Satisfiability:

- An expression is satisfiable if some assignment makes it true
- Otherwise it is unsatisfiable

Literals, clauses, conjunctive normal form

Introduction: Literals, clauses, CNF

- We define three key words: **literal**, **clause**, **conjunctive normal form** (CNF)
- CNF is the form from which we will write the linear constraints
- We look at the **equivalences** that bring any boolean expression into CNF

The three words

- A **literal** is a variable (x_a, positive literal) or its negation (`NOT` x_a, negative literal)
- A **clause** is a disjunction (`OR`) of literals, or a single literal
- An expression is in **conjunctive normal form** (CNF) if it is a conjunction (`AND`) of clauses, or a single clause

- Of the two expressions in the previous example, the second is in CNF and the first is not
- Every boolean expression **can** be rewritten in CNF
- To do so, we use the equivalences below

The equivalences that are needed (1/2)

For every triple of binary variables $x_a, x_b, x_c \in \{0, 1\}$:

Distributive properties (D1), (D2):

$$x_a \ \texttt{AND} \ (x_b \ \texttt{OR} \ x_c) \iff (x_a \ \texttt{AND} \ x_b) \ \texttt{OR} \ (x_a \ \texttt{AND} \ x_c)$$

$$x_a \ \texttt{OR} \ (x_b \ \texttt{AND} \ x_c) \iff (x_a \ \texttt{OR} \ x_b) \ \texttt{AND} \ (x_a \ \texttt{OR} \ x_c)$$

De Morgan's laws (M1), (M2):

$$\texttt{NOT}\,(x_a \ \texttt{OR} \ x_b) \iff \texttt{NOT}\ x_a \ \texttt{AND} \ \texttt{NOT}\ x_b$$

$$\texttt{NOT}\,(x_a \ \texttt{AND} \ x_b) \iff \texttt{NOT}\ x_a \ \texttt{OR} \ \texttt{NOT}\ x_b$$

The equivalences that are needed (2/2)

Absorption (A1), (A2):

$$x_a \ \texttt{OR} \ (x_a \ \texttt{AND} \ x_b) \iff x_a$$

$$x_a \ \texttt{AND} \ (x_a \ \texttt{OR} \ x_b) \iff x_a$$

Double negation (N):

$$\texttt{NOT}\,(\texttt{NOT}\ x_a) \iff x_a$$

- They hold with more than two terms inside the parentheses and with expressions in place of the variables

How they are proved

- Every equivalence is proved **by cases**
- With two variables there are 4 assignments, with three there are 8
- One checks that the two sides agree on each assignment
- The script `cap02_logic.py` runs this check on all seven equivalences

Collecting: (D1) read backwards

- Read from right to left, the distributive property (D1) allows two conjunctions sharing a literal to be **collected**:

$$\begin{aligned} & (x_a \ \texttt{AND} \ x_b) \ \texttt{OR} \ (x_a \ \texttt{AND} \ x_c) \\ \Longleftrightarrow \ & x_a \ \texttt{AND} \ (x_b \ \texttt{OR} \ x_c) \end{aligned}$$

- It is useful when putting into CNF a disjunction of several conjunctions with common literals
- It reduces, step by step, the number of terms to distribute

Example: two transformations into CNF (1/3)

- For the formula

$$\texttt{NOT}\ x_a\ \ \texttt{OR}\ \ ((x_b\ \ \texttt{OR}\ \ x_c)\ \ \texttt{AND}\ \ (x_d\ \ \texttt{OR}\ \ x_e))$$

the distributive (D2) is enough, and it becomes

$$(\texttt{NOT}\ x_a\ \ \texttt{OR}\ \ x_b\ \ \texttt{OR}\ \ x_c)\ \ \texttt{AND}\ \ (\texttt{NOT}\ x_a\ \ \texttt{OR}\ \ x_d\ \ \texttt{OR}\ \ x_e)$$

Example: two transformations into CNF (2/3)

- The formula

$$(x_a \ \texttt{AND} \ x_b) \ \texttt{OR} \ (x_a \ \texttt{AND} \ x_c) \ \texttt{OR} \ (x_b \ \texttt{AND} \ x_c)$$

  says "at least two of a, b, c"
- Collect x_a in the first two:

$$\left[x_a \ \texttt{AND} \ (x_b \ \texttt{OR} \ x_c)\right] \ \texttt{OR} \ (x_b \ \texttt{AND} \ x_c)$$

- Apply (D2) and obtain the two factors

$$x_a \ \texttt{OR} \ (x_b \ \texttt{AND} \ x_c)$$

$$\text{and} \quad (x_b \ \texttt{OR} \ x_c) \ \texttt{OR} \ (x_b \ \texttt{AND} \ x_c)$$

Example: two transformations into CNF (3/3)

- The second factor reduces to (x_b OR x_c) by absorption (A1)
- The first is distributed once more with (D2)
- Altogether:

$$(x_a \text{ OR } x_b) \text{ AND } (x_a \text{ OR } x_c) \text{ AND } (x_b \text{ OR } x_c)$$

- This will be the key to Exercise 4

Summary

The three words:

- Literal: x or `NOT` x
- Clause: `OR` of literals
- CNF: `AND` of clauses

Bringing into CNF:

- Distributivity, De Morgan, absorption, double negation
- Every equivalence is proved by cases, enumerating the assignments
- Collecting with (D1) backwards reduces the terms to distribute

From CNF to linear constraints

Introduction: From CNF to linear constraints

- This is the heart of the chapter: from an expression in CNF to a **system of linear constraints**
- **Three rules** are enough
- We see why the translation is **exact** and in which form the constraints are best written

The translation, in three rules

The translation, in three rules

A boolean expression in CNF is equivalent to a system of linear constraints on the same binary variables:

1. every **clause** becomes an inequality constraint ≥ 1;
2. every `OR` inside the clause becomes a $+$;
3. every negative literal `NOT` x becomes $1 - x$.

Why it is exact (1/2)

- A clause with positive literals P and negative literals N becomes

$$\sum_{i \in P} x_i + \sum_{i \in N} (1 - x_i) \geq 1$$

- that is, moving the constants to the right,

$$\sum_{i \in P} x_i - \sum_{i \in N} x_i \geq 1 - |N|$$

Why it is exact (2/2)

- $x_i = 1$ contributes 1 to the left-hand side if the literal is positive and satisfied
- $1 - x_i = 1$ contributes 1 if the literal is negative and satisfied
- The sum counts **how many literals of the clause are true**: asking it to be ≥ 1 is exactly asking the clause to be true
- The conjunction of clauses becomes the system of all the constraints, because they must all hold together

The form in which the constraint is written

Observation 1: The form in which the constraint is written

With two or more negative literals the equivalent form obtained by multiplying by -1 *reads better: for instance*

$$1 - x_1 + 1 - x_6 + x_7 \geq 1 \quad \textit{is written} \quad x_1 + x_6 - x_7 \leq 1.$$

It is the same constraint. What never changes is the number of constraints: ***one per clause****, and clauses are counted after simplification (removing the tautologies* x `OR` `NOT` x *and the clauses absorbed by a subclause of theirs).*

Example: three translations (1/2)

$$(\texttt{NOT}\ x_a\ \ \texttt{OR}\ \ x_b\ \ \texttt{OR}\ \ x_c)\ \ \texttt{AND}\ \ (\texttt{NOT}\ x_a\ \ \texttt{OR}\ \ x_d\ \ \texttt{OR}\ \ x_e)$$

$$\Longleftrightarrow \begin{cases} -x_a + x_b + x_c \geq 0 \\ -x_a + x_d + x_e \geq 0 \end{cases}$$

$$(x_a\ \ \texttt{OR}\ \ x_b)\ \ \texttt{AND}\ \ (x_c\ \ \texttt{OR}\ \ x_d)\ \ \texttt{AND}\ \ \texttt{NOT}\ x_e$$

$$\Longleftrightarrow \begin{cases} x_a + x_b \geq 1 \\ x_c + x_d \geq 1 \\ x_e \leq 0 \end{cases}$$

Example: three translations (2/2)

$$(x_a \ \texttt{OR} \ x_b) \ \texttt{AND} \ (\texttt{NOT} \ x_a \ \texttt{OR} \ x_b)$$
$$\texttt{AND} \ (x_a \ \texttt{OR} \ \texttt{NOT} \ x_b) \ \texttt{AND} \ (\texttt{NOT} \ x_a \ \texttt{OR} \ \texttt{NOT} \ x_b)$$

$$\iff \begin{cases} x_a + x_b \geq 1 \\ -x_a + x_b \geq 0 \\ x_a - x_b \geq 0 \\ x_a + x_b \leq 1 \end{cases}$$

- The last system has no binary solution, just as the expression was unsatisfiable
- The first three constraints force $x_a = x_b = 1$, which violates the fourth

Summary

From CNF to linear constraints:

- Clause $\rightarrow$ constraint ≥ 1; $\texttt{OR} \rightarrow +$; $\texttt{NOT}\ x \rightarrow 1 - x$
- The sum counts the true literals of the clause: the translation is exact

In practice:

- With many negative literals, write the equivalent $\leq$ form
- Always **one constraint per clause**, counting clauses after simplification

Logical implications

Introduction: Logical implications

- In models, logical conditions are almost always **implications**: "if I choose *a* then I must choose *b*"
- We see how the implication between two variables is translated
- And then the general recipe, with compound antecedent and consequent, and how much each case costs in number of constraints

The implication between two variables

The implication between two variables

The logical implication $x_a \Rightarrow x_b$ ("if a then b") is equivalent to the boolean expression

$$\texttt{NOT}\ x_a \quad \texttt{OR} \quad x_b,$$

already in CNF, and hence to the linear constraint

$$1 - x_a + x_b \geq 1 \iff x_b - x_a \geq 0.$$

The contrapositive is the same thing

- By contraposition, $x_a \Rightarrow x_b$ also implies the contrapositive

$$\texttt{NOT}\ x_b \Rightarrow \texttt{NOT}\ x_a$$

- This is not a second constraint: its boolean expression is

$$\texttt{NOT}\,(\texttt{NOT}\ x_b)\quad \texttt{OR}\quad \texttt{NOT}\ x_a$$

  which by double negation (N) is the same $\texttt{NOT}\ x_a \quad \texttt{OR} \quad x_b$
- A single constraint imposes both readings: in the model, write the one that reads better in the application at hand

Implication is not equivalence

Observation 2: Implication is not equivalence

$x_b - x_a \geq 0$ *says "if a then b", not "a if and only if b": it admits* $x_a = 0$, $x_b = 1$.

An "if and only if" needs ***two*** *implications, that is* $x_a = x_b$, *or one implication plus an optimality argument: this is how it is written when one of the two implications follows from the sign of the coefficient in the objective.*

The general recipe

- The same recipe works when antecedent and consequent are arbitrary boolean expressions:

1. write `NOT` (antecedent) `OR` (consequent)
2. put it into CNF
3. translate with the three rules

Example: the four basic combinations (1/2)

The expressions in CNF:

Implication	Expression in CNF
x_a AND $x_b \Rightarrow x_c$	NOT x_a OR NOT x_b OR x_c
x_a OR $x_b \Rightarrow x_c$	(NOT x_a OR x_c) AND (NOT x_b OR x_c)
$x_a \Rightarrow x_b$ AND x_c	(NOT x_a OR x_b) AND (NOT x_a OR x_c)
$x_a \Rightarrow x_b$ OR x_c	NOT x_a OR x_b OR x_c

Example: the four basic combinations (2/2)

The linear constraints and their number:

Implication	Constraints	#
x_a AND $x_b \Rightarrow x_c$	$x_a + x_b - x_c \leq 1$	1
x_a OR $x_b \Rightarrow x_c$	$x_c - x_a \geq 0, \quad x_c - x_b \geq 0$	2
$x_a \Rightarrow x_b$ AND x_c	$x_b - x_a \geq 0, \quad x_c - x_a \geq 0$	2
$x_a \Rightarrow x_b$ OR x_c	$x_b + x_c - x_a \geq 0$	1

How to read the table

- A **disjunction in the antecedent** and a **conjunction in the consequent** cost **two** constraints
- A conjunction in the antecedent and a disjunction in the consequent cost **one**
- This is why the aggregated constraint $\sum_j x_{jm} \leq k\, y_m$ and the disaggregated one $x_{jm} \leq y_m$ have different costs in number of rows — and different strengths in the relaxation

Example: an antecedent with three literals

$$x_a \quad \texttt{AND} \quad \texttt{NOT}\ x_b \quad \texttt{AND} \quad x_c \Rightarrow \texttt{NOT}\ x_d$$

$$\iff \quad \texttt{NOT}\ x_a \quad \texttt{OR} \quad x_b \quad \texttt{OR} \quad \texttt{NOT}\ x_c \quad \texttt{OR} \quad \texttt{NOT}\ x_d$$

$$\iff \quad x_a - x_b + x_c + x_d \leq 2$$

- Three negative literals in the clause: we use the $\leq$ form

Summary

Implications:

- $x_a \Rightarrow x_b$ is `NOT` x_a `OR` x_b, that is $x_b - x_a \geq 0$
- The contrapositive is the same constraint
- Implication is not equivalence: "if and only if" needs two implications

General case:

- `NOT` (antecedent) `OR` (consequent), CNF, translation
- `OR` in the antecedent and `AND` in the consequent cost two constraints

Splitting an implication: when it is allowed

Introduction: Splitting an implication

- It is often convenient to **split** a compound implication into simpler ones
- There are two valid splits and one that is **not** valid
- A wrong split produces a model that may exclude the optimum: let us see why

Two valid splits and one that is not

Valid:

$$(x_a \quad \texttt{OR} \quad x_b) \Rightarrow x_c \iff (x_a \Rightarrow x_c) \quad \texttt{AND} \quad (x_b \Rightarrow x_c) \quad \text{(S1)}$$

$$x_a \Rightarrow (x_b \quad \texttt{AND} \quad x_c) \iff (x_a \Rightarrow x_b) \quad \texttt{AND} \quad (x_a \Rightarrow x_c) \quad \text{(S2)}$$

Not valid: the analogous split with a **conjunctive** antecedent

$$(x_a \quad \texttt{AND} \quad x_b) \Rightarrow x_c \quad \textbf{not} \iff \quad (x_a \Rightarrow x_c) \quad \texttt{AND} \quad (x_b \Rightarrow x_c)$$

The counterexample, and which way it fails

Observation 3: The counterexample, and which way it fails

*With $x_a = 1$, $x_b = 0$, $x_c = 0$ the implication $(x_a \ \ \texttt{AND} \ \ x_b) \Rightarrow x_c$ is **true** — its antecedent is $\texttt{FALSE}$ — but $x_a \Rightarrow x_c$ is **false**.*

*The conjunction of the two split implications is therefore **strictly stronger** than the original: it cuts off solutions the problem allows. Getting this wrong does not produce a "slightly different" model, it produces a model that may exclude the optimum.*

The shortest route

- The two valid splits are often the shortest route
- In the table of the four combinations, the rows with two constraints follow directly from (S1) or (S2)
- Without going through De Morgan and distributivity

Summary

Splits:

- `OR` antecedent: split into two implications (S1)
- `AND` consequent: split into two implications (S2)
- `AND` antecedent: **cannot** be split

Why it matters:

- The wrong split is strictly stronger: it cuts feasible solutions and may exclude the optimum

Counting: at most one, at least one, exactly one

Introduction: Counting

- Many conditions are **cardinality** conditions: "at least one", "at most one", "exactly one", "at least p"...
- On a set I of binary variables they are **already linear** and need no CNF
- We will see that one cardinality constraint can beat many clauses

Cardinality conditions (1/2)

Condition	Constraint	Note
at least one	$\sum_{i \in I} x_i \geq 1$	it is the clause $\bigvee_{i \in I} x_i$: *set covering*
at most one	$\sum_{i \in I} x_i \leq 1$	*set packing*; equivalent to the $\binom{\lvert I \rvert}{2}$ clauses `NOT` x_i `OR` `NOT` x_j, but in a single constraint and with a tighter relaxation
exactly one	$\sum_{i \in I} x_i = 1$	*set partitioning*: the two above together

Cardinality conditions (2/2)

Condition	Constraint	Note
at least p	$\sum_{i \in I} x_i \geq p$	in CNF it would take $\binom{\lvert I \rvert}{\lvert I \rvert - p + 1}$ clauses
at most p	$\sum_{i \in I} x_i \leq p$	in CNF it would take $\binom{\lvert I \rvert}{p+1}$ clauses
none	$\sum_{i \in I} x_i \leq 0$	that is, $x_i = 0$ for every i

One cardinality constraint beats many clauses (1/2)

- "At most one out of three" can be written as three clauses:

$$\texttt{NOT } x_1 \quad \texttt{OR} \quad \texttt{NOT } x_2 \iff x_1 + x_2 \leq 1$$
$$\texttt{NOT } x_1 \quad \texttt{OR} \quad \texttt{NOT } x_3 \iff x_1 + x_3 \leq 1$$
$$\texttt{NOT } x_2 \quad \texttt{OR} \quad \texttt{NOT } x_3 \iff x_2 + x_3 \leq 1$$

- or as the single

$$x_1 + x_2 + x_3 \leq 1$$

One cardinality constraint beats many clauses (2/2)

- The two systems have the **same** binary solutions, but not the same relaxation
- $x_1 = x_2 = x_3 = 1/2$ satisfies the three clauses and violates the aggregated constraint
- The compact version is shorter **and** stronger: a case with no trade-off
- When counting is possible, count; CNF is for conditions that are not cardinalities

Summary

Cardinality:

- At least one (≥ 1, set covering), at most one (≤ 1, set packing), exactly one ($= 1$, set partitioning)
- Already linear: no CNF needed

Counting pays off:

- One constraint instead of many clauses
- "At most one" counted has a tighter relaxation than the pairwise clauses

Logical operators and set operators

Introduction: Logical and set operators

- Logical operators have a counterpart in **sets**
- The equivalences of logic become set identities
- This gives a second proof of the same rules, with Venn diagrams

From logic to sets

- Associate with every binary variable a subset of a universe U
- $x_a = 1$ means "the chosen element lies in A"

Logic	Sets	
x_a `AND` x_b	$A \cap B$	intersection
x_a `OR` x_b	$A \cup B$	union
`NOT` x_a	$\overline{A}$	complement

Sapienza Università di Roma

The equivalences become set identities

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \quad \text{(D1)}$$
$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C) \quad \text{(D2)}$$
$$\overline{A \cup B} = \overline{A} \cap \overline{B} \quad \text{(M1)}$$
$$\overline{A \cap B} = \overline{A} \cup \overline{B} \quad \text{(M2)}$$
$$A \cup (A \cap B) = A \quad \text{(A1)}$$
$$A \cap (A \cup B) = A \quad \text{(A2)}$$

- Those who prefer Venn diagrams get a second proof; those who prefer enumeration have the first
- Each checks the other

Summary

Logic and sets:

- `AND` $\leftrightarrow \cap$, `OR` $\leftrightarrow \cup$, `NOT` $\leftrightarrow$ complement
- Distributivity, De Morgan and absorption also hold for sets
- Venn and enumeration: two proofs that check each other

Checking the translation, not trusting it

Introduction: Checking the translation

- A translation into linear constraints must be **checked**, not trusted
- With few variables the check is an enumeration: it is the proof by cases
- The module `booleane.py` performs this check and all the rest of the chapter's work

The operational criterion

- A translation into linear constraints is correct when, for **every** binary assignment of the variables involved:

  the expression is true $\iff$ all the constraints are satisfied

- With few variables it is an enumeration of 2^n cases, done in an instant
- It is the proof, by cases, of the result
- The module `python/booleane.py` performs it: every translation of the exercises is checked this way

The module `booleane.py`

The module also does the rest of the chapter's work:

1. it eliminates implications ($A \Rightarrow B$ becomes `NOT` A `OR` B)
2. it pushes negations onto the literals with De Morgan
3. it distributes `OR` over `AND`
4. it discards tautological and absorbed clauses
5. it applies the three translation rules

- It is not a SAT solver: it is a line-by-line transcription of this chapter's rules

Summary

Checking:

- Correct $\iff$ for every assignment: expression true $\iff$ constraints satisfied
- With n variables: 2^n cases, an instant

`booleane.py`:

- Implications, De Morgan, distributivity, simplification, translation
- The transcription of the chapter's rules

Five exercises

Introduction: Five exercises

- We put the recipe into practice on five groups of logical conditions between **projects**
- Direct, negated and compound implications, "at least two of", splits
- For each condition: CNF expression, linear constraints, number of constraints

The variables

- In all five exercises the projects are numbered
- For every project p the binary variable is

$$x_p = \begin{cases} 1 & \text{if project } p \text{ is chosen} \\ 0 & \text{otherwise} \end{cases}$$

- For each condition we ask for: the CNF expression, the linear constraints and their number
- The equivalence is checked by enumerating the assignments

Exercise 1: direct implications

Ten projects. Model as linear constraints:

1. if project 2 is chosen then project 3 is chosen;
2. if project 2 is chosen then project 4 is not chosen;
3. if projects 1 and 6 are chosen then project 7 is chosen;
4. if project 1 or project 6 is chosen then project 8 is chosen;
5. if projects 2 and 3 are chosen then project 9 is not chosen;
6. if project 2 or project 3 is chosen then project 10 is not chosen.

Exercise 2: negated antecedents and consequents

Ten projects. Model:

1. if project 3 is not chosen then project 2 is chosen;
2. if project 4 is not chosen then project 2 is not chosen;
3. if project 7 is chosen then projects 1 and 6 are chosen;
4. if project 8 is chosen then project 1 or project 6 is chosen;
5. if project 9 is not chosen then projects 2 and 3 are chosen;
6. if project 10 is not chosen then project 2 or project 3 is chosen.

Exercise 3: compound antecedents and consequents (1/2)

Eight projects. Model:

1. x_7 OR $x_3 \Rightarrow x_1$ AND x_2;
2. x_1 AND x_6 AND $x_7 \Rightarrow x_8$;
3. x_5 AND x_2 AND NOT $x_4 \Rightarrow$ NOT x_3;

Exercise 3: compound antecedents and consequents (2/2)

4. $(x_1 \texttt{ OR } x_4) \texttt{ AND } x_6 \Rightarrow x_2 \texttt{ AND } (x_5 \texttt{ OR } x_7)$;
5. $(x_2 \texttt{ OR } x_5) \texttt{ AND NOT } x_8 \Rightarrow x_3 \texttt{ OR NOT } x_6$;
6. $(x_1 \texttt{ OR } x_4) \texttt{ AND } (x_2 \texttt{ OR } x_5) \texttt{ AND NOT } x_8 \Rightarrow x_3 \texttt{ AND } (\texttt{NOT } x_6 \texttt{ OR } x_7)$.

Exercise 4: "at least two of"

Nine projects. Model:

1. if project 4 is chosen then at least two of 1, 2, 3;
2. if at least two of 6, 7, 8 then project 5;
3. if project 4 is not chosen then at least two of 1, 2, 3, 9;
4. if project 8 is chosen then (1 and 6) or (1 and 7) or (2 and 6);
5. if at least two of 1, 3, 5 then project 9 is not chosen;
6. if (1 and 2) or (3 and 4) then project 5.

"At least two" can also be written by counting

Observation 4: "At least two" can also be written by counting

The condition "at least two of 1, 2, 3" is also $x_1 + x_2 + x_3 \geq 2$. *As the consequent of the implication governed by* x_4 *it becomes*

$$x_1 + x_2 + x_3 \geq 2x_4 :$$

a single constraint, against the three clauses of item 1. The two systems have the same 16 *binary solutions — checked by enumeration — but not the same relaxation.*

The counted form is stronger (1/2)

- This time it is the counted form that is **stronger**
- Summing the three clauses $x_i + x_j \geq x_4$ gives only

$$x_1 + x_2 + x_3 \geq \frac{3}{2} x_4,$$

  weaker than $\geq 2x_4$
- Conversely, with $0 \leq x_i \leq 1$, the counted form implies each clause

The counted form is stronger (2/2)

- On the problem

$$\begin{aligned} \max \quad & x_1 + x_2 + x_3 + 3x_4 \\ \text{subject to} \quad & x_1 + x_2 + x_3 + 2x_4 \leq 3 \end{aligned}$$

  whose integer optimum is $z(MILP) = 3$
- the three clauses give $z(LP^+) = 27/7 \approx 3.86$
- the counted constraint gives $z(LP^+) = 15/4 = 3.75$: closer to the integer optimum

The moral

- The moral is not "counting is always better"
- "At most one" counted was stronger than the pairwise clauses, here "at least two" counted is again stronger
- But it does not always go that way: an aggregated sum can perfectly well be **weaker** than the inequalities it summarises
- Two formulations with the same integer set must be compared on their relaxation, **by solving it**, and not by eye

Exercise 5: splits

Ten projects. Model:

1. x_1 OR $x_2 \Rightarrow x_3$;
2. $x_4 \Rightarrow x_5$ AND x_6;
3. x_1 OR $x_2 \Rightarrow x_3$ AND x_4;
4. x_1 AND $x_2 \Rightarrow x_3$;
5. x_5 OR $x_6 \Rightarrow$ NOT x_7;
6. NOT x_8 OR NOT $x_9 \Rightarrow x_{10}$.

Summary

The exercises:

- One binary per project; for each condition: CNF, constraints, count
- Direct, negated and compound implications, "at least two of", splits

"At least two":

- Three clauses or one counted constraint $x_1 + x_2 + x_3 \geq 2x_4$
- Same integer solutions, different relaxation: compare by solving

Logical constraints inside an optimisation model

Introduction: Logical constraints inside a model

- Logical constraints rarely live alone: they are added to a model that **optimises** something
- We use the 10 projects of Exercise 1, with revenues, costs and a budget
- We look at the model, the instance, the solution and the `gurobipy` code

The data

- The $n = 10$ projects of Exercise 1, with revenues r_p and costs c_p:

project p	1	2	3	4	5	6	7	8	9	10
revenue r_p	9	7	4	8	3	6	2	5	7	6
cost c_p	4	3	2	4	2	3	1	3	4	3

- Budget $b = 14$

Project selection with logical constraints

Project selection with logical constraints

$$\max \quad \sum_{p=1}^{n} r_p \, x_p$$

$$\text{subject to} \quad \sum_{p=1}^{n} c_p \, x_p \leq b,$$

the 8 constraints of exercise ??,

$$x_p \in \{0, 1\}, \qquad \forall p \in \{1, 2, \ldots, n\}.$$

Description

- The **objective** adds up the revenues of the chosen projects
- The first constraint, a single one, is the **budget**
- The **eight constraints** are the linear translations of the six implications of Exercise 1: they add no variables, only rows
- The **domains** are the n binary conditions, one per project

The model on the instance

Project selection, ten binaries

$$
\begin{aligned}
\max\ & 9x_2+7x_3+4x_4+8x_5+3x_6+6x_7+2x_8+5x_9+7x_{10}+6x_{11} \\
\text{subject to}\ & 4x_2+3x_3+2x_4+4x_5+2x_6+3x_7+x_8+3x_9+4x_{10}+3x_{11} \le 14 \\
& -x_3+x_4 \ge 0 \\
& x_3+x_5 \le 1 \\
& x_2+x_7-x_8 \le 1 \\
& -x_2+x_9 \ge 0 \\
& -x_7+x_9 \ge 0 \\
& x_3+x_4+x_{10} \le 2 \\
& x_3+x_{11} \le 1 \\
& x_4+x_{11} \le 1 \\
& x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11} \in \{0,1\}
\end{aligned}
$$

The solution

- Without the logical constraints the optimum is 30
- With them the optimum drops to $z(MILP) = 28$
- Projects $1, 2, 3, 5, 8$ are chosen, of total cost 14: the budget is tight
- The relaxation is $z(LP^+) = 29$

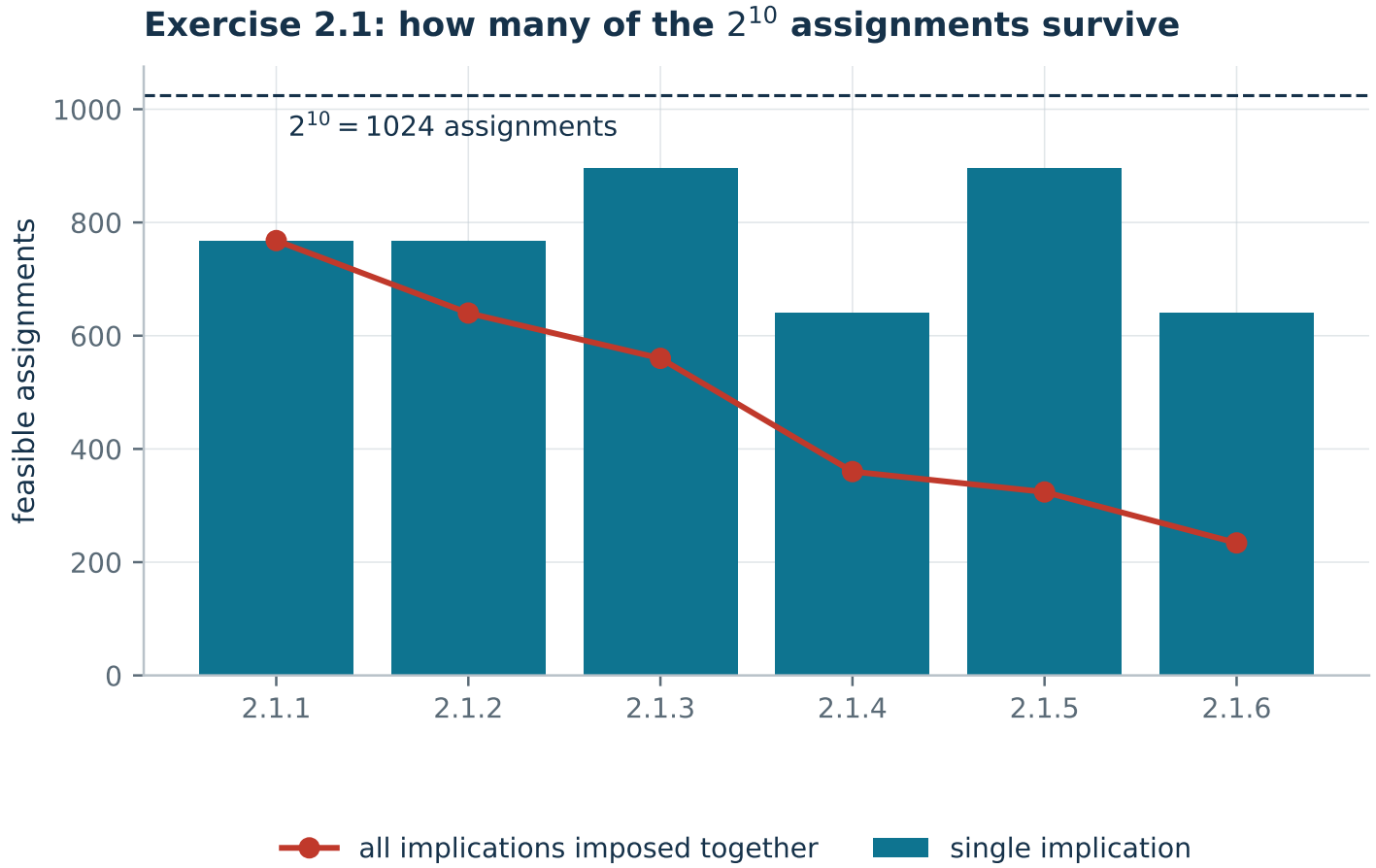
The six implications hold:

- 2 is chosen and so is 3; 4 is not chosen; 1 and 6 are not both chosen
- 1 is chosen and so is 8; 2 and 3 are chosen and 9 is not; 2 is chosen and 10 is not

How much six implications cut

- The $2^{10} = 1024$ assignments of the ten binaries drop to 234 once all six implications are imposed: less than a quarter
- None of these implications, on its own, cuts more than half the space; together they cut three quarters
- This is why a model is written one constraint at a time but judged as a whole

The surviving assignments



- Bars: each implication on its own; line: all the implications imposed up to that point

The model in gurobipy

- Next to `gurobipy` we use the small teaching module `booleane.py`, distributed with the course material
- `V` creates a variable; `AND`, `OR`, `NOT` and `IMP` compose it: the formula is written as by hand
- `cnf` converts it into conjunctive normal form
- `vincolo` turns a clause into the coefficients, the direction and the right-hand side of the inequality

The code (1/3): the implications

```
from booleane import cnf, vincolo
from booleane import IMP, AND, OR, NOT, V

x = {p: V(f"x{p}") for p in range(1, 11)}
implications = [
    IMP(x[2], x[3]),
    IMP(x[2], NOT(x[4])),
    IMP(AND(x[1], x[6]), x[7]),
    IMP(OR(x[1], x[6]), x[8]),
    IMP(AND(x[2], x[3]), NOT(x[9])),
    IMP(OR(x[2], x[3]), NOT(x[10]))]
```

The code (2/3): variables, objective, budget

```
import gurobipy as gp
from gurobipy import GRB

P = range(1, 11)
m = gp.Model("project_selection")
xv = m.addVars(P, vtype=GRB.BINARY, name="x")
m.setObjective(gp.quicksum(r[p]*xv[p] for p in P),
               GRB.MAXIMIZE)
m.addConstr(gp.quicksum(c[p]*xv[p] for p in P) <= b)
```

The code (3/3): one clause, one constraint

```
# one clause, one constraint
for formula in implications:
    for clause in cnf(formula):
        coef, sense, rhs = vincolo(clause)
        lhs = gp.quicksum(k*xv[int(n[1:])]
                          for n, k in coef.items())
        m.addConstr(lhs <= rhs if sense == "<="
                        else lhs >= rhs)
m.optimize()
```

- All the code does is the procedure of this chapter, written once
- The two bounds of this course — the primal one and the dual one — reproduce, in teaching form, the two sides of the **sandwich** that a MIP solver tries to close

Summary

The model:

- Project selection: objective, budget, eight clauses, binary domains
- Logical constraints add rows, not variables

The results:

- Optimum from 30 down to $z(MILP) = 28$ (projects $1, 2, 3, 5, 8$), $z(LP^+) = 29$
- From 1024 to 234 assignments: together, the implications cut three quarters

What this chapter has put in your hands

Introduction: What this chapter has put in your hands

- We collect the recipe in a few lines
- And we name the three shapes that recur most often in models

The recipe, always the same

- One binary per elementary fact
- Bring the condition into CNF — a conjunction of ORs
- Every clause becomes a linear inequality

- The `AND` costs nothing: constraints are already in AND with each other
- The `OR` becomes a sum ≥ 1; the `NOT` becomes $1 - x$
- The implication $x \Rightarrow y$ becomes $x \leq y$
- The "if and only if" asks for both inequalities

Three shapes with a name

- **Set covering**: $\sum_{j \in S} x_j \geq 1$ (at least one)
- **Set packing**: $\sum_{j \in S} x_j \leq 1$ (at most one)
- **Set partitioning**: $\sum_{j \in S} x_j = 1$ (exactly one)

Recognising them in a statement is half of the modelling work.

Summary (1/2)

Language:

- Propositions $\leftrightarrow$ binaries; literals, clauses, CNF
- Distributivity, De Morgan, absorption to bring into CNF

Translation:

- Clause $\rightarrow$ constraint ≥ 1, `OR` $\rightarrow +$, `NOT` $x \rightarrow 1 - x$
- Implication: `NOT` (antecedent) `OR` (consequent)
- Valid splits only for `OR` antecedent and `AND` consequent

Summary (2/2)

Counting:

- Set covering, packing, partitioning: already linear
- Formulations with the same integer set are compared on the relaxation

Checking:

- Enumerating the 2^n assignments: the proof by cases
- In models, logical constraints add rows, and together they cut a lot