SAPIENZA
UNIVERSITÀ DI ROMA

Artificial Intelligence

2023/2024 Prof: Sara Bernardini

Lab 4:
A*, α-β pruning and CSP

Francesco Argenziano
email: argenziano@diag.uniroma1.it

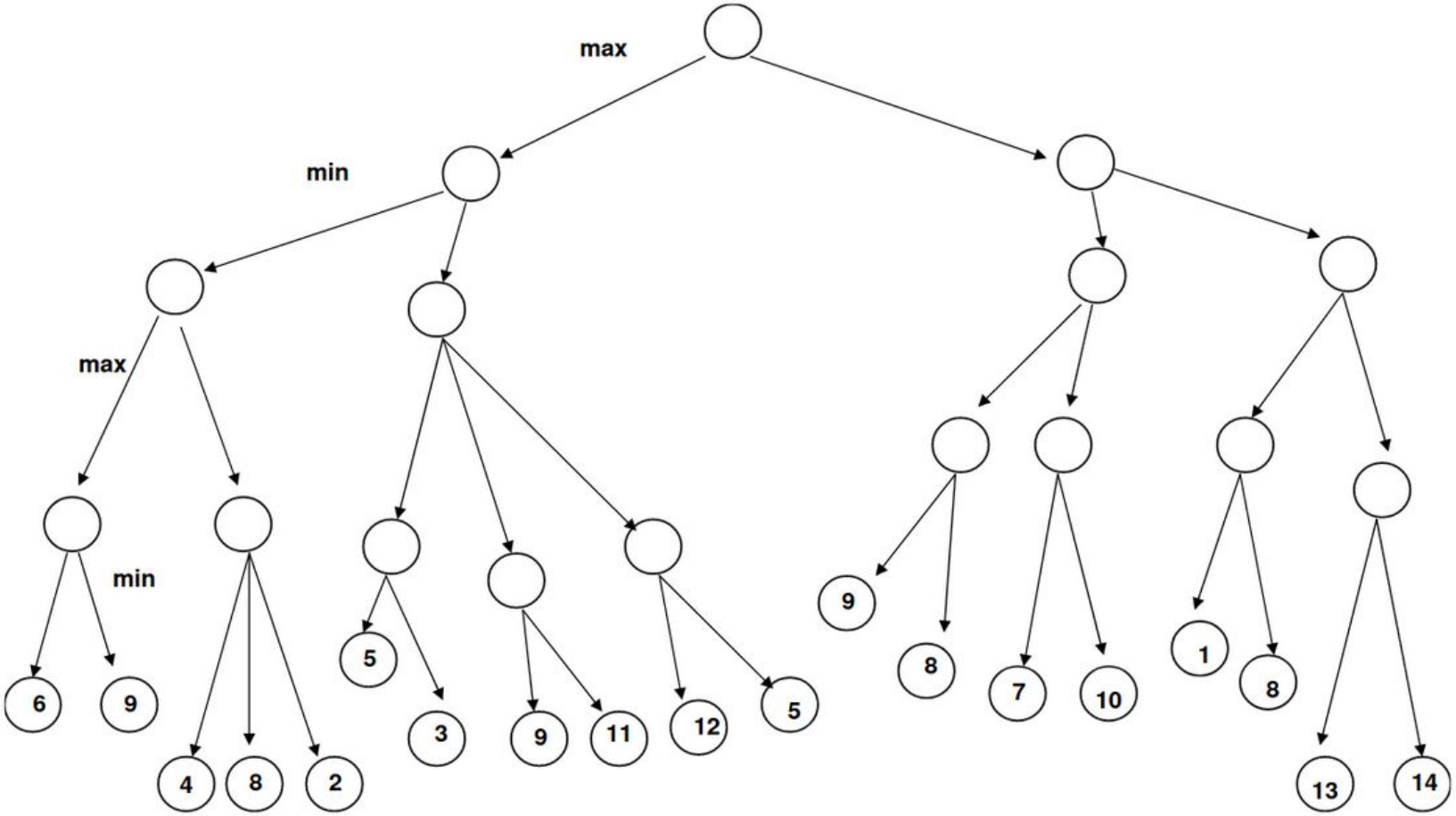
*The slides have been prepared using the textbook material available on the web, and the slides of the previous editions of the course by Prof. Luigia Carlucci Aiello, Prof. Daniele Nardi and Dott. Fabio Previtali.

Example: Crossing the river

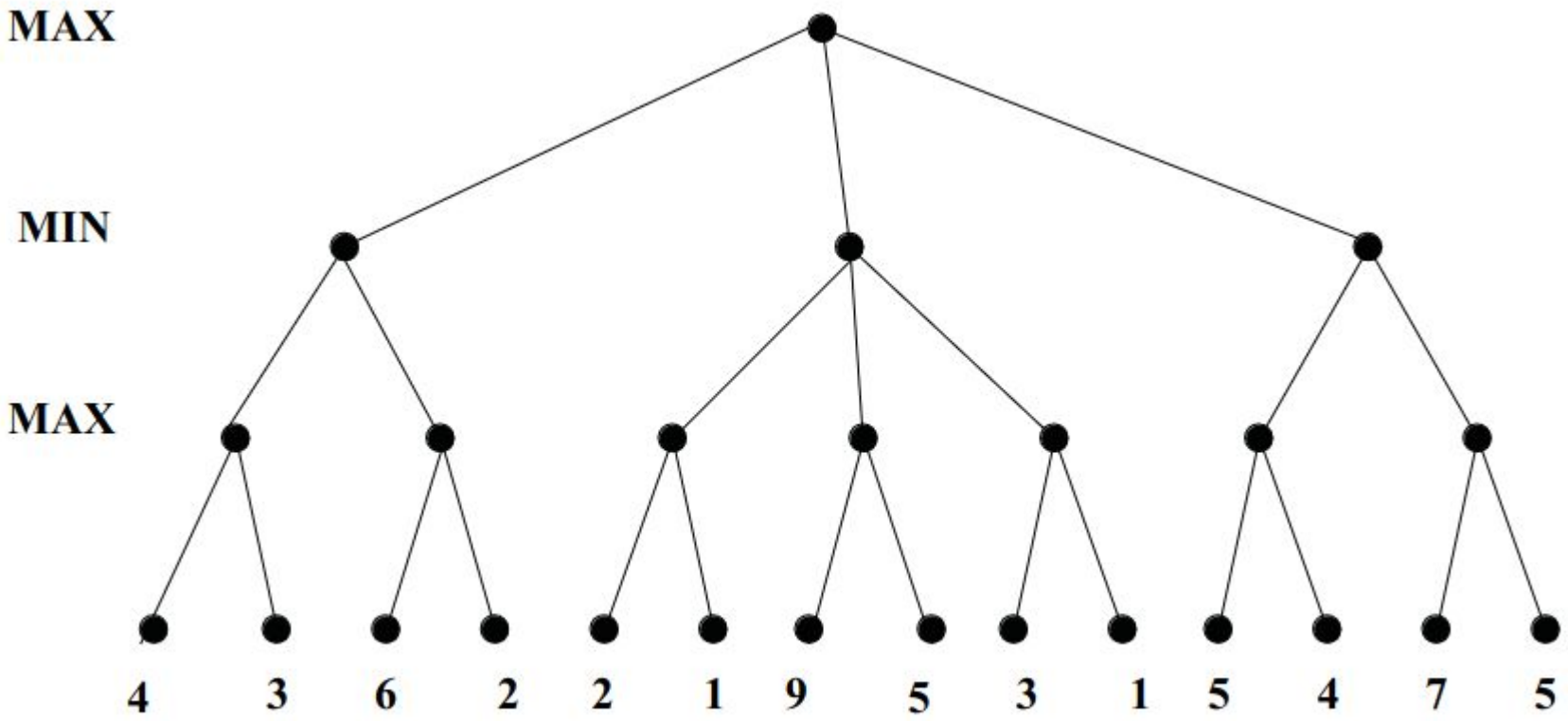
A man has a **wolf**, a **sheep** and a **cabbage**. He is on a river bench with a **boat**, whose maximum **load** for a single trip is the man **plus one of his 3 goods**. The man wants to cross the river with his goods, but he must avoid that - when he is far away - **the wolf eats the sheep** and that, **the sheep eats the cabbage**. How can the man reach is goal?

1. Characterize the **state space**
2. Specify the **operators**
3. Find a minimal sequence of moves to **solve the problem**
4. Find a good **heuristics** to be used by A∗
5. Draw the **search tree generated by A∗**. For each node indicate: the number (state), f, g, and h and an integer indicating the expansion order

Example: α-β pruning 1



Example: α-β pruning 2



Hands-on with Z3

What is Z3?

Z3 is a state-of-the art theorem prover from Microsoft Research. It can be used to check the satisfiability of logical formulas over one or more theories.

- ➢ we are just going to use it for CSP exercises

How to install it:

If you have Ubuntu:

- ➢ “sudo apt-get -y install z3”

What is Z3?

Z3 is a state-of-the art theorem prover from Microsoft Research. It can be used to check the satisfiability of logical formulas over one or more theories.

- ➢ we are just going to use it for CSP exercises

How to install it:

If you have Ubuntu:

- ➢ “sudo apt-get -y install z3”

if you don’t have Ubuntu:

- ➢ download Ubuntu

Syntax

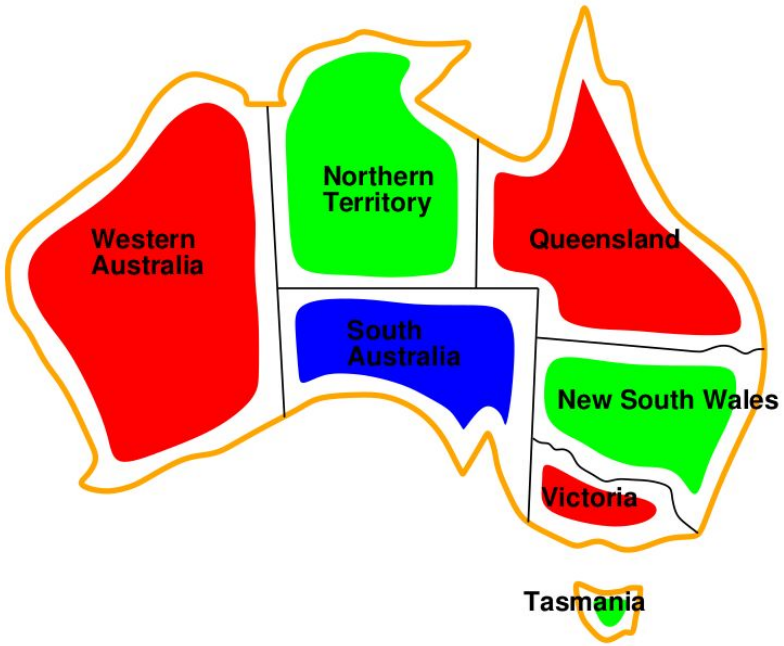
Statement	Description
General	
`(check-sat)`	Checks whether the CSP defined up to this point is satisfiable.
`(declare-const var Int)`	Declares a new variable with name `var`.
`(assert E)`	Adds boolean expression `E` as constraint.
`(get-value (E))`	Prints the value of `E`, where `E` can be an arbitrary expression such as constant, variable, function, or mathematical or boolean combination thereof (must occur after `(check-sat)`).
`(get-model)`	Prints all variable assignments (must occur after `(check-sat)`).
`(echo "message")`	Prints `message` to the console.
`; This is a comment`	Commenting.

Syntax

Mathematical Expressions	
c	Constants $c \in \mathbb{Z}$.
`var`	Evaluates to the value of variable `var`.
`(`$\circ$ E_1 ... E_n`)`	Evaluates to $E_1 \circ E_2 \circ \cdots \circ E_n$, where $\circ$ can be any of `+`, `-`, and `*`.

Boolean Expressions	
`true`	Constant for true.
`false`	Constant for false.
`(not` E`)`	Negation of the boolean expression E.
`(and` E_1 ... E_n`)`	Conjunction over the boolean expressions E_1 to E_n.
`(or` E_1 ... E_n`)`	Disjunction over the boolean expressions E_1 to E_n.
`(`$\circ$ E_1 ... E_n`)`	Is true iff for the evaluation of the expressions E_1 to E_n, it holds that $E_1 \circ E_2$ and $E_2 \circ E_3$, ..., and $E_{n-1} \circ E_n$, where $\circ$ can be any of `<`, `<=`, `>`, `>=`, and `=`.
`(distinct` E_1 ... E_n`)`	Is true iff every expression E_1 to E_n evaluates to a different value.

Example: Australia Graph Coloring



- **Variables:** WA, NT, SA, Q, NSW, V, T.
- **Domains:** Red, green, blue.
- **Constraints:** Adjacent states must have different colors.

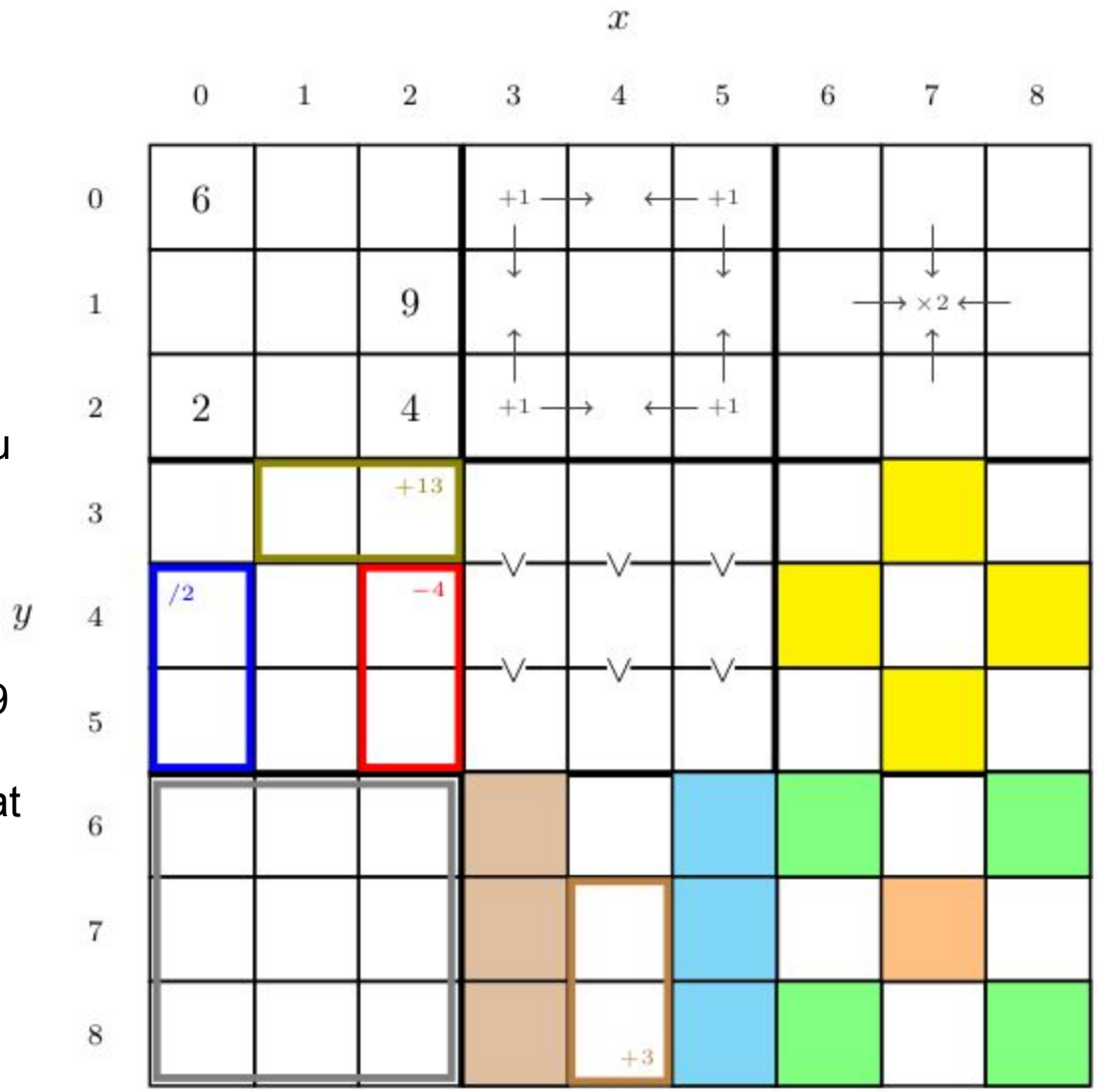
Example: Generalized Sudoku

Goal of this exercise is to model in Z3 this Generalized Sudoku puzzle as a CSP.

Why Generalized?
Because in addition to standard sudoku constraints:
- no duplicates in the same 3x3 square
- no duplicates in the same row
- no duplicates in the same column
- all cell has a number between 1 and 9

there are also additional constraints that differ from (3x3) square to square.

We refer with <i, j> to the cell at x=i and y=j

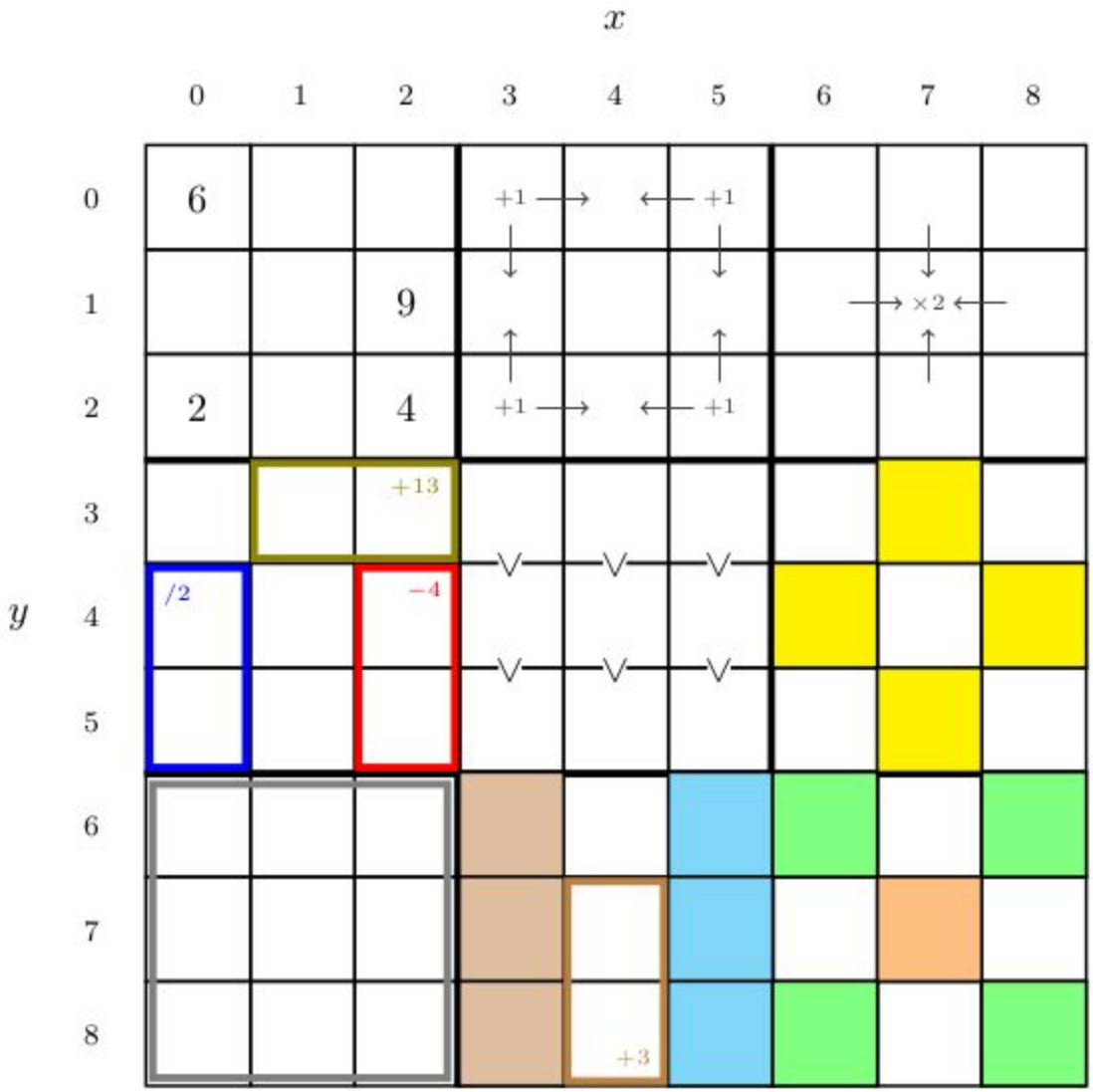


Example: Generalized Sudoku

Top left square:
typical sudoku, some numbers are already in the grid.

Top middle square:
for every corner cell of this square, one of the horizontally or vertically adjacent cells must equal the value plus 1.
For example, for the cell <3, 0>, if <3, 0> = 4, then either <3, 1> = 5 or <4, 0> = 5.

Top right square:
the sum of the indicated cells must be equal to twice the value of the center cell. In other words:
<7, 0> + <6, 1> + <7, 2> + <8, 1> =
= 2 × <7, 1>

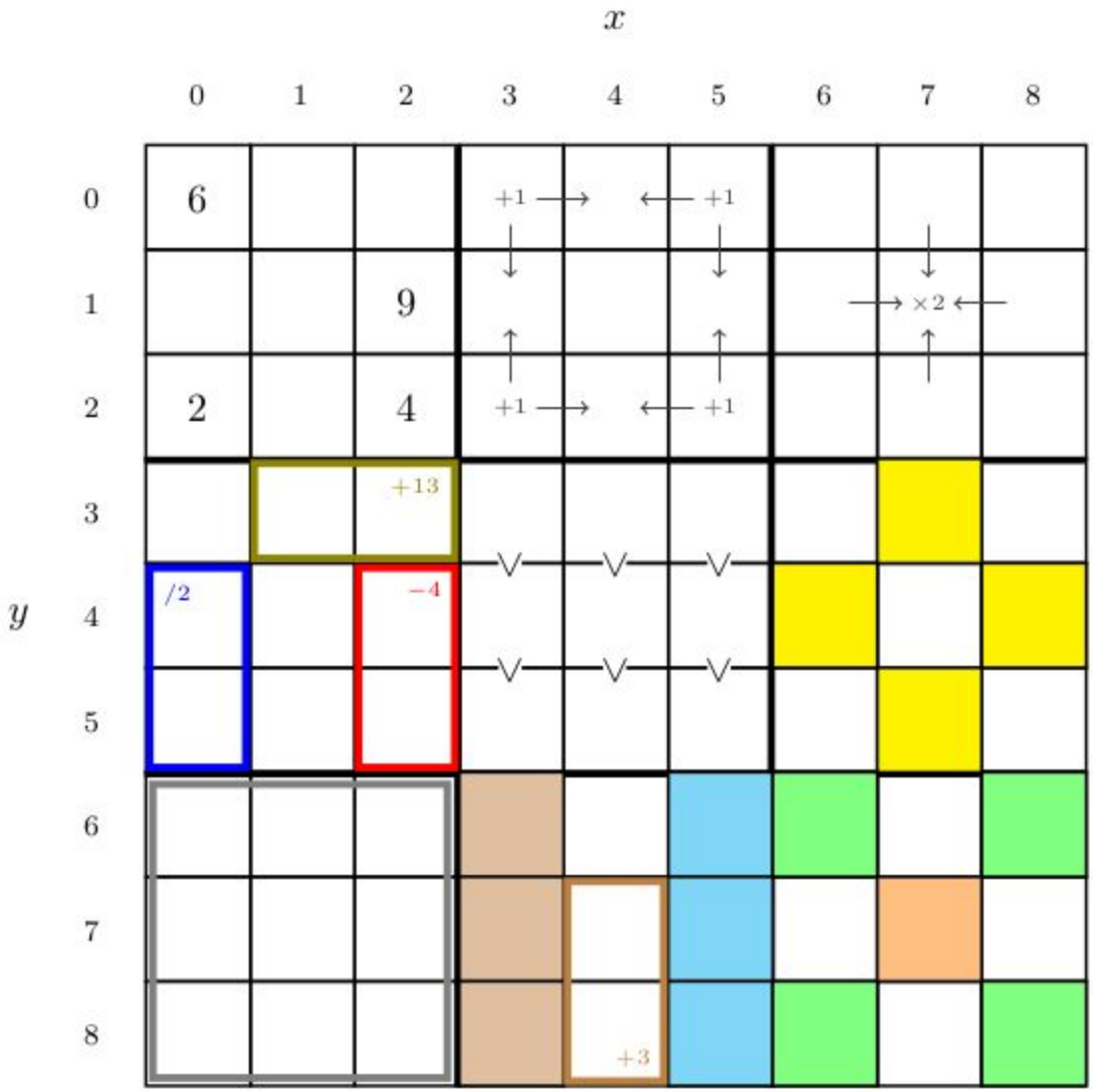


Example: Generalized Sudoku

Middle left square:
numbers must comply with
(a) <1, 3> + <2, 3> = 13
(b) <0, 5> / <0, 4> = 2
(c) <2, 5> − <2, 4> = 4

Middle square:
numbers must comply with
(a) <3, 3> **>** <3, 4> **>** <3, 5>
(b) <4, 3> **>** <4, 4> **>** <4, 5>
(c) <5, 3> **>** <5, 4> **>** <5, 5>

Middle right square:
exactly one of the yellow cells must contain a value smaller than 5, the other yellow cells must contain a value greater or equal to 5.



Example: Generalized Sudoku

Bottom left square:
the sum of all rows in this square must be equal

Bottom middle square:
(a) Multiplying the sums of the two indicated columns gives an odd number:
(<3, 6> + <3, 7> + <3, 8>) × (<5, 6> + <5, 7> + <5, 8>) must be odd.
(b) The numbers must comply with the arithmetic expressions drawn in the figure:
<4, 7> + <4, 8> = 3.

Bottom right square:
the values of the green cells must be either all odd or all even.
Moreover, if the green cells contain odd numbers, then the orange cell must contain an even number. If the green cells contain even numbers, then the orange cell must contain an odd number.