In [1]:

```
"""
    Helper functions used troughout the rest of this notebook
"""
import time

from cpmpy.transformations.normalize import toplevel_list
from factory import *
from read_data import get_data

import numpy as np
np.set_printoptions(linewidth=90)

instance = "Benchmarks/Instance1.txt"
data = get_data(instance)

nurses = data.staff['name'].tolist()

weeks = [f"Week {i+1}" for i in range(data.horizon // 7)]
weekdays = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
days_name = [f"{weekdays[i % 7]} {1+(i // 7)}" for i in range(data.horizon)]
max_consequtive_constraints = []
weekdays_cons = []

import matplotlib
import matplotlib.colors as mcolors

def color_shift(shift):
    cmap =list(mcolors.TABLEAU_COLORS.values())
#     cmap = ["yellow", "blue","red", "orange", "cyan"]
    if shift is None or shift == '':
        return 'background-color: white'
    return f"background-color: {cmap[factory.shift_name_to_idx[shift]]}"

def highlight_cell(cells):
    def do_function(x):
        return [f'background: {cells[(x.name,i)]}'
         if (x.name,i) in cells else "" for i,_ in enumerate(x)]
    return do_function
```

```python
def highlight_cell_border(cells):
    def do_function(x):
        return [f'border: 5px {cells[x.name,i]}'
         if (x.name,i) in cells else "" for i,_ in enumerate(x)]
    return do_function

def highlight_cover(covers):
    def do_function(x):
        borders = []
        for i,_ in enumerate(x):
            s = ''
            if i in covers:
                s += 'border-left: 5px solid red; border-right:5px solid red;'
                if x.name == nurses[0]:
                    s += 'border-top: 5px solid red;'
                if x.name.startswith("Cover"):
                    s += 'border-bottom: 5px solid red;'
            borders += [s]
        return borders
    return do_function

def highlight_row(nurse_id, windows):
    def do_function(x):
        borders = []
        for id, window in zip(nurse_id, windows):
            for i,_ in enumerate(x):
                s = ''
                if i in window:
                    if x.name == nurses[id]:
                        s = 'border-top: 5px solid red; border-bottom:5px solid red;'
                        if i == window[0]:
                            s += 'border-left: 5px solid red;'
                        if i == window[-1]:
                            s += 'border-right: 5px solid red;'
                borders += [s]
        return borders
    return do_function

def highlight_weekends(nurse_id):
    def do_function(x):
```

```
        for id in nurse_id:
            for i,_ in enumerate(x):
                s = ''
                if i%7 in [5,6]:
                    if x.name == nurses[id]:
                        s = 'border-top: 5px solid indigo; border-bottom:5px solid indigo;'
                        if i%7 == 5:
                            s += 'border-left: 5px solid indigo;'
                        if i%7 == 6:
                            s += 'border-right: 5px solid indigo;'
                borders += [s]
        return borders
    return do_function

def highlight_changes(nurse_view, opt_sol):
    style = visualize(nurse_view.value())
    cells = {}
    for i in range(np.size(opt_sol,0)):
        for j in range(np.size(opt_sol,1)):
            if nurse_view[i,j].value() != opt_sol[i,j]:
                cells.update({(nurses[i], j): "solid lawngreen"})
    #cells = {(nurses[i], j): "solid lawngreen" for j in range(np.size(opt_sol,0)) for i in range(n
    style.apply(highlight_cell_border(cells), axis=1)
    return style

def visualize_constraints(subset, nurse_view, do_clear=True):
    if do_clear:
        nurse_view.clear()
    style = visualize(nurse_view.value())
    cells = {(cons.cell[0], cons.cell[1]) : cons.cell[2] for cons in subset if hasattr(cons,"cell")
    style.apply(highlight_cell(cells), axis=1)

    covers = {cons.cover for cons in subset if hasattr(cons, 'cover')}
    style.apply(highlight_cover(covers), axis=1)
    nurse_id = {cons.nurse_id for cons in subset if hasattr(cons, 'nurse_id') and hasattr(cons, 'wi
    windows = {cons.window for cons in subset if hasattr(cons, 'window')}
    style.apply(highlight_row(nurse_id, windows), axis=1)
    nurse_id = {cons.nurse_id for cons in subset if hasattr(cons, 'nurse_id') and hasattr(cons, 'ma
    style.apply(highlight_weekends(nurse_id), axis=1)
    return style
```

```python
def visualize_step(step, nurse_view):
    E,S,N = step
    print(f"Propagating constraint: {next(iter(S))}")
    if any(len(vals) == 0 for vals in N.values()):
        # found UNSAT
        return visualize_constraints(S, nurse_view, do_clear=False)
    else:
        for v in E:
            if E[v] > N[v]:
                # derived something here
                assert len(N[v]) <=  1, "only allow assigments here..."
                # hacky way to find index
                r = int(v.name.split(",")[0].split('[')[1])
                c = int(v.name.split(",")[1].split(']')[0])
                nurse_view[r,c]._value = next(iter(N[v]))

    return visualize_constraints(S, nurse_view, do_clear=False)

#cons.max_weekends = max_weekends
```

Explainable Constraint Solving - A Hands-On Tutorial

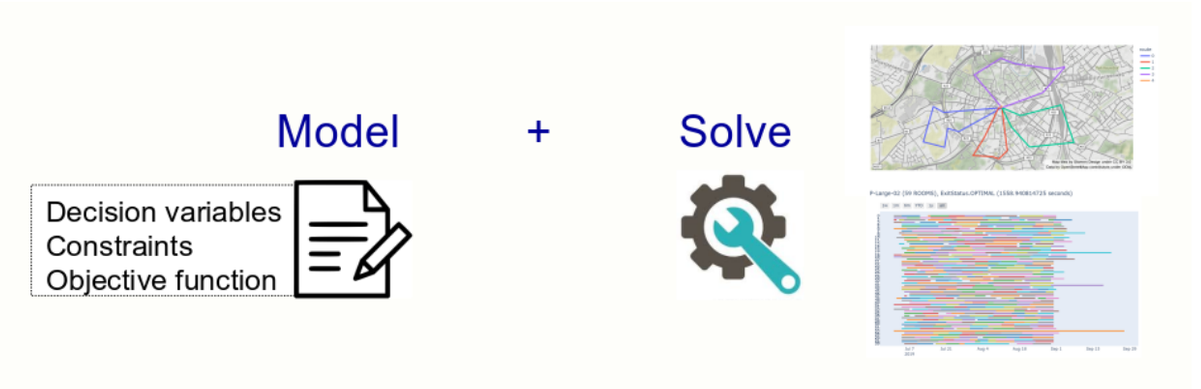
Ignace Bleukx, Dimos Tsouros, Tias Guns

Explainable Constraint Solving - A Hands-On Tutorial

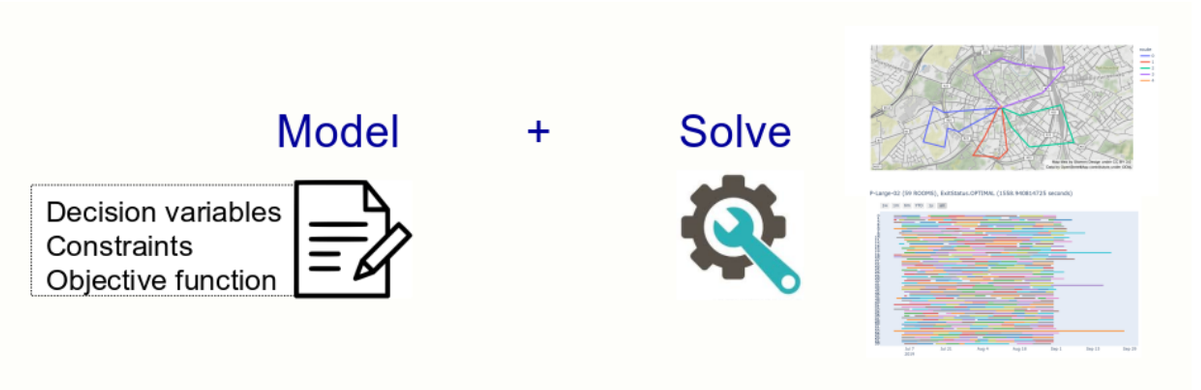
Ignace Bleukx, Dimos Tsouros, Tias Guns

Hands-on: this presentation is an executable Jupyter notebook

Model + Solve

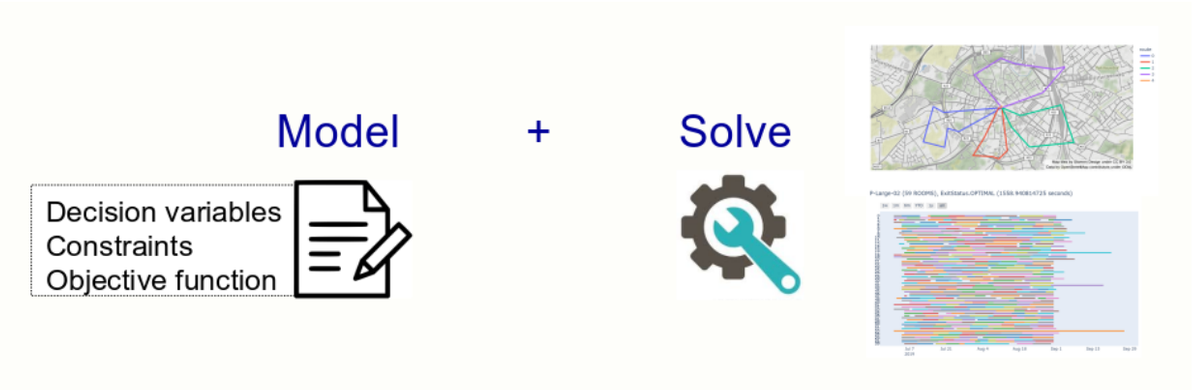

Model
+
Solve
Decision variables
Constraints
Objective function

Model + Solve



- What if the model is UNSAT?
- What if the solution is unexpected?
- What if the user wants to change something?

Model + Solve



- What if the model is UNSAT?
- What if the solution is unexpected?
- What if the user wants to change something?

--> Trustworthy & Explainable AI

Trustworthy & Explainable constraint solving

Human-aware AI:

- Respect human *agency*
- *Support* users in decision making
- Provide explanations and learning opportunities

Trustworthy & Explainable constraint solving

Human-aware AI:

- Respect human *agency*
- *Support* users in decision making
- Provide explanations and learning opportunities

Acknowledges that a 'model' is only an approximation,
that it might result in *undesirable* solutions.

Explainable AI (XAI), brief highlights

D. Gunning, 2015: DARPA XAI challenge

"Every explanation is set within a context that depends on..." -> domain dependent

M. Fox et al, 2017: Explainable Planning

Need for trust, interaction and transparancy.

T. Miller, 2018: Explainable AI: Beware of Inmates Running the Asylum

Insights from the social sciences: *Someone* explains *something* to *someone*

R. Guidotti, 2018: A survey of methods for explaining black box ML models

The vast majority of work/attention

’Explaining’ constraint propagation?

Concept from Lazy Clause Generation / CP-SAT solvers (Stuckey, 2010)

"every time a propagator determines a domain change of a variable,
it records a **clause** that *explains* the domain change."

- Explains one domain change of one propagator
- Is a clause (disjunction of literals)
- Receiver is a SAT solver, not a person

Explainable Constraint Programming (XCP)

In general, "**Why X?**" (with X a solution or UNSAT)

To be defined...

Explainable Constraint Programming (XCP)

In general, "**Why X?**" (with X a solution or UNSAT)

To be defined... 3 patterns:

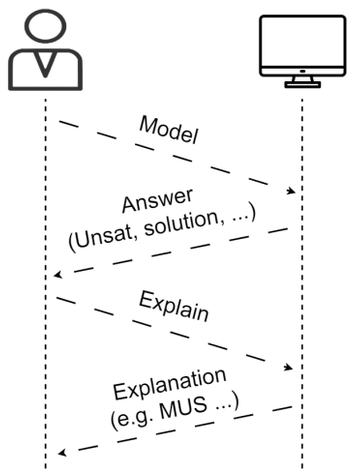
- Causal explanation:
    - How was X derived?
- Contrastive explanation:
    - Why X and not Z?
- Conversational explanation:
    - Iteratively refine explanation & model

Explainable constraint solving

"Explaining UNSAT" vs "Explaining a solution"

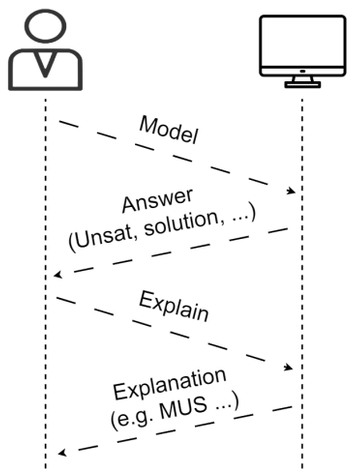
Explainable constraint solving

Causal explanation, mode of interaction:



Explainable constraint solving

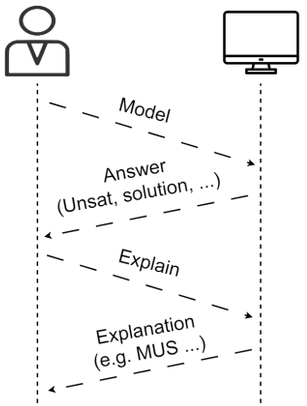
Causal explanation, mode of interaction:



Then what?

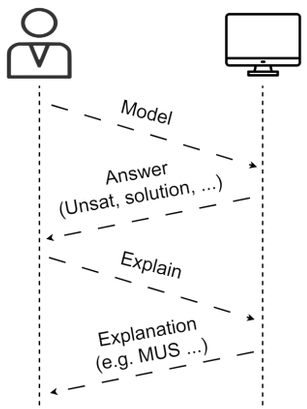
Explainable constraint solving

Causal explanation, mode of interaction:



Then what? Options: 1. User is happy with the answer (e.g. better understands the problem) 2. User changes the answer and uses that (solution interaction; no solver involvement) 3. User changes the model and reiterates (e.g. better understands how to model the problem) 4. User interacts with interactive system (e.g. conversational explanations)

Hands-on Explainable Constraint Programming (XCP)



* The model: Nurse Rostering * The system: CPMpy modeling library * Explain UNSAT: * Causal explanations (MUS, OUS, sequence) * Conversational explanations * Explain a solution: * Causal explanations * Contrastive explanations

The model: Nurse Scheduling

	Day 1			Day 2			...	Day 7			Number of shifts
Shifts	M	E	N	M	E	N	...	M	E	N	
Nurse 1							...				6
Nurse 2							...				7
Nurse 3							...				7
Nurse 4							...				7
……..	...	...	...	...	...	...	...	...	...	...	...
Nurse J							...				6
Number of nurses assigned	15	18	13	13	18	13	...	13	14	13	314

- The assignment of *shifts* and *holidays* to nurses.
- Each nurse has their own restrictions and preferences, as does the hospital.

Nurse Rostering: data

Instances from http://www.schedulingbenchmarks.org/

"benchmark test instances from various sources including industrial collaborators and scientific publications."

Nurse Rostering: data

Instances from http://www.schedulingbenchmarks.org/

"benchmark test instances from various sources including industrial collaborators and scientific publications."

In [2]:
```
#instance = "http://www.schedulingbenchmarks.org/nrp/data/Instance1.txt"
instance = "Benchmarks/Instance1.txt"
data = get_data(instance)

data.staff
```

Out[2]:

	# ID	MaxShifts	MaxTotalMinutes	MinTotalMinutes	MaxConsecutiveShifts	MinConsecutiveShifts	MinConsecut
0	A	D=14	4320	3360	5	2	
1	B	D=14	4320	3360	5	2	
2	C	D=14	4320	3360	5	2	
3	D	D=14	4320	3360	5	2	
4	E	D=14	4320	3360	5	2	
5	F	D=14	4320	3360	5	2	
6	G	D=14	4320	3360	5	2	
7	H	D=14	4320	3360	5	2	

In [3]:

```
print("Nr of days to schedule:", data.horizon)
print("Nr of shift types:", len(data.shifts))

pd.merge(data.days_off, data.staff[["# ID","name"]], left_on="EmployeeID", right_on="# ID", how="le
```

```
Nr of days to schedule: 14
Nr of shift types: 1
```

Out[3]:

	DayIdx	# ID	name
0	0	A	Megan
1	5	B	Katherine
2	8	C	Robert
3	2	D	Jonathan
4	9	E	William
5	5	F	Richard
6	1	G	Kristen
7	7	H	Kevin

Nurse Rostering: constraints

hospital constraints/preferences: * nb of nurses assigned * max nb of shifts * max nb of weekend shifts * min nb of (consecutive) days off * min/max minutes worked * min/max consecutive shifts * shift rotation ### nurse constraints/preferences: * specific days off-duty * specific shift requests (on/off)

	Day 1			Day 2			...	Day 7			Number of shifts
Shifts	M	E	N	M	E	N	...	M	E	N	
Nurse 1							...				6
Nurse 2							...				7
Nurse 3							...				7
Nurse 4							...				7
.....	...	...	...	...	...	...	...	...	...	...	...
Nurse J							...				6
Number of nurses assigned	15	18	13	13	18	13	...	13	14	13	314

The system: http://cpmpy.readthedocs.io

CPMpy is a Constraint Programming and Modeling library in Python,
based on numpy, with direct solver access.

Features used in this tutorial:

- Easy integration with visualisation tools (pandas, matplotlib)
- Object-oriented programming (constraints are Python objects we can create, copy, update)
- Repeatedly solving subsets of constraints (assumption variables)
- Incremental solving (e.g. SAT, MIP/Hitting set)

Nurse Rostering in CPMpy

Variables:

assignment of shift types (0=none) to nurses

Nurse Rostering in CPMpy

Variables:

assignment of shift types (0=none) to nurses

In [4]:
```
nurse_view = cp.intvar(0, len(data.shifts), # lb, ub
                       shape=(len(data.staff), data.horizon),
                       name="nv")
nurse_view
```

Out[4]:
```
NDVarArray([[nv[0,0], nv[0,1], nv[0,2], nv[0,3], nv[0,4], nv[0,5], nv[0,6], nv[0,7],
             nv[0,8], nv[0,9], nv[0,10], nv[0,11], nv[0,12], nv[0,13]],
            [nv[1,0], nv[1,1], nv[1,2], nv[1,3], nv[1,4], nv[1,5], nv[1,6], nv[1,7],
             nv[1,8], nv[1,9], nv[1,10], nv[1,11], nv[1,12], nv[1,13]],
            [nv[2,0], nv[2,1], nv[2,2], nv[2,3], nv[2,4], nv[2,5], nv[2,6], nv[2,7],
             nv[2,8], nv[2,9], nv[2,10], nv[2,11], nv[2,12], nv[2,13]],
            [nv[3,0], nv[3,1], nv[3,2], nv[3,3], nv[3,4], nv[3,5], nv[3,6], nv[3,7],
             nv[3,8], nv[3,9], nv[3,10], nv[3,11], nv[3,12], nv[3,13]],
            [nv[4,0], nv[4,1], nv[4,2], nv[4,3], nv[4,4], nv[4,5], nv[4,6], nv[4,7],
             nv[4,8], nv[4,9], nv[4,10], nv[4,11], nv[4,12], nv[4,13]],
            [nv[5,0], nv[5,1], nv[5,2], nv[5,3], nv[5,4], nv[5,5], nv[5,6], nv[5,7],
             nv[5,8], nv[5,9], nv[5,10], nv[5,11], nv[5,12], nv[5,13]],
            [nv[6,0], nv[6,1], nv[6,2], nv[6,3], nv[6,4], nv[6,5], nv[6,6], nv[6,7],
             nv[6,8], nv[6,9], nv[6,10], nv[6,11], nv[6,12], nv[6,13]],
            [nv[7,0], nv[7,1], nv[7,2], nv[7,3], nv[7,4], nv[7,5], nv[7,6], nv[7,7],
             nv[7,8], nv[7,9], nv[7,10], nv[7,11], nv[7,12], nv[7,13]]], dtype=object)
```

Constraints:

Specific days off-duty

Constraints:

Specific days off-duty

In [5]:
```
for (empl_id, row) in data.days_off.iterrows():
    empl_idx = data.staff.index[data.staff["# ID"] == empl_id][0]
    day_idx = row["DayIdx"]

    con = (nurse_view[empl_idx, day_idx] == 0)

    con.set_description(f"{data.staff.iloc[empl_idx]['name']} should not work on day {day_idx}")
    print(con)
```

```
Megan should not work on day 0
Katherine should not work on day 5
Robert should not work on day 8
Jonathan should not work on day 2
William should not work on day 9
Richard should not work on day 5
Kristen should not work on day 1
Kevin should not work on day 7
```

Max consecutive shifts constraints:

In [6]:
```
for nurse_id, row in data.staff.iterrows():
    max_days = row["MaxConsecutiveShifts"]

    # post on rolling window: in max_days+1 window, at least one day off
    k = max_days+1
    for i in range(data.horizon - max_days):
        con = cp.Count(nurse_view[nurse_id, i:i+k], 0) >= 1

        con.set_description(f"{row['name']} can work at most {max_days} days in a row")
        print(con, "--", con.__repr__())
```

```
Megan can work at most 5 days in a row -- count([nv[0,0],nv[0,1],nv[0,2],nv[0,3],nv[0,4],
nv[0,5]],0) >= 1
Megan can work at most 5 days in a row -- count([nv[0,1],nv[0,2],nv[0,3],nv[0,4],nv[0,5],
nv[0,6]],0) >= 1
Megan can work at most 5 days in a row -- count([nv[0,2],nv[0,3],nv[0,4],nv[0,5],nv[0,6],
nv[0,7]],0) >= 1
Megan can work at most 5 days in a row -- count([nv[0,3],nv[0,4],nv[0,5],nv[0,6],nv[0,7],
nv[0,8]],0) >= 1
Megan can work at most 5 days in a row -- count([nv[0,4],nv[0,5],nv[0,6],nv[0,7],nv[0,8],
nv[0,9]],0) >= 1
Megan can work at most 5 days in a row -- count([nv[0,5],nv[0,6],nv[0,7],nv[0,8],nv[0,9],
nv[0,10]],0) >= 1
Megan can work at most 5 days in a row -- count([nv[0,6],nv[0,7],nv[0,8],nv[0,9],nv[0,1
0],nv[0,11]],0) >= 1
Megan can work at most 5 days in a row -- count([nv[0,7],nv[0,8],nv[0,9],nv[0,10],nv[0,1
1],nv[0,12]],0) >= 1
Megan can work at most 5 days in a row -- count([nv[0,8],nv[0,9],nv[0,10],nv[0,11],nv[0,1
2],nv[0,13]],0) >= 1
Katherine can work at most 5 days in a row -- count([nv[1,0],nv[1,1],nv[1,2],nv[1,3],nv
[1,4],nv[1,5]],0) >= 1
Katherine can work at most 5 days in a row -- count([nv[1,1],nv[1,2],nv[1,3],nv[1,4],nv
[1,5],nv[1,6]],0) >= 1
Katherine can work at most 5 days in a row -- count([nv[1,2],nv[1,3],nv[1,4],nv[1,5],nv
[1,6],nv[1,7]],0) >= 1
Katherine can work at most 5 days in a row -- count([nv[1,3],nv[1,4],nv[1,5],nv[1,6],nv
```

```
[1,7],nv[1,8]],0) >= 1
Katherine can work at most 5 days in a row -- count([nv[1,4],nv[1,5],nv[1,6],nv[1,7],nv
[1,8],nv[1,9]],0) >= 1
Katherine can work at most 5 days in a row -- count([nv[1,5],nv[1,6],nv[1,7],nv[1,8],nv
[1,9],nv[1,10]],0) >= 1
Katherine can work at most 5 days in a row -- count([nv[1,6],nv[1,7],nv[1,8],nv[1,9],nv
[1,10],nv[1,11]],0) >= 1
Katherine can work at most 5 days in a row -- count([nv[1,7],nv[1,8],nv[1,9],nv[1,10],nv
[1,11],nv[1,12]],0) >= 1
Katherine can work at most 5 days in a row -- count([nv[1,8],nv[1,9],nv[1,10],nv[1,11],nv
[1,12],nv[1,13]],0) >= 1
Robert can work at most 5 days in a row -- count([nv[2,0],nv[2,1],nv[2,2],nv[2,3],nv[2,
4],nv[2,5]],0) >= 1
Robert can work at most 5 days in a row -- count([nv[2,1],nv[2,2],nv[2,3],nv[2,4],nv[2,
5],nv[2,6]],0) >= 1
Robert can work at most 5 days in a row -- count([nv[2,2],nv[2,3],nv[2,4],nv[2,5],nv[2,
6],nv[2,7]],0) >= 1
Robert can work at most 5 days in a row -- count([nv[2,3],nv[2,4],nv[2,5],nv[2,6],nv[2,
7],nv[2,8]],0) >= 1
Robert can work at most 5 days in a row -- count([nv[2,4],nv[2,5],nv[2,6],nv[2,7],nv[2,
8],nv[2,9]],0) >= 1
Robert can work at most 5 days in a row -- count([nv[2,5],nv[2,6],nv[2,7],nv[2,8],nv[2,
9],nv[2,10]],0) >= 1
Robert can work at most 5 days in a row -- count([nv[2,6],nv[2,7],nv[2,8],nv[2,9],nv[2,1
0],nv[2,11]],0) >= 1
Robert can work at most 5 days in a row -- count([nv[2,7],nv[2,8],nv[2,9],nv[2,10],nv[2,1
1],nv[2,12]],0) >= 1
Robert can work at most 5 days in a row -- count([nv[2,8],nv[2,9],nv[2,10],nv[2,11],nv[2,
12],nv[2,13]],0) >= 1
Jonathan can work at most 5 days in a row -- count([nv[3,0],nv[3,1],nv[3,2],nv[3,3],nv[3,
4],nv[3,5]],0) >= 1
Jonathan can work at most 5 days in a row -- count([nv[3,1],nv[3,2],nv[3,3],nv[3,4],nv[3,
5],nv[3,6]],0) >= 1
Jonathan can work at most 5 days in a row -- count([nv[3,2],nv[3,3],nv[3,4],nv[3,5],nv[3,
6],nv[3,7]],0) >= 1
Jonathan can work at most 5 days in a row -- count([nv[3,3],nv[3,4],nv[3,5],nv[3,6],nv[3,
7],nv[3,8]],0) >= 1
Jonathan can work at most 5 days in a row -- count([nv[3,4],nv[3,5],nv[3,6],nv[3,7],nv[3,
8],nv[3,9]],0) >= 1
Jonathan can work at most 5 days in a row -- count([nv[3,5],nv[3,6],nv[3,7],nv[3,8],nv[3,
```

```
Jonathan can work at most 5 days in a row -- count([nv[3,6],nv[3,7],nv[3,8],nv[3,9],nv[3,
10],nv[3,11]],0) >= 1
Jonathan can work at most 5 days in a row -- count([nv[3,7],nv[3,8],nv[3,9],nv[3,10],nv
[3,11],nv[3,12]],0) >= 1
Jonathan can work at most 5 days in a row -- count([nv[3,8],nv[3,9],nv[3,10],nv[3,11],nv
[3,12],nv[3,13]],0) >= 1
William can work at most 5 days in a row -- count([nv[4,0],nv[4,1],nv[4,2],nv[4,3],nv[4,
4],nv[4,5]],0) >= 1
William can work at most 5 days in a row -- count([nv[4,1],nv[4,2],nv[4,3],nv[4,4],nv[4,
5],nv[4,6]],0) >= 1
William can work at most 5 days in a row -- count([nv[4,2],nv[4,3],nv[4,4],nv[4,5],nv[4,
6],nv[4,7]],0) >= 1
William can work at most 5 days in a row -- count([nv[4,3],nv[4,4],nv[4,5],nv[4,6],nv[4,
7],nv[4,8]],0) >= 1
William can work at most 5 days in a row -- count([nv[4,4],nv[4,5],nv[4,6],nv[4,7],nv[4,
8],nv[4,9]],0) >= 1
William can work at most 5 days in a row -- count([nv[4,5],nv[4,6],nv[4,7],nv[4,8],nv[4,
9],nv[4,10]],0) >= 1
William can work at most 5 days in a row -- count([nv[4,6],nv[4,7],nv[4,8],nv[4,9],nv[4,1
0],nv[4,11]],0) >= 1
William can work at most 5 days in a row -- count([nv[4,7],nv[4,8],nv[4,9],nv[4,10],nv[4,
11],nv[4,12]],0) >= 1
William can work at most 5 days in a row -- count([nv[4,8],nv[4,9],nv[4,10],nv[4,11],nv
[4,12],nv[4,13]],0) >= 1
Richard can work at most 5 days in a row -- count([nv[5,0],nv[5,1],nv[5,2],nv[5,3],nv[5,
4],nv[5,5]],0) >= 1
Richard can work at most 5 days in a row -- count([nv[5,1],nv[5,2],nv[5,3],nv[5,4],nv[5,
5],nv[5,6]],0) >= 1
Richard can work at most 5 days in a row -- count([nv[5,2],nv[5,3],nv[5,4],nv[5,5],nv[5,
6],nv[5,7]],0) >= 1
Richard can work at most 5 days in a row -- count([nv[5,3],nv[5,4],nv[5,5],nv[5,6],nv[5,
7],nv[5,8]],0) >= 1
Richard can work at most 5 days in a row -- count([nv[5,4],nv[5,5],nv[5,6],nv[5,7],nv[5,
8],nv[5,9]],0) >= 1
Richard can work at most 5 days in a row -- count([nv[5,5],nv[5,6],nv[5,7],nv[5,8],nv[5,
9],nv[5,10]],0) >= 1
Richard can work at most 5 days in a row -- count([nv[5,6],nv[5,7],nv[5,8],nv[5,9],nv[5,1
0],nv[5,11]],0) >= 1
Richard can work at most 5 days in a row -- count([nv[5,7],nv[5,8],nv[5,9],nv[5,10],nv[5,
11],nv[5,12]],0) >= 1
```

```
[5,12],nv[5,13]],0) >= 1
Kristen can work at most 5 days in a row -- count([nv[6,0],nv[6,1],nv[6,2],nv[6,3],nv[6,
4],nv[6,5]],0) >= 1
Kristen can work at most 5 days in a row -- count([nv[6,1],nv[6,2],nv[6,3],nv[6,4],nv[6,
5],nv[6,6]],0) >= 1
Kristen can work at most 5 days in a row -- count([nv[6,2],nv[6,3],nv[6,4],nv[6,5],nv[6,
6],nv[6,7]],0) >= 1
Kristen can work at most 5 days in a row -- count([nv[6,3],nv[6,4],nv[6,5],nv[6,6],nv[6,
7],nv[6,8]],0) >= 1
Kristen can work at most 5 days in a row -- count([nv[6,4],nv[6,5],nv[6,6],nv[6,7],nv[6,
8],nv[6,9]],0) >= 1
Kristen can work at most 5 days in a row -- count([nv[6,5],nv[6,6],nv[6,7],nv[6,8],nv[6,
9],nv[6,10]],0) >= 1
Kristen can work at most 5 days in a row -- count([nv[6,6],nv[6,7],nv[6,8],nv[6,9],nv[6,1
0],nv[6,11]],0) >= 1
Kristen can work at most 5 days in a row -- count([nv[6,7],nv[6,8],nv[6,9],nv[6,10],nv[6,
11],nv[6,12]],0) >= 1
Kristen can work at most 5 days in a row -- count([nv[6,8],nv[6,9],nv[6,10],nv[6,11],nv
[6,12],nv[6,13]],0) >= 1
Kevin can work at most 5 days in a row -- count([nv[7,0],nv[7,1],nv[7,2],nv[7,3],nv[7,4],
nv[7,5]],0) >= 1
Kevin can work at most 5 days in a row -- count([nv[7,1],nv[7,2],nv[7,3],nv[7,4],nv[7,5],
nv[7,6]],0) >= 1
Kevin can work at most 5 days in a row -- count([nv[7,2],nv[7,3],nv[7,4],nv[7,5],nv[7,6],
nv[7,7]],0) >= 1
Kevin can work at most 5 days in a row -- count([nv[7,3],nv[7,4],nv[7,5],nv[7,6],nv[7,7],
nv[7,8]],0) >= 1
Kevin can work at most 5 days in a row -- count([nv[7,4],nv[7,5],nv[7,6],nv[7,7],nv[7,8],
nv[7,9]],0) >= 1
Kevin can work at most 5 days in a row -- count([nv[7,5],nv[7,6],nv[7,7],nv[7,8],nv[7,9],
nv[7,10]],0) >= 1
Kevin can work at most 5 days in a row -- count([nv[7,6],nv[7,7],nv[7,8],nv[7,9],nv[7,1
0],nv[7,11]],0) >= 1
Kevin can work at most 5 days in a row -- count([nv[7,7],nv[7,8],nv[7,9],nv[7,10],nv[7,1
1],nv[7,12]],0) >= 1
Kevin can work at most 5 days in a row -- count([nv[7,8],nv[7,9],nv[7,10],nv[7,11],nv[7,1
```

Object-oriented Nurse Rostering CPMpy model factory

In [7]:
```
factory = NurseSchedulingFactory(data)
model, nurse_view = factory.get_full_model()  # CPMpy model with all constraints
model.solve(solver="ortools")
```

Out[7]:
```
True
```

Object-oriented Nurse Rostering CPMpy model factory

In [7]:
```
factory = NurseSchedulingFactory(data)
model, nurse_view = factory.get_full_model()  # CPMpy model with all constraints
model.solve(solver="ortools")
```

Out[7]:
```
True
```

In [8]:
```
nurse_view.value()
```

Out[8]:
```
array([[0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1],
       [1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0],
       [1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0],
       [1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1],
       [1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1],
       [0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1],
       [1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0]])
```

Nurse rostering in CPMpy: visualisation, with pandas

Nurse rostering in CPMpy: visualisation, with pandas

In [9]:

```
def visualize(sol):
    df = pd.DataFrame(sol,
                      columns=pd.MultiIndex.from_product((weeks, weekdays)),
                      index=factory.data.staff.name)

    mapping = factory.idx_to_name
    df = df.applymap(lambda v : mapping[v] if v is not None and v < len(mapping) else '') # convert

    for shift_type in factory.shift_name_to_idx:
        if shift_type == "F":
            continue
        sums = (df == shift_type).sum() # cover for each shift type
        req = factory.data.cover["Requirement"][factory.data.cover["ShiftID"] == shift_type]
        req.index = sums.index
        df.loc[f'Cover {shift_type}'] = sums.astype(str) + "/" + req.astype(str)
    df["Total shifts"] = (df != "F").sum(axis=1) # shifts done by nurse

    subset = (df.index.tolist()[:-len(factory.data.shifts)], df.columns[:-1])
    style = df.style.set_table_styles([{'selector': '.data','props': [('text-align', 'center')]}])
    style = style.applymap(lambda v : 'border: 1px solid black', subset=subset)
    style = style.applymap(color_shift, subset=subset) # color cells
    return style
```

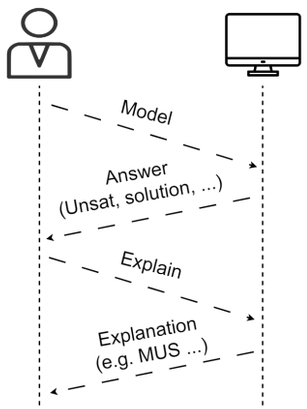
Nurse rostering in CPMpy: visualisation, with pandas

In [10]:
```
visualize(nurse_view.value())
```

Out[10]:

	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Katherine	D	D	D	D	D	F	F	D	D	F	F	D	D	F	9
Robert	D	D	D	F	F	D	D	D	F	F	D	D	F	F	8
Jonathan	D	D	F	F	F	D	D	D	D	D	F	F	F	F	7
William	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Richard	D	D	D	D	D	F	F	F	D	D	F	F	D	D	9
Kristen	F	F	D	D	D	F	F	D	D	D	F	F	D	D	8
Kevin	D	D	F	F	F	F	F	F	D	D	D	D	D	F	7
Cover D	5/5	7/7	6/6	5/4	5/5	2/5	2/5	6/6	7/7	4/4	2/2	5/5	6/6	4/4	14

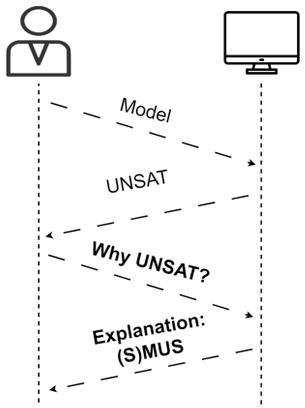
Hands-on Explainable Constraint Programming (XCP)



* The model: Nurse Rostering * The system: CPMpy modeling library * **Explain UNSAT**: * Causal explanations (MUS, OUS, sequence) * Conversational explanations * Explain a solution: * Causal explanations * Contrastive explanations

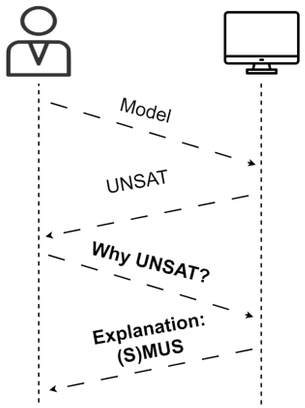
Minimal Unsatisfiable Subsets (MUS)

* Model NSP as decision model * "Nurse constraints" are also hard constraints



Minimal Unsatisfiable Subsets (MUS)

* Model NSP as decision model * "Nurse constraints" are also hard constraints

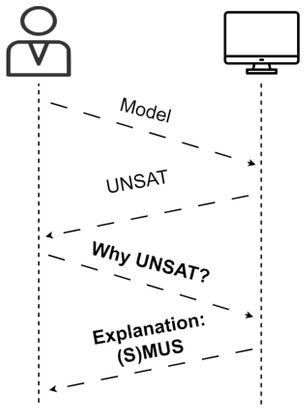


In [11]:
```
# model as decision model
factory = NurseSchedulingFactory(data)
model, nurse_view = factory.get_decision_model()  # CMPpy DECISION Model
model.solve()
```

Out[11]:
```
False
```

Minimal Unsatisfiable Subsets (MUS)

* Model NSP as decision model * "Nurse constraints" are also hard constraints



In [11]:
```
# model as decision model
factory = NurseSchedulingFactory(data)
model, nurse_view = factory.get_decision_model()  # CMPpy DECISION Model
model.solve()
```

Out[11]:
```
False
```

... no solution found

In [12]:

```
constraints = toplevel_list(model.constraints, merge_and=False) # normalization for later
print(f"Model has {len(constraints)} constraints:")
for cons in constraints: print(".", cons)
```

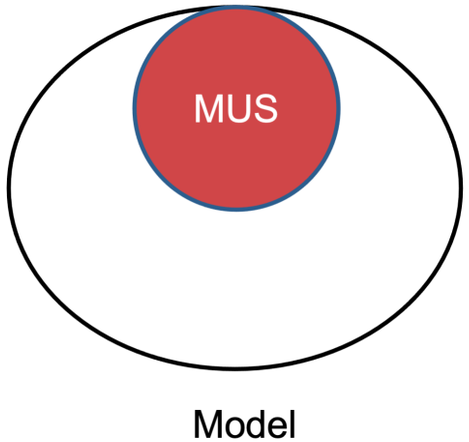
```
Model has 168 constraints:
. Megan cannot work more than 14 shifts of type 1
. Katherine cannot work more than 14 shifts of type 1
. Robert cannot work more than 14 shifts of type 1
. Jonathan cannot work more than 14 shifts of type 1
. William cannot work more than 14 shifts of type 1
. Richard cannot work more than 14 shifts of type 1
. Kristen cannot work more than 14 shifts of type 1
. Kevin cannot work more than 14 shifts of type 1
. Megan cannot work more than 4320min
. Katherine cannot work more than 4320min
. Robert cannot work more than 4320min
. Jonathan cannot work more than 4320min
. William cannot work more than 4320min
. Richard cannot work more than 4320min
. Kristen cannot work more than 4320min
. Kevin cannot work more than 4320min
. Megan cannot work more than 3360min
. Katherine cannot work more than 3360min
. Robert cannot work more than 3360min
. Jonathan cannot work more than 3360min
. William cannot work more than 3360min
. Richard cannot work more than 3360min
. Kristen cannot work more than 3360min
. Kevin cannot work more than 3360min
. Megan can work at most 5 days before having a day off
. Megan can work at most 5 days before having a day off
. Megan can work at most 5 days before having a day off
. Megan can work at most 5 days before having a day off
. Megan can work at most 5 days before having a day off
. Megan can work at most 5 days before having a day off
. Megan can work at most 5 days before having a day off
. Megan can work at most 5 days before having a day off
. Megan can work at most 5 days before having a day off
. Katherine can work at most 5 days before having a day off
```

. Katherine can work at most 5 days before having a day off
. Katherine can work at most 5 days before having a day off
. Katherine can work at most 5 days before having a day off
. Katherine can work at most 5 days before having a day off
. Katherine can work at most 5 days before having a day off
. Katherine can work at most 5 days before having a day off
. Katherine can work at most 5 days before having a day off
. Katherine can work at most 5 days before having a day off
. Robert can work at most 5 days before having a day off
. Robert can work at most 5 days before having a day off
. Robert can work at most 5 days before having a day off
. Robert can work at most 5 days before having a day off
. Robert can work at most 5 days before having a day off
. Robert can work at most 5 days before having a day off
. Robert can work at most 5 days before having a day off
. Robert can work at most 5 days before having a day off
. Jonathan can work at most 5 days before having a day off
. Jonathan can work at most 5 days before having a day off
. Jonathan can work at most 5 days before having a day off
. Jonathan can work at most 5 days before having a day off
. Jonathan can work at most 5 days before having a day off
. Jonathan can work at most 5 days before having a day off
. Jonathan can work at most 5 days before having a day off
. Jonathan can work at most 5 days before having a day off
. Jonathan can work at most 5 days before having a day off
. William can work at most 5 days before having a day off
. William can work at most 5 days before having a day off
. William can work at most 5 days before having a day off
. William can work at most 5 days before having a day off
. William can work at most 5 days before having a day off
. William can work at most 5 days before having a day off
. William can work at most 5 days before having a day off
. William can work at most 5 days before having a day off
. William can work at most 5 days before having a day off
. Richard can work at most 5 days before having a day off
. Richard can work at most 5 days before having a day off
. Richard can work at most 5 days before having a day off
. Richard can work at most 5 days before having a day off
. Richard can work at most 5 days before having a day off

. Richard can work at most 5 days before having a day off
. Richard can work at most 5 days before having a day off
. Richard can work at most 5 days before having a day off
. Kristen can work at most 5 days before having a day off
. Kristen can work at most 5 days before having a day off
. Kristen can work at most 5 days before having a day off
. Kristen can work at most 5 days before having a day off
. Kristen can work at most 5 days before having a day off
. Kristen can work at most 5 days before having a day off
. Kristen can work at most 5 days before having a day off
. Kristen can work at most 5 days before having a day off
. Kristen can work at most 5 days before having a day off
. Kristen can work at most 5 days before having a day off
. Kevin can work at most 5 days before having a day off
. Kevin can work at most 5 days before having a day off
. Kevin can work at most 5 days before having a day off
. Kevin can work at most 5 days before having a day off
. Kevin can work at most 5 days before having a day off
. Kevin can work at most 5 days before having a day off
. Kevin can work at most 5 days before having a day off
. Kevin can work at most 5 days before having a day off
. Kevin can work at most 5 days before having a day off
. Megan should work at least 2 days before having a day off
. Katherine should work at least 2 days before having a day off
. Robert should work at least 2 days before having a day off
. Jonathan should work at least 2 days before having a day off
. William should work at least 2 days before having a day off
. Richard should work at least 2 days before having a day off
. Kristen should work at least 2 days before having a day off
. Kevin should work at least 2 days before having a day off
. Megan should work at most 1 weekends
. Katherine should work at most 1 weekends
. Robert should work at most 1 weekends
. Jonathan should work at most 1 weekends
. William should work at most 1 weekends
. Richard should work at most 1 weekends
. Kristen should work at most 1 weekends
. Kevin should work at most 1 weekends
. Megan should not work on Mon 1
. Katherine should not work on Sat 1
. Robert should not work on Tue 2

. William should not work on Wed 2
. Richard should not work on Sat 1
. Kristen should not work on Tue 1
. Kevin should not work on Mon 2
. Megan should have at least 2 off consecutively
. Katherine should have at least 2 off consecutively
. Robert should have at least 2 off consecutively
. Jonathan should have at least 2 off consecutively
. William should have at least 2 off consecutively
. Richard should have at least 2 off consecutively
. Kristen should have at least 2 off consecutively
. Kevin should have at least 2 off consecutively
. Megan requests to work shift D on Wed 1
. Megan requests to work shift D on Thu 1
. Katherine requests to work shift D on Mon 1
. Katherine requests to work shift D on Tue 1
. Katherine requests to work shift D on Wed 1
. Katherine requests to work shift D on Thu 1
. Katherine requests to work shift D on Fri 1
. Robert requests to work shift D on Mon 1
. Robert requests to work shift D on Tue 1
. Robert requests to work shift D on Wed 1
. Robert requests to work shift D on Thu 1
. Robert requests to work shift D on Fri 1
. Jonathan requests to work shift D on Tue 2
. Jonathan requests to work shift D on Wed 2
. Richard requests to work shift D on Mon 1
. Richard requests to work shift D on Tue 1
. Kevin requests to work shift D on Wed 2
. Kevin requests to work shift D on Thu 2
. Kevin requests to work shift D on Fri 2
. Kevin requests to work shift D on Sat 2
. Kevin requests to work shift D on Sun 2
. Robert requests to not work shift D on Sat 2
. Robert requests to not work shift D on Sun 2
. Richard requests to not work shift D on Tue 2
. Kevin requests to not work shift D on Wed 1
. Kevin requests to not work shift D on Thu 1
. Shift D on Mon 1 must be covered by 5 nurses out of 8
. Shift D on Tue 1 must be covered by 7 nurses out of 8

. Shift D on Thu 1 must be covered by 4 nurses out of 8
. Shift D on Fri 1 must be covered by 5 nurses out of 8
. Shift D on Sat 1 must be covered by 5 nurses out of 8
. Shift D on Sun 1 must be covered by 5 nurses out of 8
. Shift D on Mon 2 must be covered by 6 nurses out of 8
. Shift D on Tue 2 must be covered by 7 nurses out of 8
. Shift D on Wed 2 must be covered by 4 nurses out of 8
. Shift D on Thu 2 must be covered by 2 nurses out of 8
. Shift D on Fri 2 must be covered by 5 nurses out of 8
. Shift D on Sat 2 must be covered by 6 nurses out of 8
. Shift D on Sun 2 must be covered by 4 nurses out of 8



Trim model to minimal set of constraints
... Minimize cognitive burden for user

How to compute a MUS?

Deletion-based MUS algorithm

[Joao Marques-Silva. Minimal Unsatisfiability: Models, Algorithms and Applications. ISMVL 2010. pp. 9-14]

How to compute a MUS?

Deletion-based MUS algorithm

[Joao Marques-Silva. Minimal Unsatisfiability: Models, Algorithms and Applications. ISMVL 2010. pp. 9-14]

In [13]:

```python
def mus_naive(constraints):
    m = cp.Model(constraints)
    assert m.solve() is False, "Model should be UNSAT"

    core = constraints
    i = 0
    while i < len(core):
        subcore = core[:i] + core[i+1:]
        if cp.Model(subcore).solve():
            i += 1 # removing makes it SAT, need to keep
        else:
            core = subcore # can safely delete
    return core
```

How to compute a MUS?

CPMpy implements an incremental version of this, using assumption variables

- `cpmpy.tools.mus`

How to compute a MUS?

CPMpy implements an incremental version of this, using assumption variables

- `cpmpy.tools.mus`

In [14]:
```python
from cpmpy.tools.mus import mus

subset = mus(model.constraints)

print("Length of MUS:", len(subset))
for cons in subset:
    print("-", cons)
```

```
Length of MUS: 11
- Kevin requests to work shift D on Sun 2
- Robert requests to work shift D on Fri 1
- Robert requests to work shift D on Thu 1
- Robert requests to work shift D on Wed 1
- Robert requests to work shift D on Tue 1
- Robert requests to work shift D on Mon 1
- Richard should not work on Sat 1
- Katherine should not work on Sat 1
- Kevin should work at most 1 weekends
- Robert can work at most 5 days before having a day off
- Shift D on Sat 1 must be covered by 5 nurses out of 8
```

In [15]:

```
style = visualize_constraints(subset, nurse_view)
display(style)
```

	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine															14
Robert															14
Jonathan															14
William															14
Richard															14
Kristen															14
Kevin															14
Cover D	0/5	0/7	0/6	0/4	0/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	0/4	14

Many MUS'es may exist...

Liffiton, M.H., & Malik, A. (2013). Enumerating infeasibility: Finding multiple MUSes quickly. In Proceedings of the 10th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR 2013) (pp. 160–175)

Many MUS'es may exist...

Liffiton, M.H., & Malik, A. (2013). Enumerating infeasibility: Finding multiple MUSes quickly. In Proceedings of the 10th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR 2013) (pp. 160–175)

In [16]:
```python
# MARCO MUS/MSS enumeration
from explanations.marco_mcs_mus import do_marco
cnt = 0

t0 = time.time()
for (kind, sset) in do_marco(model, solver="ortools"):
    if kind == "MUS":
        print("M", end="")
        cnt += 1
    else: # MSS
        print(".", end="")

    if time.time() - t0 > 20:  break  # for tutorial: break after 20s
print(f"\nFound {cnt} MUSes")
```

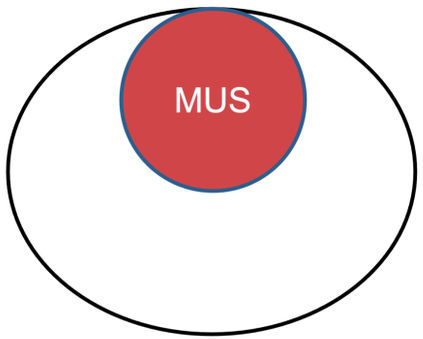
```
MMM....
Found 3 MUSes
```

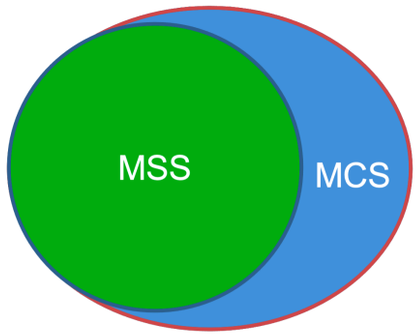
Many MUS'es may exist...

15+ exist for this little problem alone... Which one to show?

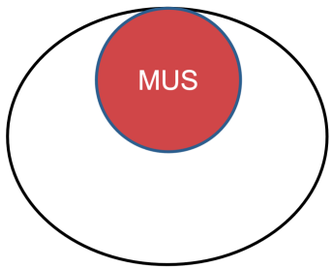
In explanations less is more, so find **smallest one directly!**

Correction subsets and MUS'es

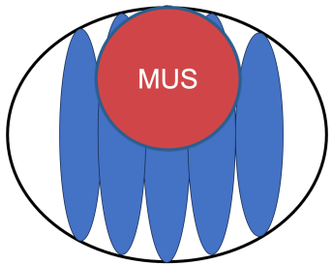




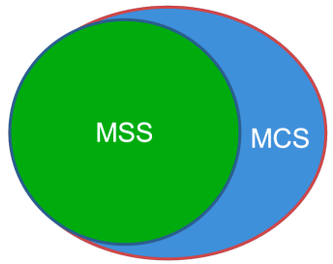
Hitting set duality



Model



Model



Model

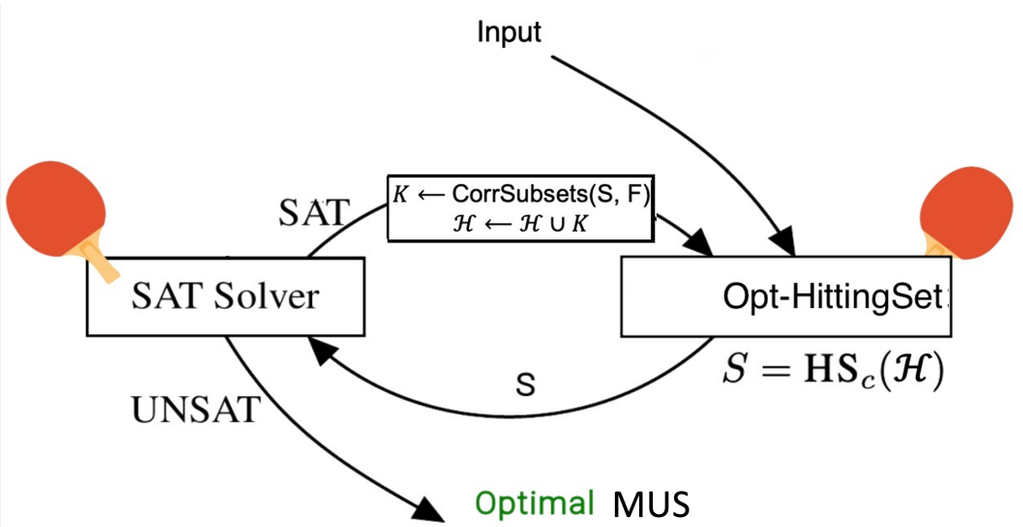
Given all correction subsets, smallest minimal unsatisfiable subset = smallest hitting set to MCS'es

Enumerating all correction subsets is also expensive...

Do it incremental!

How to calculate a Smallest MUS (sMUS)?

1) Initialize sets to hit with MCS 2) Find hitting set and check if SAT 3) If SAT: take complement of HS and add to sets to hit 4) Repeat until HS is UNSAT



In [17]:

```python
from explanations.subset import smus

small_subset = smus(model.constraints, hs_solver="gurobi")

print("Length of sMUS:", len(small_subset))
for cons in small_subset:
    print("-", cons)
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2023-11-09
Length of sMUS: 3
- Robert should not work on Tue 2
- Richard requests to not work shift D on Tue 2
- Shift D on Tue 2 must be covered by 7 nurses out of 8
```

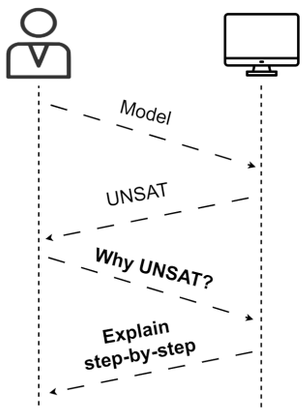
In [18]:

```
style = visualize_constraints(small_subset, nurse_view)
display(style)
```

	Week 1							Week 2						Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine															14
Robert															14
Jonathan															14
William															14
Richard															14
Kristen															14
Kevin															14
Cover D	0/5	0/7	0/6	0/4	0/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	0/4	14

Step-wise explanation

- We were lucky, the SMUS is pretty understandable - What if its not? - Disect MUS into smaller steps - cfr Step-wise Explanations



In [19]:

```
style = visualize_constraints(subset, nurse_view)
display(style)
```

	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine															14
Robert															14
Jonathan															14
William															14
Richard															14
Kristen															14
Kevin															14
Cover D	0/5	0/7	0/6	0/4	0/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	0/4	14

In [19]:
```
style = visualize_constraints(subset, nurse_view)
display(style)
```

	Week 1							Week 2							Total shifts
name	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
Megan															14
Katherine															14
Robert															14
Jonathan															14
William															14
Richard															14
Kristen															14
Kevin															14
Cover D	0/5	0/7	0/6	0/4	0/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	0/4	14

In [20]:
```
from explanations.stepwise import find_sequence
seq = find_sequence(subset)
```

```
Found sequence of length 11
Filtered sequence to length 11
```

In [21]:
```
nurse_view.clear()
visualize_step(seq[0], nurse_view)
```

```
Propagating constraint: Robert requests to work shift D on Fri 1
```

Out[21]:

	Week 1							Week 2						Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine															14
Robert					D										14
Jonathan															14
William															14
Richard															14
Kristen															14
Kevin															14
Cover D	0/5	0/7	0/6	0/4	1/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	0/4	14

In [22]:
```
visualize_step(seq[1], nurse_view)
```

```
Propagating constraint: Robert requests to work shift D on Wed 1
```

Out[22]:

	Week 1							Week 2						Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine															14
Robert			D		D										14
Jonathan															14
William															14
Richard															14
Kristen															14
Kevin															14
Cover D	0/5	0/7	1/6	0/4	1/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	0/4	14

In [23]:
```
visualize_step(seq[2], nurse_view)
```

```
Propagating constraint: Robert requests to work shift D on Tue 1
```

Out[23]:

	Week 1							Week 2						Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine															14
Robert		D	D		D										14
Jonathan															14
William															14
Richard															14
Kristen															14
Kevin															14
Cover D	0/5	1/7	1/6	0/4	1/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	0/4	14

In [24]:
```
visualize_step(seq[3], nurse_view)
```

```
Propagating constraint: Katherine should not work on Sat 1
```

Out[24]:

	Week 1							Week 2						Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine						F									13
Robert		D	D		D										14
Jonathan															14
William															14
Richard															14
Kristen															14
Kevin															14
Cover D	0/5	1/7	1/6	0/4	1/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	0/4	14

In [25]:
```
visualize_step(seq[4], nurse_view)
```

```
Propagating constraint: Kevin requests to work shift D on Sun 2
```

Out[25]:

	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine						F									13
Robert		D	D		D										14
Jonathan															14
William															14
Richard															14
Kristen															14
Kevin														D	14
Cover D	0/5	1/7	1/6	0/4	1/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	1/4	14

In [26]:
```
visualize_step(seq[5], nurse_view)
```

```
Propagating constraint: Robert requests to work shift D on Mon 1
```

Out[26]:

	Week 1							Week 2						Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine						F									13
Robert	D	D	D		D										14
Jonathan															14
William															14
Richard															14
Kristen															14
Kevin														D	14
Cover D	1/5	1/7	1/6	0/4	1/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	1/4	14

In [27]:
```
visualize_step(seq[6], nurse_view)
```

```
Propagating constraint: Kevin should work at most 1 weekends
```

Out[27]:

	Week 1							Week 2						Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine						F									13
Robert	D	D	D		D										14
Jonathan															14
William															14
Richard															14
Kristen															14
Kevin						F								D	13
Cover D	1/5	1/7	1/6	0/4	1/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	1/4	14

In [28]:
```
visualize_step(seq[7], nurse_view)
```

```
Propagating constraint: Richard should not work on Sat 1
```

Out[28]:

	Week 1							Week 2						Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine						F									13
Robert	D	D	D		D										14
Jonathan															14
William															14
Richard						F									13
Kristen															14
Kevin						F								D	13
Cover D	1/5	1/7	1/6	0/4	1/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	1/4	14

In [29]:
```
visualize_step(seq[8], nurse_view)
```

```
Propagating constraint: Robert requests to work shift D on Thu 1
```

Out[29]:

	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine						F									13
Robert	D	D	D	D	D										14
Jonathan															14
William															14
Richard						F									13
Kristen															14
Kevin						F								D	13
Cover D	1/5	1/7	1/6	1/4	1/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	1/4	14

In [30]:
```
visualize_step(seq[9], nurse_view)
```

```
Propagating constraint: Robert can work at most 5 days before having a day off
```

Out[30]:

	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine						F									13
Robert	D	D	D	D	D	F									13
Jonathan															14
William															14
Richard						F									13
Kristen															14
Kevin						F								D	13
Cover D	1/5	1/7	1/6	1/4	1/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	1/4	14

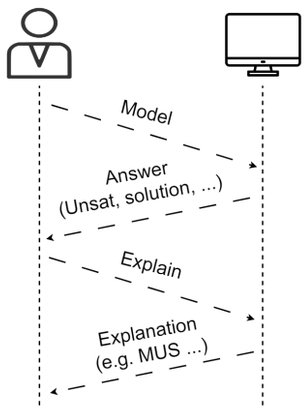
In [31]:
```
visualize_step(seq[10], nurse_view)
```

```
Propagating constraint: Shift D on Sat 1 must be covered by 5 nurses out of 8
```

Out[31]:

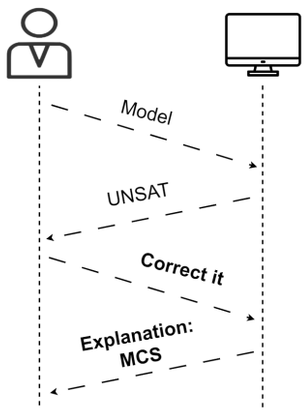
	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine						F									13
Robert	D	D	D	D	D	F									13
Jonathan															14
William															14
Richard						F									13
Kristen															14
Kevin						F								D	13
Cover D	1/5	1/7	1/6	1/4	1/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	1/4	14

Hands-on Explainable Constraint Programming (XCP)



* The model: Nurse Rostering * The system: CPMpy modeling library * Explain UNSAT: * Causal explanations (MUS, OUS, sequence) * **Conversational explanations** * Explain a solution: * Causal explanations * Contrastive explanations

Fixing UNSAT Models



How to modify model in order to find solution?

First idea: find constraints to be removed: correction subset!

How to compute?

Grow-based MCS

How to compute?

Grow-based MCS

In [32]:
```
def mcs_naive(constraints):
    mss = []
    mcs = []
    for cons in constraints:
        if cp.Model(mss + [cons]).solve():
            mss += [cons]
        else:
            mcs += [cons]
    return mcs
```

In [33]:

```
from explanations.subset import mcs

corr_subset = set(mcs(model.constraints))

print("By removing these constraints, the model becomes SAT:")
for cons in corr_subset: print(cons)

visualize_constraints(corr_subset, nurse_view)
```

```
By removing these constraints, the model becomes SAT:
Shift D on Sat 1 must be covered by 5 nurses out of 8
Shift D on Sun 1 must be covered by 5 nurses out of 8
Shift D on Thu 2 must be covered by 2 nurses out of 8
Shift D on Mon 2 must be covered by 6 nurses out of 8
Shift D on Tue 2 must be covered by 7 nurses out of 8
```

Out[33]:

	Week 1							Week 2						Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine															14
Robert															14
Jonathan															14
William															14
Richard															14
Kristen															14
Kevin															14
Cover D	0/5	0/7	0/6	0/4	0/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	0/4	14

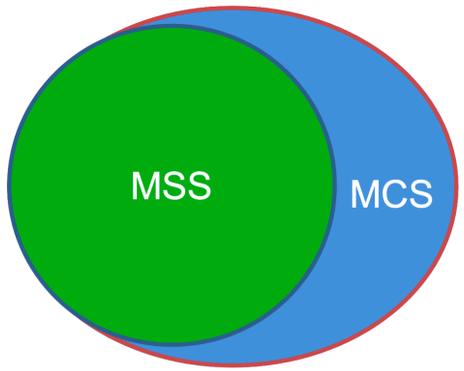
In [34]:

```
mss = []
for cons in toplevel_list(model.constraints, merge_and=False):
    if cons not in corr_subset:
        mss.append(cons)

corrected_model = cp.Model(mss)
assert corrected_model.solve()

visualize(nurse_view.value())
```

Out[34]:

	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan	F	D	D	D	D	F	F	F	F	F	F	D	D	D	7
Katherine	D	D	D	D	D	F	F	F	F	F	F	F	D	D	7
Robert	D	D	D	D	D	F	F	F	F	D	D	F	F	F	7
Jonathan	D	D	F	F	D	D	F	F	D	D	D	F	F	F	7
William	F	D	D	F	F	F	F	D	D	F	F	D	D	D	7
Richard	D	D	D	F	F	F	F	F	F	D	D	D	D	F	7
Kristen	F	F	D	D	D	F	F	D	D	F	F	D	D	F	7
Kevin	D	D	F	F	F	F	F	F	F	D	D	D	D	D	7
Cover D	5/5	7/7	6/6	4/4	5/5	1/5	0/5	2/6	3/7	4/4	4/2	5/5	6/6	4/4	14

Cardinal-minimal MCS



Model

Minimal MCS corresponds to Maximal MSS
Find unweighted MAX-CSP solution

In [35]:

```
from explanations.subset import optimal_mcs

corr_subset = set(optimal_mcs(model.constraints))
print("By removing these constraints, the model becomes SAT:")
for cons in corr_subset: print(cons)

visualize_constraints(corr_subset, nurse_view)
```

```
By removing these constraints, the model becomes SAT:
Shift D on Sat 1 must be covered by 5 nurses out of 8
Richard requests to not work shift D on Tue 2
Shift D on Sun 1 must be covered by 5 nurses out of 8
Robert should not work on Tue 2
```

Out[35]:

	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan															14
Katherine															14
Robert															14
Jonathan															14
William															14
Richard															14
Kristen															14
Kevin															14
Cover D	0/5	0/7	0/6	0/4	0/5	0/5	0/5	0/6	0/7	0/4	0/2	0/5	0/6	0/4	14

In [36]:

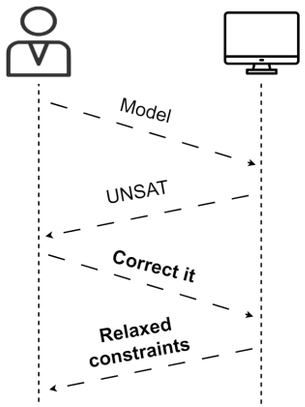
```
mss = []
for cons in toplevel_list(model.constraints, merge_and=False):
    if cons not in corr_subset:
        mss.append(cons)

corrected_model = cp.Model(mss)
assert corrected_model.solve()

visualize(nurse_view.value())
```

Out[36]:

	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan	F	D	D	D	D	F	F	D	D	F	F	D	D	F	8
Katherine	D	D	D	D	D	F	F	D	D	F	F	F	D	D	9
Robert	D	D	D	D	D	F	F	D	D	D	F	F	F	F	8
Jonathan	D	D	F	F	D	D	F	F	D	D	D	D	F	F	8
William	F	D	D	F	F	F	F	D	D	F	F	D	D	D	7
Richard	D	D	D	F	F	F	F	D	D	D	F	F	D	D	8
Kristen	F	F	D	D	D	F	F	D	D	F	F	D	D	F	7
Kevin	D	D	F	F	F	F	F	F	F	D	D	D	D	D	7
Cover D	5/5	7/7	6/6	4/4	5/5	1/5	0/5	6/6	7/7	4/4	2/2	5/5	6/6	4/4	14

Relaxation of constraints

* Removing constraints from model is DRASTIC * What happens if you break a leg on Sunday? ... * Solution: modify constraints in-place



Relaxation of constraints

- Introduce slack for each constraint
- Slack indicates how much a constraint may be violated
    - Less is better of course!
- Minimize *max* of slack values

E.g., *relax* cover requirement for shifts

Senthooran I, Klapperstueck M, Belov G, Czauderna T, Leo K, Wallace M, Wybrow M, Garcia de la Banda M. Human-centred feasibility restoration in practice. Constraints. 2023 Jul 20:1-41.

In [37]:

```
slack_factory = NurseSchedulingFactory(data)
slack_model, slack_nurse_view, slack_view = slack_factory.get_slack_model()  # CMPpy Model

for _, cover in slack_factory.data.cover.iterrows():
    # read the data
    day = cover["# Day"]
    shift = slack_factory.shift_name_to_idx[cover["ShiftID"]]
    requirement = cover["Requirement"]

    nb_nurses = cp.Count(nurse_view[:, day], shift)
    expr = nb_nurses == requirement-slack_view[day]
```

In [38]:

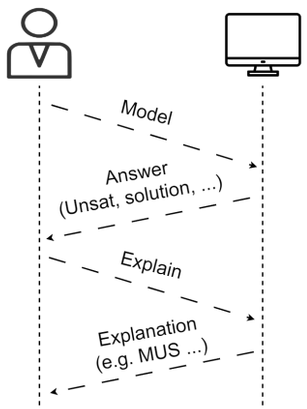
```
slack_factory = NurseSchedulingFactory(data)
slack_model, slack_nurse_view, slack_view = slack_factory.get_slack_model()  # CMPpy Model

slack_model.solve()
print("Slack:", slack_view.value())

visualize(slack_nurse_view.value())
```

```
Slack: [-1  1  0 -2  0  3  3  3  3 -1 -3  2  3  2]
```

Out[38]:

	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan	F	F	D	D	D	D	D	F	F	F	D	D	F	F	7
Katherine	D	D	D	D	D	F	F	D	D	F	F	D	D	F	9
Robert	D	D	D	D	D	F	F	F	F	D	D	F	F	F	7
Jonathan	D	D	F	F	F	F	F	D	D	D	F	F	D	D	7
William	D	D	D	D	F	F	D	D	D	F	F	F	F	F	7
Richard	D	D	D	D	D	F	F	F	F	D	D	F	F	F	7
Kristen	F	F	D	D	D	D	F	F	D	D	D	F	F	F	7
Kevin	D	D	F	F	F	F	F	F	F	D	D	D	D	D	7
Cover D	6/5	6/7	6/6	6/4	5/5	2/5	2/5	3/6	4/7	5/4	5/2	3/5	3/6	2/4	14

Hands-on Explainable Constraint Programming (XCP)



* The model: Nurse Rostering * The system: CPMpy modeling library * Explain UNSAT: * Causal explanations (MUS, OUS, sequence) * Conversational explanations * **Explain a solution**: * Causal explanations * Contrastive explanations

Reformulation as Optimization problem

- Formulate the Nurse Rostering Problem as a Weighted MAX-CSP problem
- Using hard constraints and soft constraints
- Preferences modeled as soft constraints, and minimize penalty of unsatisfied preferences

In [39]:

```
model, nurse_view = factory.get_full_model()
assert model.solve()

opt_sol = nurse_view.value()
display(visualize(opt_sol))
print("Total penalty:", model.objective_value())
print("Time to calculate:", model.status().runtime, "s")
```

	Week 1							Week 2						Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Katherine	D	D	D	D	D	F	F	F	D	D	F	F	D	D	9
Robert	D	D	D	F	F	D	D	D	F	F	D	D	F	F	8
Jonathan	D	D	F	F	F	D	D	D	D	D	F	F	F	F	7
William	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Richard	D	D	D	D	D	F	F	D	D	F	F	D	D	F	9
Kristen	F	F	D	D	D	F	F	D	D	D	F	F	D	D	8
Kevin	D	D	F	F	F	F	F	F	D	D	D	D	D	F	7
Cover D	5/5	7/7	6/6	5/4	5/5	2/5	2/5	6/6	7/7	4/4	2/2	5/5	6/6	4/4	14

```
Total penalty: 607
Time to calculate: 0.36061600000000005 s
```

Multiple solutions

- User not satisfied with optimal solution?
- There could be multiple optimal solutions
- Find (a subset of) them converting it to a decision problem
    - Enforcing the optimal objective value
- Use `solveAll()`

In [40]:

```
opt_model = cp.Model(model.constraints)
opt_model += (model.objective_ == model.objective_value())

opt_model.solveAll(solver="ortools", solution_limit=3,
                   display=lambda: display(visualize(nurse_view.value())))  # callback that visuali
```

	Week 1							Week 2						Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Katherine	D	D	D	D	D	F	F	D	D	F	F	D	D	F	9
Robert	D	D	D	F	F	D	D	D	F	F	D	D	F	F	8
Jonathan	D	D	F	F	F	D	D	D	D	D	F	F	F	F	7
William	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Richard	D	D	D	D	D	F	F	F	D	D	F	F	D	D	9
Kristen	F	F	D	D	D	F	F	D	D	D	F	F	D	D	8
Kevin	D	D	F	F	F	F	F	F	D	D	D	D	D	F	7
Cover D	5/5	7/7	6/6	5/4	5/5	2/5	2/5	6/6	7/7	4/4	2/2	5/5	6/6	4/4	14

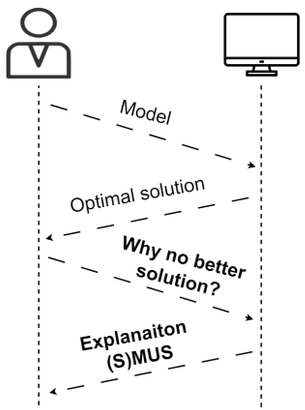
	Week 1							Week 2						Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Katherine	D	D	D	D	D	F	F	F	D	D	F	F	D	D	9
Robert	D	D	D	F	F	D	D	D	F	F	D	D	F	F	8
Jonathan	D	D	F	F	F	D	D	D	D	D	F	F	F	F	7
William	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Richard	D	D	D	D	D	F	F	D	D	F	F	D	D	F	9
Kristen	F	F	D	D	D	F	F	D	D	D	F	F	D	D	8
Kevin	D	D	F	F	F	F	F	F	D	D	D	D	D	F	7
Cover D	5/5	7/7	6/6	5/4	5/5	2/5	2/5	6/6	7/7	4/4	2/2	5/5	6/6	4/4	14

	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Katherine	D	D	D	D	D	F	F	D	D	F	F	F	D	D	9
Robert	D	D	D	F	F	D	D	F	F	D	D	D	F	F	8
Jonathan	D	D	F	F	F	D	D	D	D	D	F	F	F	F	7
William	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Richard	D	D	D	D	D	F	F	D	D	F	F	D	D	F	9
Kristen	F	F	D	D	D	F	F	D	D	D	F	F	D	D	8
Kevin	D	D	F	F	F	F	F	F	D	D	D	D	D	F	7
Cover D	5/5	7/7	6/6	5/4	5/5	2/5	2/5	6/6	7/7	4/4	2/2	5/5	6/6	4/4	14

Out[40]: 3

Causal explanation: Why is there no better solution?

* Reduce to UNSAT problem * Add better-than optimal objective function as constraint * Then use the techniques mentioned to explain why it is now UNSAT

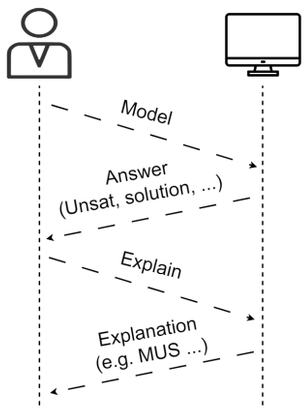


Bleukx, I., Devriendt, J., Gamba, E., Bogaerts B., & Guns T. (2023). Simplifying Step-wise Explanation Sequences. In International Conference on Principles and Practice of Constraint Programming 2023

In [41]:
```
opt_model = cp.Model(model.constraints)
opt_model += (model.objective_ < model.objective_value())

opt_model.solve()
```

Out[41]:
```
False
```

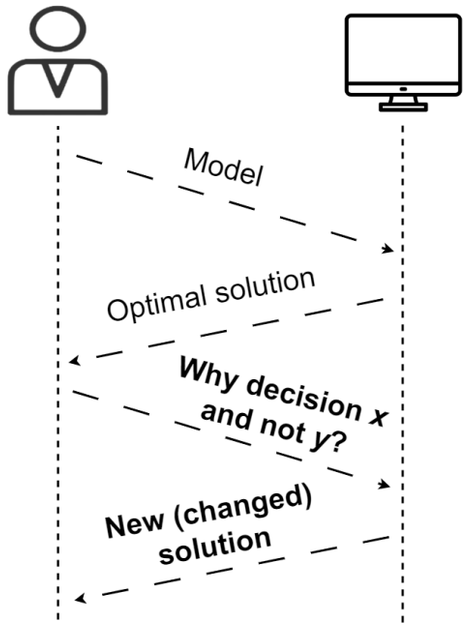
Hands-on Explainable Constraint Programming (XCP)



* The model: Nurse Rostering * The system: CPMpy modeling library * Explain UNSAT: * Causal explanations (MUS, OUS, sequence) * Conversational explanations * Explain a solution: * Causal explanations * **Contrastive explanations**

Changing the solution

* Some assignment might not be what the user wants or expects * Preferences or constraints not given to the model * Add given assignment as constraint and solve again * May result in less optimal objective value * Show new (changed) solution to the user



In [42]:

```
mmodel = model.copy()
mmodel += nurse_view[2,5] == 0 # robert does not want to work on 1st saturday

assert mmodel.solve()
print("Total penalty: ", mmodel.objective_value())
```

```
Total penalty:  608
```

In [42]:
```
mmodel = model.copy()
mmodel += nurse_view[2,5] == 0 # robert does not want to work on 1st saturday

assert mmodel.solve()
print("Total penalty: ", mmodel.objective_value())
```

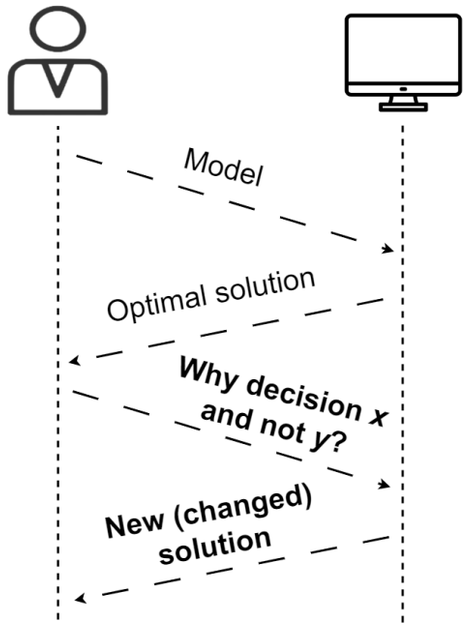
```
Total penalty:  608
```

In [43]:
```
style = highlight_changes(nurse_view, opt_sol)
display(style)
```

	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan	F	D	D	D	D	F	F	D	D	D	F	F	D	D	9
Katherine	D	D	D	D	D	F	F	D	D	F	F	D	D	F	9
Robert	F	D	D	D	D	F	F	F	F	D	D	D	D	D	9
Jonathan	D	D	F	F	F	D	D	D	D	D	F	F	F	F	7
William	D	D	D	F	F	D	D	D	D	F	F	F	F	F	7
Richard	D	D	D	F	F	F	F	D	D	F	F	D	D	D	8
Kristen	F	F	D	D	D	F	F	D	D	F	F	D	D	D	8
Kevin	D	D	F	F	D	D	D	F	F	D	D	D	F	F	8
Cover D	5/5	7/7	6/6	4/4	5/5	3/5	3/5	6/6	6/7	4/4	2/2	5/5	5/6	4/4	14

Slightly changing the solution

* Previous solution is very different from original! * Change only a few parts of it? * Tradeoff between difference and penalty



In [44]:

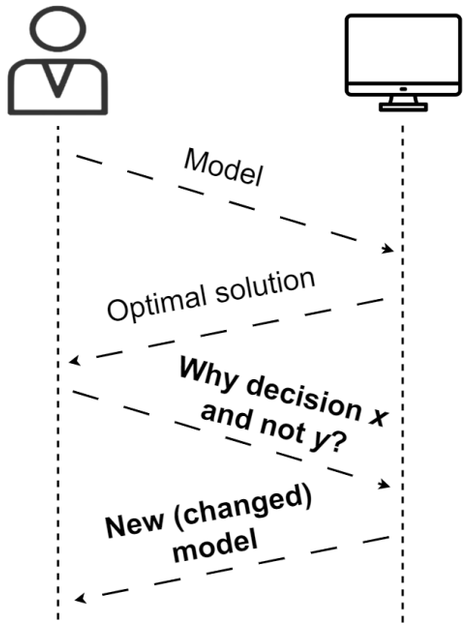
```
mmodel += cp.sum(nurse_view != opt_sol)<= 3 # allow to make 3 changes
assert mmodel.solve()
print("Total penalty: ", mmodel.objective_value())

style = highlight_changes(nurse_view, opt_sol)
display(style)
```

```
Total penalty:  707
```

	Week 1							Week 2						Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Katherine	D	D	D	D	D	F	F	F	D	D	F	F	D	D	9
Robert	D	D	D	D	F	F	D	D	F	F	D	D	F	F	8
Jonathan	D	D	F	F	F	D	D	D	D	D	F	F	F	F	7
William	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Richard	D	D	D	D	D	F	F	D	D	F	F	D	D	F	9
Kristen	F	F	D	D	D	F	F	D	D	D	F	F	D	D	8
Kevin	D	D	F	F	F	F	F	F	D	D	D	D	D	F	7
Cover D	5/5	7/7	6/6	6/4	5/5	1/5	2/5	6/6	7/7	4/4	2/2	5/5	6/6	4/4	14

Counterfactual optimisation model

* "Why not Y" -> "Under what conditions would Y be optimal?"
[Korikov, Anton, and J. Christopher Beck. "Counterfactual explanations via inverse constraint programming." In 27th International Conference on Principles and Practice of Constraint Programming (CP 2021).]
* Given: model with linear objective function $w * c$, and a 'foil' Y (partial assignment)
* Find: new objective function weights w' such that optimal solution satisfies Y *
Explains necessary changes to the **model** rather than the solution!



Counterfactual optimisation model

Find currently optimal solution X:

In [45]:
```
factory = NurseSchedulingFactory(data)
model, nurse_view = factory.get_full_model()

model.solve()
print("Total penalty: ", model.objective_value())
visualize(nurse_view.value())
```

```
Total penalty:  607
```

Out[45]:

	Week 1							Week 2							Total shifts
	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
name															
Megan	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Katherine	D	D	D	D	D	F	F	F	D	D	F	F	D	D	9
Robert	D	D	D	F	F	D	D	D	F	F	D	D	F	F	8
Jonathan	D	D	F	F	F	D	D	D	D	D	F	F	F	F	7
William	F	D	D	D	D	F	F	D	D	F	F	D	D	D	9
Richard	D	D	D	F	F	F	D	D	D	D	F	F	F	F	7
Kristen	F	F	D	D	D	F	F	D	D	F	F	D	D	D	8
Kevin	D	D	F	F	D	D	D	F	F	D	D	D	F	F	8
Cover D	5/5	7/7	6/6	4/4	5/5	3/5	4/5	6/6	6/7	4/4	2/2	5/5	4/6	4/4	14

Counterfactual optimisation model

Robert is unhappy!

In [46]:
```
nurse = "Robert"

for (w,pref) in zip(*model.objective_.args):
    if nurse in str(pref):
        print(f"{pref.value()} \t w:{w} \t{pref}")
```

```
False    w:1    Robert requests to work shift D on Mon 1
False    w:1    Robert requests to work shift D on Tue 1
False    w:1    Robert requests to work shift D on Wed 1
True     w:1    Robert requests to work shift D on Thu 1
True     w:1    Robert requests to work shift D on Fri 1
False    w:1    Robert requests to not work shift D on Sat 2
False    w:1    Robert requests to not work shift D on Sun 2
```

Counterfactual optimisation model

Robert is unhappy!

In [46]:
```
nurse = "Robert"

for (w,pref) in zip(*model.objective_.args):
    if nurse in str(pref):
        print(f"{pref.value()} \t w:{w} \t{pref}")
```

```
False    w:1    Robert requests to work shift D on Mon 1
False    w:1    Robert requests to work shift D on Tue 1
False    w:1    Robert requests to work shift D on Wed 1
True     w:1    Robert requests to work shift D on Thu 1
True     w:1    Robert requests to work shift D on Fri 1
False    w:1    Robert requests to not work shift D on Sat 2
False    w:1    Robert requests to not work shift D on Sun 2
```

In [47]:
```
desc = "Robert requests to work shift D on Fri 1"
weight,d_on_fri1 = next((w,pref) for w,pref in zip(*model.objective_.args) if str(pref) == desc)
print(f"{d_on_fri1.value()} \t w:{w} \t{d_on_fri1}")
```

```
True     w:1    Robert requests to work shift D on Fri 1
```

Counterfactual optimisation model

Robert does not want to work on Fri 1!

How should he minimally change *his* preferences for that?

Counterfactual optimisation model

Robert does not want to work on Fri 1!

How should he minimally change *his* preferences for that?

In [48]:
```
foil = {d_on_fri1 : False}  # don't want to work on Fri 1!
print("Foil:", foil)
print("\n")

other_prefs = [(w,pref) for w,pref in zip(*model.objective_.args) if nurse in str(pref) and str(pre
print(f"{nurse}'s other preferences:")
for w,pref in other_prefs:
    print("- Weight",w,":",pref)
```

```
Foil: {roster[2,4] != 1: False}


Robert's other preferences:
- Weight 1 : Robert requests to work shift D on Mon 1
- Weight 1 : Robert requests to work shift D on Tue 1
- Weight 1 : Robert requests to work shift D on Wed 1
- Weight 1 : Robert requests to work shift D on Thu 1
- Weight 1 : Robert requests to not work shift D on Sat 2
- Weight 1 : Robert requests to not work shift D on Sun 2
```

Counterfactual optimisation model

Algorithmically, it is a beautiful inverse optimisation problem with a multi-solver main/subproblem algorithm

Counterfactual optimisation model

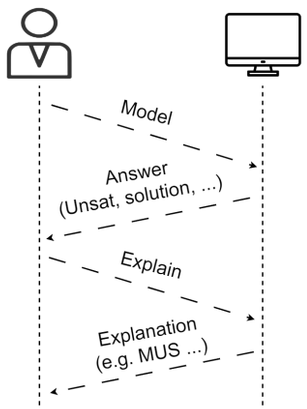
Algorithmically, it is a beautiful inverse optimisation problem with a multi-solver main/subproblem algorithm

In [49]:

```
from explanations.counterfactual import inverse_optimize

new_obj = inverse_optimize(model=model, minimize=True,
                           user_sol = foil,
                           allowed_to_change = set(p[1] for p in other_prefs))
print(f"Done! Found solution with total penalty {new_obj.value()}, was {model.objective_value()}\n"

# Let's look at the preferences he should enter, to avoid Fri 1!
print(f"{nurse}'s new preferences:")
for w,pref in zip(*new_obj.args):
    if nurse in str(pref) and str(pref) != desc and w != 1:  # previous weights were 1
        print("Weight",w,":",pref)
```

```
Done! Found solution with total penalty 607, was 608

Robert's new preferences:
Weight 0 : Robert requests to not work shift D on Sat 2
```

Hands-on Explainable Constraint Programming (XCP)



* The model: Nurse Rostering * The system: CPMpy modeling library * Explain UNSAT: * Causal explanations (MUS, OUS, sequence) * Conversational explanations * Explain a solution: * Causal explanations * Contrastive explanations

CHAT-Opt:
Conversational **H**uman-**A**ware **T**echnology for **Opt**imisation

Towards **co-creation** of constraint optimisation solutions

- Solver that learns from user and environment
- Towards conversational: explanations and stateful interaction

https://people.cs.kuleuven.be/~tias.guns/chat-opt.html

Visitors welcome!

Explainable Constraint Programming (XCP)

In general, "**Why X?**" (with X a solution or UNSAT)

3 patterns:

- Causal explanation:
    - How was X derived?
- Contrastive explanation:
    - Why X and not Z?
- Conversational explanation:
    - Iteratively refine explanation & model

Connections to wider XAI

- Explanations in planning, e.g. MUGS (Eiflet et al), Model Reconciliation (Chakraborti et al), ...
- Explanations for KR/justifications (Swartout et al), ASP (Fandinno et al), in OWL (Kalyanpur et al), ...
- Formal explanations of ML models (e.g. impl. hitting-set based, Ignatiev et al)

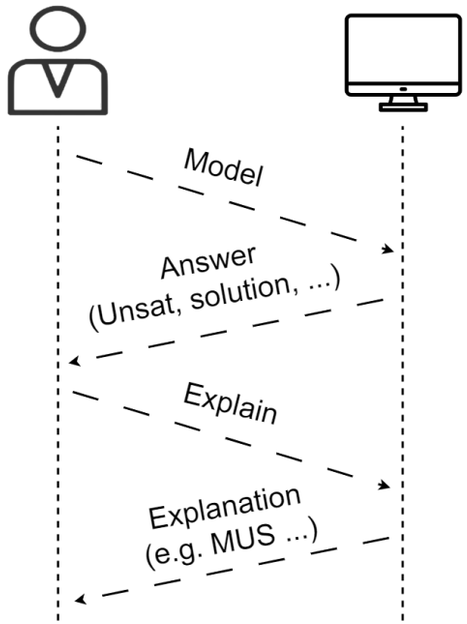
Explainable Constraint Programming (XCP)

Recurring challenges:

- Definition of explanation: question and answer format
- Computational efficiency
- Explanation selection: which explanation to show
- User Interaction? (visualisation, conversational, statefull, ...)
- Explanation evaluation: computational, formal, user survey, user study, ...

Conclusion

* Explanation of UNSAT/SAT/Opt * Causal explanations relate back to finding a (cost-optimal) MUS * Need for programmable multi-solver tooling: CPMpy * Many open challenges and new problems to be discovered! * Less developed: contrastive explanations, conversational explanations * We need incremental CP-solvers!



References mentioned (many more exist!!!)

MUS

- Liffiton, M. H., & Sakallah, K. A. (2008). Algorithms for computing minimal unsatisfiable subsets of constraints. Journal of Automated Reasoning, 40, 1-33.
- Ignatiev, A., Previti, A., Liffiton, M., & Marques-Silva, J. (2015, August). Smallest MUS extraction with minimal hitting set dualization. In International Conference on Principles and Practice of Constraint Programming (pp. 173-182). Cham: Springer International Publishing.
- Joao Marques-Silva. Minimal Unsatisfiability: Models, Algorithms and Applications. ISMVL 2010. pp. 9-14

Feasibility restoration

- Senthooran, I., Klapperstueck, M., Belov, G., Czauderna, T., Leo, K., Wallace, M., ... & De La Banda, M. G. (2021). Human-centred feasibility restoration. In 27th International Conference on Principles and Practice of Constraint Programming (CP 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Explaining optimization problems

- Korikov, A., & Beck, J. C. (2021). Counterfactual explanations via inverse constraint programming. In 27th International Conference on Principles and Practice of Constraint Programming (CP 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Explanation in planning, ASP, KR

- Eifler, Rebecca, Michael Cashmore, Jörg Hoffmann, Daniele Magazzeni, and Marcel Steinmetz. "A new approach to plan-space explanation: Analyzing plan-property dependencies in oversubscription planning." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 06, pp. 9818-9826. 2020.
- Chakraborti, Tathagata, Sarath Sreedharan, Yu Zhang, and Subbarao Kambhampati. "Plan explanations as model reconciliation: moving beyond explanation as soliloquy." In Proceedings of the 26th International Joint Conference on Artificial Intelligence, pp. 156-163. 2017.
- Fandinno, Jorge, and Claudia Schulz. "Answering the “why” in answer set programming–A survey of explanation approaches." Theory and Practice of Logic Programming 19, no. 2 (2019): 114-203.
- Swartout, William, Cecile Paris, and Johanna Moore. "Explanations in knowledge systems: Design for explainable expert systems." IEEE Expert 6, no. 3 (1991): 58-64.
- Kalyanpur, Aditya, Bijan Parsia, Evren Sirin, and Bernardo Cuenca-Grau. "Repairing unsatisfiable concepts in OWL ontologies." In The Semantic Web: Research and Applications: 3rd European Semantic Web Conference, ESWC 2006 Budva, Montenegro, June 11-14, 2006 Proceedings 3, pp. 170-184. Springer Berlin Heidelberg, 2006.

Formal explantions in ML

- Ignatiev, Alexey, Nina Narodytska, and Joao Marques-Silva. "Abduction-based explanations for machine learning models." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, no. 01, pp. 1511-1519. 2019.

</small>