1 Modelling The Second International Nurse Rostering Competition (2014)

In short, the scheduling problem presented in [**?**] consists of assigning nurses to shifts in multiple consecutive planning periods. The competition was held in 2015, on the website the benchmarks and a solution validator are still available.

This is an example for a scenario (global part of the problem description) from a test instance:

```
SCENARIO = n005w4

WEEKS = 4

SKILLS = 2
HeadNurse
Nurse

SHIFT_TYPES = 3
Early (2,5)
Late (2,3)
Night (4,5)

FORBIDDEN_SHIFT_TYPES_SUCCESSIONS
Early 0
Late 1 Early
Night 2 Early Late

CONTRACTS = 2
FullTime (15,22) (3,5) (2,3) 2 1
PartTime (7,11) (3,5) (3,5) 2 1

NURSES = 5
Patrick FullTime 2 HeadNurse Nurse
Andrea FullTime 2 HeadNurse Nurse
Stefaan PartTime 2 HeadNurse Nurse
Sara PartTime 1 Nurse
Nguyen FullTime 1 Nurse
```

Only one time unit is necessary: a counter of days since the start of the planning horizon. Create groups: HeadNurse, Nurse, Early, Late, Night, Saturday1, Sunday1, Saturday2, Sunday2, Saturday3, Sunday3, Saturday4, Sunday4 Nurse can be used as an alias for Agent, Shift for TimedTask.

```
using Nurse = Agent<std::string, std::string, std::string>;
using Shift = TimedTask<std::string, std::string, std::string, int>;
```

Create each nurse, add the appropriate groups for roles. When reading from a file, this can be done in a loop; here only examplary for a single nurse:

```
Nurse patrick("Patrick");
patrick.addGroup("HeadNurse");
patrick.addGroup("Nurse");
```

There is only a single type of assignment for this problem. Since the problem is to assign nurses *to* shifts, each shift will be fixed in one assignment.

```
class Shift : public Assignment {

        Expression generate() override;
        Agent nurse;
        const TimedTask shift;
        std::vector<Rule> requirements;

};
```

Create a TimedTask for every role required for each shift. Add the appropriate groups for role requirements, shift type and day of the week. Add these requirements to all Shifts s:

```
s.Require(shift.inGroup(HeadNurse) IMPLIES nurse.inGroup(HeadNurse));
s.Require(shift.inGroup(Nurse) IMPLIES nurse.inGroup(Nurse));
```

Ideas to model the constraints as described in [**?**] as rules:

H1. Single assignment per day A1.Simultaneous(A2) IMPLIES A1.Nurse != A2.Nurse, or

A1.Shift.StartTime == A2.Shift.StartTime IMPLIES A1.Nurse != A2.Nurse

H2. Under-staffing This is already enforced by making required tasks non-optional.

H3. Illegal shift type successions For every illegal combination (prec, succ):

InGroup(A1.Shift, prec) AND InGroup(A2.Shift, succ) AND ImmediateSuccessor(A1.Shift, A2.Shift) IMPLIES A1.Nurse != A2.Nurse

H4. Missing required skill Enforced by making required role tasks non-optional.

S1. Insufficient staffing for optimal coverage Enforced by adding the appropriate weight (30) as penalty to optional shifts.

S2. Consecutive assignments Define

SameType(A1, A2) := (A1.shift.inGroup(Early) AND A2.shift.inGroup(Early)) OR (A1.shift.inGroup(Late) AND A2.shift.inGroup(Late)) OR (A1.shift.inGroup(Night) AND A2.shift.inGroup(Night))

MinConsecutive(A1.nurse == A2.nurse AND SameType(A1, A2)) ¿= MIN

S3. Consecutive days off MaxConsecutive(A1.nurse == A2.nurse AND SameType(A1, A2)) ¡= MAX

S4. Preferences

S5. Complete week-end

S6. Total assignments (only evaluated at the end of the planning period)

S7. Total working week-ends (only evaluated at the end of the planning period)