A complete Assignment Algorithm and Its Application in Constraint Declarative Languages

N. Brauner, R. Echahed, G. Finke, H. Gregor, F. Prost

To cite this version:
N. Brauner, R. Echahed, G. Finke, H. Gregor, F. Prost. A complete Assignment Algorithm and Its Application in Constraint Declarative Languages. Cahier du laboratoire Leibniz N°111. 2004. <hal-00082787>

HAL Id: hal-00082787
https://hal.archives-ouvertes.fr/hal-00082787
Submitted on 28 Jun 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A complete Assignment Algorithm and Its Application in Constraint Declarative Languages

N. Brauner, R. Echahed, G. Finke, H. Gregor, F. Prost
A complete Assignment Algorithm and Its Application in Constraint Declarative Languages

N. Brauner, R. Echahed, G. Finke, H. Gregor, F. Prost

Institut d’Informatique et de Mathématiques Appliquées de Grenoble, Laboratoire Leibniz, 46, av Félix Viallet, 38000 Grenoble, France
email: name@imag.fr

Abstract. An important property of answer generation strategies for functional logic programming (FLP) languages is the complete exploration of the solution space. Integrating constraints into FLP proves to be useful in many cases, as the resulting constraint functional logic programming (CFLP) offers more facilities and more efficient operational semantics. CFLP can be achieved using a conditional rewrite system with a narrowing-based operational semantics. A common idea to improve the efficiency of such operational semantics is to use specific algorithms from operations research as constraint solvers. If the algorithm does not return a complete set of solutions, the property of completeness might be lost. We present a real world timetabling problem illustrating this approach. We propose an algorithm, obtained as an integration of three known optimization algorithms for the linear assignment problem (LAP), enumerating solutions to the LAP in order of increasing weight, such that the answer generation is complete again. We show, how the narrowing process can be tailored to use this algorithm and provide an efficient way to solve the timetable generation problem.

Keywords: Functional-logic programming, Constraints, Narrowing, Timetable generation

1 Introduction

In software development, it is often practical to use different programming paradigms for modeling different parts of the problem to solve. A system integrating various programming paradigms allows the programmer to express each part of the problem with the best suiting concepts. The formalization of the whole program also becomes easier and the resulting software design is often considerably less complex.

By using a conditional rewrite system with narrowing as operational semantics, it is possible to combine constraint logic programming [14] and functional logic programming [24, 11] in a single constraint functional logic programming framework [20, 15, 12]. A program in such a framework is composed of conditional rewrite rules of the form

\[ L \rightarrow R \mid Q_i(s_1, \ldots, s_{m_i}), i \in \{0..n\} \]
where \( Q_i(s_1, \ldots, s_m) \) denotes a list, possibly empty, of conditions. This rule can be applied as a classical rewrite rule only if the conditions \( Q_i \) hold. Typically, \( Q \) is an equation or an \( m \)-ary constraint (predicate). With appropriate narrowing strategies, it is possible to obtain a complete operational semantics for the interpretation of such conditional rewrite rules. Nevertheless, for some specific constraints the evaluation time of narrowing, which is essentially an exhaustive search among the possibilities, can be prohibitive. In order to cope with this problem under those specific conditions, it is possible to use operations research algorithms, which are much more efficient. In order to maintain the completeness of the calculus when specialized constraint solvers are integrated, each of these solvers needs to return a complete set of solutions. In this paper we focus on a particular case involving a solver for the linear assignment problem.

The linear assignment problem is efficiently solved by the Hungarian method \cite{17}, which is described more in detail in section 2.2. This algorithm returns exactly one optimal solution to the stated problem. It thereby leads to an incomplete operational semantics. A first aim of this paper is to propose an algorithm for the linear assignment problem that enumerates all solutions to the assignment problem in order of decreasing quality (increasing cost). This is done by the combination of three known algorithms. The completeness of classical operational semantics of logic or functional logic programming languages with constraints is then recovered. A second aim is to illustrate the proposed algorithm by a case study: the timetable generation for medical staff.

We assume the reader is familiar with narrowing-based languages (see e.g. \cite{11}). Thus we rather focus, in section 2, on the three algorithms for the assignment problem which we propose to combine. Then we show in section 3 how a combination of these algorithms can be done in order to enumerate all possible (including non-optimal) assignments. Section 4 presents a practical case study, namely timetable generation of medical staff in a Parisian hospital. We illustrate how declarative programming can be efficiently used in such a case thanks to this new complete algorithm for LAP. Finally, we conclude and discuss further works in section 5.

2 Linear Assignment Problem: Definition and Solutions

In this section we precise the linear assignment problem and present three algorithms from literature about this problem. The first one, the Hungarian method, gives exactly one answer, the second one enumerates all optimal assignments, while the final one considers suboptimal assignments.

2.1 Definition

A graph \( G = (V, E) \) with vertex set \( V \) and edge set \( E \) is said to be bipartite, if \( V \) can be partitioned into two sets \( S \) and \( T \) such that in each of these sets no two vertices are connected by an edge. A subset \( M \) of \( E \) is called a matching, if no vertex in \( V \) is incident to more than one edge in \( M \). A perfect matching is a
matching $M$ such that every vertex of the graph is incident to exactly one edge in $M$. Such a matching can only exist, if $|S| = |T|$. A bipartite graph is said to be complete, if $(s_i, t_j) \in E$ for every pair of $s_i \in S$ and $t_j \in T$.

Given a complete weighted bipartite graph $G = (S \cup T, E)$ with integer weights $w_{ij}$ (in this paper we consider positive weights) associated to every edge $(s_i, t_j) \in E$, the problem of finding a minimum weight perfect matching $M$ is known as the linear assignment problem (LAP). This problem can be expressed in the form of a linear program by introducing variables $x_{ij}$ with

$$x_{ij} = \begin{cases} 1 & \text{if } (s_i, t_j) \in M, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

The complete linear programming formulation of an LAP of size $n$ herewith becomes

Constraints:
$$\sum_{i=1}^{n} x_{ij} = 1 \quad \sum_{j=1}^{n} x_{ij} = 1$$

Objective: Minimize $\sum_{i,j=1}^{n} w_{ij} x_{ij}$

The definition of an LAP can easily be extended to cover incomplete bipartite graphs and graphs in which $|S| \neq |T|$. In the first case, $w_{ij}$ is set to $\infty$ whenever $(s_i, t_j) \notin E$, and a solution to an LAP only exists if $\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_{ij} < \infty$. In a graph with $|S| > |T|$, a set $T'$ of $|S| - |T|$ additional vertices can be introduced with weights $w_{ij} = 0$ (or the minimum of weights if negative weights are allowed) for all $s_i \in S$. Figure 1 shows the graph for an LAP with $|S| = 4$, $|T| = 3$, and one additional vertex $t_4$.

![Fig. 1. Graph for an LAP of size 4.](image)

The so-called Hungarian method [17,13] (or Kuhn-Munkres algorithm) solves the assignment problem in polynomial time. It is based on work of the Hungarian
mathematicians König and Egerváry and is an adapted version of the primal-dual algorithm for network flows. For an assignment between two vertex sets of cardinality \( n \) the time complexity of the algorithm is \( O(n^3) \). It operates on the \( n \times n \) weight matrix \( W = w_{ij} \), and computes a permutation \( \sigma = (j_1, \ldots, j_n) \) of columns 1, \ldots, \( n \) such that the sum \( \sum_i w_{ij} \), is minimized. \( \sum_i w_{ij} \), then equals the total weight of the assignment. In operations research literature the weight matrix \( W \) is usually referred to as cost matrix. An \( O(n^2 \log n) \) algorithm was proposed in [8]. It is currently the best complexity for solving LAP.

In this paper, we focus on the Hungarian method, which allows for an easier implementation and has the property that a solution is constructed incrementally. Given a solution for weight matrix \( W = w_{ij} \), this incrementality permits finding a solution to a problem, in which only one assignment is excluded from the solution (i.e. one value \( w_{ij} \) in the weight matrix is changed), very quickly.

### 2.2 Hungarian Method

We informally present the Hungarian Method using the example presented in figure 1. We refer to [17] for a more formal presentation.

The weight matrix \( W \) for our example is

\[
\begin{array}{cccc}
  & t_1 & t_2 & t_3 & t_4 \\
 s_1 & 4 & 9 & 1 & 0 \\
 s_2 & 2 & 5 & 8 & 0 \\
 s_3 & 13 & 7 & 1 & 0 \\
 s_4 & 4 & 12 & 9 & 0 \\
\end{array}
\]

The permutation \( \sigma = (1, 2, 3, 4) \) constitutes an optimal solution with \( 4 + 5 + 1 + 0 = 10 \) being the value of the objective function.

The rows and columns of the weight matrix are called lines. An optimal assignment does not change, if a constant is added or subtracted from a line. Indeed, the assignment problem consists of exactly one entry in every row and every column of the matrix.

Furthermore, if there exists a solution in zeros in a matrix with only non-negative entries, this solution will obviously be optimal with the value of the objective function being zero.

The Hungarian method makes use of these two properties. It applies transformations to the weight matrix that leave the optimal permutation unchanged, until there exists a solution in zeros. It then constructs such a solution out of the matrix obtained.

Any matrix with integer numbers can be transformed into a matrix with nonnegative entries and at least one zero in every row and column by subtracting the minimum of each row and then the minimum of each column. This procedure is the first step of the Hungarian method, but it does not necessarily yield a
solution in zeros:

\[
\begin{pmatrix}
4 & 9 & 1 & 0 \\
2 & 5 & 8 & 0 \\
13 & 7 & 1 & 0 \\
4 & 12 & 9 & 0
\end{pmatrix}
\rightarrow
\begin{pmatrix}
\pm 0 & 2 & 4 & 0 & 0 \\
\pm 0 & 0 & 7 & 0 \\
\pm 0 & 11 & 2 & 0 & 0 \\
\pm 0 & 2 & 7 & 8 & 0
\end{pmatrix}
\]

A subset of the zeros in the matrix \( W \) is said to be an \textit{independent set}, if there are no two zeros in the set, which are lying in the same row or in the same column. That is, a set \( \{(i_1, j_1), \ldots, (i_m, j_m)\} \) of zeros represented by their row and column numbers in matrix \( W \) is independent, if for all pairs of zeros in the set, we have \( i_k \neq i_l \) and \( j_k \neq j_l \) whenever \( k \neq l \). We also refer to the elements of such an independent set as \textit{independent zeros}. We note that a solution in zeros – if it exists – forms an independent set, as it only has one entry in every row and every column. An independent set is called \textit{complete}, if every zero in \( W \) lies in the line of an independent zero.

A set of lines \( C \) is called a \textit{cover} of matrix \( W \), if every zero in \( W \) lies in one of the lines of \( C \). The lines of the cover are referred to as \textit{covering lines}.

König and Egerváry established a connection between these two notions by showing that the maximum number of independent zeros is always equal to the minimum number of covering lines. Their theorem is used in the Hungarian method to construct a solution from the initial weight matrix. The complete algorithm consists of the following steps:

1. For the initial weight matrix subtract the minimum from each row and then subtract the minimum from each column.

2. Construct a maximum independent set and a minimal cover of same cardinality \( k \). Exit if \( k = n \).

3. Let \( h \) be the minimum of all noncovered elements. Add \( h/2 \) to all elements in each covering line. Then subtract \( h/2 \) from each noncovered line. This results in \( h \) being subtracted from all noncovered elements, and \( h \) being added to all doubly covered elements. All entries that are only covered once remain unchanged.

4. Go to step 2.

This algorithm is terminating after a finite number of iterations: During all steps the entries of the matrix remain nonnegative integers. Execution of step 3 changes the sum of all entries by \( knh/2 - (2n - k)nh/2 = -(n - k)hn \). This means that a positive integer amount is subtracted from the sum of the matrix elements if \( k < n \). Therefore, this case only occurs a finite number of times.

For our example, steps 2 and 3 yield the following two matrices:
The independent zeros in these matrices are marked $0$, and lines indicate the constructed cover. The accumulated amount that has been subtracted from each line is shown next to the matrix. During the next iteration, we obtain the independent set in the second matrix. This set has cardinality $n$, which means that it already represents an optimal solution to our LAP.

2.3 Enumerating Optimal Assignments

The Hungarian method returns exactly one optimal solution to the LAP, even though more than one optimal solution might exist. We extend the algorithm for solving the LAP to obtain a complete set of solutions, even nonoptimal ones. Therefore we need an algorithm to enumerate all optimal solutions, and then suboptimal solutions. Suboptimal solutions are generated in order of increasing cost. We present Uno’s algorithm [25] for the enumeration of the complete set of optimal solutions given one initial optimal solution. Chegireddy and Hamacher’s algorithm [6] described in section 2.4 covers suboptimal solutions.

The Hungarian method produces a matrix $W' = w'_{ij}$ containing at least one set of independent zeros of cardinality $n$. Sums of added or subtracted on $W$ lines during transformations of the Hungarian method represent the weight of an optimal solution. As these transformations do not change the set of optimal permutations, there is a one-to-one correspondence between the complete independent sets of $W'$ and the optimal solutions to the LAP represented by $W$. For this reason, the problem of enumerating all optimal solutions of an LAP can be reduced to the problem of enumerating all complete sets of independent zeros of $W'$. 
In the bipartite graph representing the LAP, we can now remove all edges corresponding to nonzero entries in $W'$, and obtain the following graph $G'$:

\[
\begin{array}{cccc}
t_1 & t_2 & t_3 & t_4 \\
0 & 2 & 0 & 0 \\
0 & 0 & 9 & 2 \\
9 & 0 & 0 & 0 \\
0 & 5 & 8 & 0 \\
\end{array}
\]

→

Every perfect matching in $G'$ corresponds to a complete independent set in $W'$, and thus to a minimum weight perfect matching in the graph $G$, which represents our LAP.

As the starting point for Uno’s algorithm we need the graph $G'$ and a perfect matching $M$. We define a directed graph $D(G', M)$ by defining an orientation from $S$ to $T$ for every edge that is included in $M$, and from $T$ to $S$ for all remaining edges:

Uno showed that the problem of enumerating the perfect matchings in $G'$ can be solved by enumerating directed cycles of $D(G', M)$. An alternative optimal solution is obtained by inverting the direction of every edge along one or more nonoverlapping cycles. The edges directed from $S$ to $T$ then define the new matching.

Edges that do not belong to any cycle can be eliminated by a strongly connected component search. If, after this procedure, the graph still contains edges, there is some cycle which can be found by depth first search, and thus another perfect matching $M'$ can be obtained by inverting all edges along the cycle (indicated by edges $\Rightarrow$):
We now choose an edge $e$ with $e \in M$ and $e \notin M'$ and use $G'$ to construct two graphs $G^+(e)$ and $G^-(e)$ in order to find the remaining matchings. $G^+(e)$ is the graph obtained by deleting $e$ and all vertices and edges adjacent to it. $G^-(e)$ is the graph obtained by deleting $e$ from $G'$. $G^+(s_1 t_1)$ and $G^-(s_1 t_1)$ for our example are as follows:

All perfect matchings from $G'$ are contained in the two graphs. The matchings in $G^+(e)$ correspond to the matchings in $G'$ including $e$. $G^-(e)$ contains all perfect matchings in $G'$ that do not include $e$. We call the algorithm recursively with $D(G^+(e), M \setminus \{e\})$ and $D(G^-(e), M')$ to obtain the remaining matchings.

Given a graph $G$ and a matching $M$, the complete algorithm consists of the following steps:

Algorithm $\text{Enumerate\_Perfect\_Matchings}(G, M)$

1. Eliminate unnecessary edges from $G$ by a strongly connected component search.
2. Stop, if $G$ contains no edge.
3. Choose an edge $e \in M$.
4. Find a cycle containing $e$ using depth first search.
5. Output perfect matching $M'$ constructed by exchanging edges from $M$ along the cycle.
6. Call $\text{Enumerate\_Perfect\_Matchings}(G^+(e), M \setminus \{e\})$ to enumerate all perfect matchings in $G$ that contain $e$.
7. Call $\text{Enumerate\_Perfect\_Matchings}(G^-(e), M')$ to enumerate all perfect matchings in $G$ not containing $e$.

Steps 1 and 4 each takes $O(m + n)$ time for a graph with $m$ edges and $n$ vertices, and are therefore the critical steps for the time complexity of the algorithm. Even though in general $m$ is of the order of $n^2$, Uno describes a strategy for choosing an edge $e$ that results in an overall complexity of $O(nN_p)$ for finding all $N_p$ perfect matchings.
2.4 Enumerating Nonoptimal Assignments

The problem of enumerating all possible assignments in order of increasing cost is also known as the problem of finding $K$-best perfect matchings. Murty’s original algorithm [23] has been improved and generalized by Lawler in [18] and last by Chegireddy and Hamacher [6].

The idea is to partition the solution space $P$ of the minimum weight perfect matching problem for bipartite graphs iteratively into subspaces $P_1, \ldots, P_k$. For each of these subspaces $P_i$, the optimal solution $M_i$ and second best solution $N_i$, as well as second best matching problem are known. The $M_i$ are also known to represent the $k$-best perfect matchings within $P$. As the $P_i$ are a partition of the solution space $P$, the next best solution in $P$ has to be one of the second best matchings $N_i$, namely the matching with minimum weight $w_{N_i}$. Let $N_p$ be this matching with minimum weight among all second best matchings $N_i$, and $P_p$ the corresponding solution space. We partition the solution space $P_p$ into two subspaces $P'_p$ and $P_{k+1}$ such that $P'_p$ contains $M_p$ and $P_{k+1}$ contains $N_p$. Our $(k + 1)$-best matching $M_{k+1}$ is then set to $N_p$, and solution space $P_p$ is replaced by $P'_p$. As $N_p(= M_{k+1})$ is no longer included in $P_p$, we calculate new second best matchings $N_p$ and $N_{k+1}$ for the solution spaces $P_p$ and $P_{k+1}$. We are then prepared to proceed with finding the next best solution.

The algorithm requires a means to partition the solution space given two different solutions, and an algorithm to find the second best perfect matching in a bipartite graph, given a minimum weight perfect matching. For any two different perfect matchings $M_a$ and $M_b$, there is always an edge contained in $M_a$, which is not element of $M_b$. We can therefore use such an edge $e_k \in M_p - N_p$ to partition the solution space $P_p$ into $P'_p$, containing all solutions from $P_p$ that include $e_k$, and into $P_{k+1} \equiv P_p \setminus P'_p$. We can now represent every solution space $P_i$ by two edge sets $I_i$ and $O_i$, standing for edges that have to be included in any solution in solution space $P_i$ and edges that cannot take part in any solution in $P_i$, respectively. $P_i$ can then be defined as $\{M: M$ is a perfect matching, $I_i \subseteq M, O_i \cap M = \emptyset\}$.

The second best perfect matching in solution subspace $P_k$, provided the best perfect matching $M_k$ in $P_k$ is already known, can be found as follows. We define a graph $G_k$ obtained from the graph representation $G$ of our LAP by removing all edges $O_k$ and all edges and vertices incident to the edges in $I_k$ from $G$. We then orientate the edges of $G_k$ from $S$ to $T$ and set $w_{ij}$ to $-w_{ij}$ for every edge that is included in the matching $M_k$, and orientate from $T$ to $S$ all remaining edges. The weights of the latter remain unchanged. For the example from figure 1, edge sets $I_k = \{(s_2, t_2)\}$ and $O_k = \{(s_1, t_1)\}$, and matching $M_k = \{(s_1, t_3), (s_2, t_2), (s_3, t_4), (s_4, t_1)\}$ we obtain the following graph (dotted lines represent the deleted edges):
Every perfect matching in this graph $G_k$ together with the edges in $I_k$ is a perfect matching for $G$ and contained in $P_k$, as the $I_k$ are included by definition, and the edges in $O_k$ do not exist in $G_k$. On the other hand, every perfect matching in $P_k$ becomes a perfect matching for $G_k$ by removing the edges contained in $I_k$. This is due to the fact that no edge removed from $G$ during the construction of $G_k$ can be included in any matching in $P_k$. The second best perfect matching for $G_k$ therefore corresponds to the second best perfect matching in $P_k$. Similar to Uno’s algorithm for enumerating all perfect matchings with minimum weight, we obtain alternative perfect matchings by searching for a directed cycle in the graph and inverting the direction and weight of all edges along the cycle. The next best matching thereby corresponds to the cycle of minimum weight. This can be obtained by calculating the shortest paths for every vertex pair $(t_j, s_i)$. Besides the Floyd-Warshall algorithm, Chegireddy and Hamacher propose two other algorithms for this task, which all take $O(n^3)$ time to find the cycle with minimum weight.

The algorithm takes an undirected bipartite graph $G$ with edge weights $w_{ij}$ as input and outputs the $K$-best perfect matchings $M_1, \ldots, M_K$ in order of increasing weight.

1. Set $I_1 := \emptyset$, $O_1 := \emptyset$, $k := 1$.
2. Find best perfect matching $M_1$ and second best perfect matching $N_1$ in $G$.
3. Choose $N_p$ as matching with minimum weight $w_{N_p}$ among all second best perfect matchings $\{N_1, \ldots, N_k\}$.
4. Stop, if $w_{N_p} = 1$ (only $k$ matchings exist).
5. Set $M_{k+1} := N_p$.
6. Stop, if $k \geq K - 1$.
7. Choose $e_k \in M_p - N_p$.
8. Set $I_{k+1} := I_p$, $I_p := I_p \cup \{e_k\}$, $O_{k+1} := O_p \cup \{e_k\}$, and $k := k + 1$.
9. Find second best matchings $N_p$ and $N_k$ in solution spaces $P_p$ and $P_k$. If no second best matching exists, because $M_i$ is the only matching in $P_i$ ($i \in p,k$), set $w_{N_i} := \infty$. Continue with step 3.

The algorithm to find a second best perfect matching $N_i$, given the best perfect matching in solution space $P_i$ in step 9, takes $O(n^3)$ time. In every iteration we call this algorithm two times, thereby obtaining an overall complexity of $O(Kn^3)$ for the $(K - 1)$ iterations as stated by Chegireddy and Hamacher [6]. But we also have to take into account the operation of finding the second best matching $N_i$ with minimum weight in step 3. Using a priority queue, this
is possible in $O(\log k)$ with $k$ being the length of the queue. For the $K$ iterations this yields $\sum_{k=1}^{K} \log k = \log K!$, which is actually growing faster than the factor $K$ in $Kn^3$. Thus, for the time complexity of the algorithm as a function of $n$ and $K$, we obtain $O(Kn^3 + \log K!)$. For most practical applications of the algorithm however, this difference is not very important, as $K$ and its influence on the running time is relatively small: the dependency on the problem size $n$ is of greater interest.

3 A Complete Constraint Solver For the Assignment Problem

In this section we explain how we have modified the narrowing-based operational semantics of a declarative language in order to use a specific algorithm for a constraint that encodes the assignment problem. Since the algorithm is complete, the operational semantics remains complete too.

In order to obtain a single homogeneous and complete solver for the LAP, all of the three algorithms presented in section 2 are combined. At first, an optimal solution to the LAP is computed, then all required information in order to calculate the following solutions is stored. The solver returns the solution and a handle to the stored information. Afterwards, the next best solution to the LAP based on the stored information is computed.

The Hungarian method is used to calculate the initial assignment required by the two other algorithms. As Uno’s algorithm is a lot more efficient for enumerating optimal solutions than the algorithm by Chegireddy and Hamacher, this algorithm is used first. Then comes the enumeration of suboptimal solutions in order of increasing weight using the algorithm for enumerating $K$-best solutions.

The transition between the Hungarian method and Uno’s algorithm only consists of the construction of the graph $G'$. The transition from Uno’s algorithm to the algorithm of Chegireddy and Hamacher is more complicated, as we have to partition the solution space $P$ into $N_p$ subspaces, each containing one of the $N_p$ optimal solutions, which can be seen as the $N_p$-best solutions to the stated LAP.

We proceed similarly with the algorithm for finding the $K$-best solutions (see section 2.4). The solution space is partitioned by creating the two edge sets $I_i$ and $O_i$ for each optimal solution $M_i$, with $i \in 1, \ldots, N_p$. When the algorithm has finished, solution subspace $P_i$ is defined as the subspace containing exactly those solutions that contain all edges in $I_i$, and none of the edges in $O_i$.

The computation of $I_i$ and $O_i$ is done incrementally. The sets $I_i$ and $O_i$ are initially empty. Now suppose that we have a list of $k$ solutions $M_1, \ldots, M_k$ with corresponding solution spaces $P_1, \ldots, P_k$. The computation induced by another solution $M_{k+1}$ is as follows. We first have to find out, which solution space $P_i$ ($i = 1, \ldots, k$) $M_{k+1}$ belongs to. This can be done by checking, if all edges in $I_i$ are contained in $M_{k+1}$ and none of the edges in $O_i$ is contained in $M_{k+1}$. This process is repeated until the matching solution space $P_i$ is found. We then
proceed, as in the algorithm presented in section 2.4 for the insertion of the solution $M_{k+1}$. Once obtained the list of all $N_p$ optimal solutions, the second best solution is computed for each of solution spaces $P_1, \ldots, P_{N_p}$.

We now address the complexity of the complete constraint solver algorithm. By inserting a solution, two edges to the sets $I_i$ and $O_{k+1}$ are added, thus at the end, all of these sets together contain $2(N_p - 1)$ edges. This results in a time complexity of $O(N_p)$. Checking, whether a certain edge is contained in a solution, is possible in $O(1)$. This matching procedure must be carried out for each of the $N_p$ solutions, which yields a total complexity of $O(N_p^2)$. Next, the difference between two solutions must be found, each time a solution is inserted. It results in a complexity of $O(N_p n)$ for inserting all of the $N_p$ solutions. Finally, second best solution has to be computed for each of the $N_p$ solution spaces, which can be done in $O(N_p n^3)$. The total complexity for the transition is therefore $O(N_p^2 + N_p n + N_p n^3)$, which is the same as $O(N_p^2 + N_p n^3)$.

4 A Case Study: Timetabling of Medical Staff

Due to the complicated and often changing set of constraints, timetabling problems are very hard to solve, even for relatively small instances. This is why one of the more successful approaches in the area of timetabling is constraint logic programming. It is used to combine the advantages of declarative programming (facility to handle symbolic constraints) and efficient solver algorithms.

In this section we present a case study using our hybrid system. A first version of this case [3] was done with a standard evaluation strategy. In this older version the system was sometimes not able to find a solution even if one was theoretically possible. Thanks to our complete constraint solver no solution is missed anymore by our system.

The idea to get the best of both worlds, is to use a specific operations research algorithm for a particular constraint to be solved, and use the standard operational semantics for the other constraints. In our case study we use the algorithm from section 3 for LAP constraints and standard narrowing of conditional rewrite rules for the others.

4.1 Timetabling of Medical Staff

The solver proposed in section 3 has been tested on a real world application. The problem consists in the generation of a weekly timetable for the personnel of a Parisian hospital. There are ten shifts for a week, a morning and a late shift for each of the five workdays. The staff is composed of physicians and interns. They are to be assigned to the open services according to their qualifications. A shift can be represented as follows:

<table>
<thead>
<tr>
<th>Monday AM</th>
<th>Echography</th>
<th>Stomatology</th>
<th>Emergency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bill</td>
<td>Bob</td>
<td>Murray</td>
</tr>
</tbody>
</table>
This shift means that Bill is assigned to the service of Echography, Bob to the service of Stomatology and Murray to the service of Emergency on Monday AM.

In order to obtain an assignment of medical staff to the services, we use the algorithm of section 3. Therefore, the weight matrix of section 2.2 has to be interpreted this way: columns represent services and rows represent medical staff. Notice that since there are more staff members than services (if it is not the case, then the assignment problem is unsolvable), and since the LAP is defined for $n \times n$ matrices, we introduce virtual “off work” services. The weights are computed following the suitability of some staff member to some service. I.e. some medical doctors may prefer, or may be more qualified for certain services. Weights also depend on the assignments in other shifts. For example, an infinite weight (i.e. a very large value) is set when the assignment is physically impossible (staff member not working for instance) and semi-infinite weights when the assignment would lead to violation of administrative constraints.

A timetable is acceptable if it respects a set of constraints of very different kind and importance. First, all of the open services have to be assigned appropriately qualified staff members. For some services, on a given day, the staff has to be different for the morning and late shifts. For other services it is the opposite, the staff has to be the same all day long. Policies also state that no one should work more than three times on the same service during a week. Also, the workload should be distributed as fairly as possible. An additional constraint is that every staff member shall have two afternoons off work per week. These examples do not form an exhaustive list of all constraints (mandatory or not) required for an acceptable timetable.

At this point the reader may notice that there are two large classes of constraints. The first one gathers constraints which are local to a single shift. Basically, they consist in the LAP, i.e. once assigned to a service, a staff member cannot be assigned elsewhere. Pre-assignment, which is obtained by staff members when they give their schedule (day-off for the week), is also local. It consists mainly in the assignment (which is done by setting the weight to 0) of staff members to virtual services. The second class of constraints are the ones with inter-shifts scope (e.g. no more than three assignments per week on a same service). Those two classes are given a different treatment in our implementation and have interactions. Indeed, a locally optimal LAP solution may lead, because of inter-shifts constraints, to a globally inconsistent set of constraints. Suppose for instance that Bill has been assigned to the same service, say Echography, three times in a row from Monday AM, while Bob has been assigned to Stomatology. It is not possible anymore to assign Bill to the service of Echography. Now suppose that, later in the week, Bob is off work and that only Bill may be, due to other constraints, assigned to Echography. We have an impossible set of constraints to satisfy. Now, if Monday AM we switch Bill and Bob, a global solution may be possible, even if the assignment for the shift on Monday AM is sub-optimal. Therefore, it can be useful to find another locally optimal solution, or even a sub-optimal one, in order to achieve a global solution. Hence, the use of the algorithm from section 3 to enumerate all solutions is mandatory.
4.2 Implementation: Modifying Narrowing by Integrating a Complete LAP Solver

More precisely, our implementation is based on constraint functional logic programming. Programs are sets of conditional rules of the form $l \rightarrow r \mid c$: $l$ is rewritten into $r$ provided that condition $c$ is satisfied, see [20] for a detailed presentation. A goal is a multiset of basic conditions (usually equations). The operational semantics solves goals using narrowing: it produces a substitution $\sigma$ of variables that occur in the goal.

A cost is associated with the violation of every constraint imposed on an acceptable timetable. The more important the constraint is, the higher the cost is. For every shift a cost matrix is computed. It contains the costs for assigning each of the available staff members to the open services. The LAP solver is then used to find an assignment of the staff members to the services, which minimizes the total cost, using the Hungarian method. If an acceptable assignment for the shift has been found, the cost matrix for the following shift is calculated based on previous assignments. This process continues until a timetable for the entire week is obtained.

It is possible that for some shift, no acceptable assignment can be found by the Hungarian method due to assignments in previous shifts. In this case, the LAP solver needs to be called again in order to find alternative assignments for previously assigned shifts. This backtracking mechanism is integrated into the narrowing process.

Within the conditional rewrite system used to represent the constraints of an acceptable timetable, two rules are essential for timetable generation:

$$\text{timetable } tt \rightarrow \text{true } \mid (tt = \text{timetable\_scheme}) \text{ and } (\text{is\_timetable } tt)$$

$$\text{is\_timetable empty } \rightarrow \text{true}$$

$$\text{is\_timetable (cons (shift day period assignments) tail)}$$

$$\rightarrow \text{true } \mid (\text{is\_timetable tail}) \text{ and } (\text{linear\_assignment (compute\_matrix tail) assignments}).$$

The first rule (timetable) contains two conditions stating that an acceptable timetable, say $tt$, consists of ten shifts ($tt = \text{timetable\_scheme}$), and that these shifts have to contain assignments that respect the imposed constraints (is\_timetable $tt$). The second rule (is\_timetable) is recursively defined: An empty list of shifts represents a valid timetable. A nonempty list of shifts represents a valid timetable, if the tail of the list contains assignments that do not violate any of the constraints, and if the first shift contains valid assignments. The latter is ensured by the constraint linear\_assignment, which calls our algorithm for the LAP with the cost matrix that is calculated based on the previous assignments by compute\_matrix tail, and other constraints that are represented very naturally using rewrite rules. Sample rules are:
present (physician Gerd) Monday PM → true
open Echography Friday AM → false
unqualified (physician Fred) Emergency → true

The first rule declares that Gerd is present Monday PM, and therefore can be assigned to a service. The second rule declares that a service is closed, so no working physician for this shift should be assigned to this service. Finally the last rule declares that Fred is not qualified for the service of Emergency. Such rules are used to compute the weights of assignments.

Computation in constraint functional logic programming can be done by goal solving. A goal – like the condition in a conditional rewrite rule – thereby consists of a list of equations and constraints. A goal has been solved, if a substitution of the variables occurring in the goal has been found such that all equations hold and all constraints are satisfied.

The goal to generate a timetable is of the form \texttt{timetable x = true?} Thus, the narrowing process has to find a satisfying substitution for variable \(x\). For this purpose a stack of goals is maintained as well as partial substitutions that might lead to a complete satisfying substitution. It corresponds to a depth first traversal of the search space. At the start this stack only contains the initial goal entered. Then, for each narrowing step, the goal and the partial substitution on top of the stack are removed. A satisfying substitution for the first equation or constraint in the goal is looked for. For each such substitution, a new goal and partial substitution are put on top of the stack. Whenever a conditional rewrite rule is used in the narrowing step, its conditions have to be added to the goal. If a satisfying substitution for all of the variables has been found, it is displayed as a possible solution to the initial goal.

When computing a timetable, the goal \texttt{timetable x = true} is replaced with \(x = \text{timetable\_scheme}\) and \texttt{is\_timetable x = true}. Once the first equation has been solved by substituting \(x\) with a timetable scheme that contains variables for each service to fill, the rule \texttt{is\_timetable} is applied recursively. It goes on until the goal becomes a list of ten constraints \texttt{linear\_assignment}; one for each shift of the week.

A constraint solver that only uses the Hungarian method to find a solution to the assignment problem would solve these constraints one at a time. It computes a substitution that represents an optimal assignment and then removes the constraint from the goal, before it is put back on top of the stack for the next narrowing step. This strategy, however, is not complete, and in our example, as shown earlier, acceptable timetables might not be found, even though they exist.

In our current implementation, the Hungarian method is used initially to compute a solution to the constraint \texttt{linear\_assignment}. Instead of simply removing the constraint from the goal that is put back on top of the stack, two goals are put on the stack. The first goal still contains the constraint \texttt{linear\_assignment} for the shift that has just been solved, but it is put onto the stack together with the handle to the stored information that the LAP solver returned after the first call. The next time that this goal is removed from the stack, the LAP algorithm will return the next best assignment, if another ac-
acceptable assignment exists. The second goal to be put on top of the stack is the same as in an implementation only based on the Hungarian method. This goal is the next to be removed from the stack in the following narrowing step. The narrowing process in the case that acceptable assignments are found by the Hungarian method for every shift is thus the same as if only the Hungarian method was used as a constraint solver.

The difference between the two only becomes important, if for one of the shifts, no assignment satisfying all of the constraints that an acceptable timetable has to fulfill could be found. In this case, an implementation only based on the Hungarian method for solving the LAP would fail, as it could not solve the goal on top of the stack, and the stack would be empty. In our current implementation, there would be no acceptable assignment for the shift on top of the stack, either, but afterwards the stack would not be empty. Instead, the next goal on top would yield the next best solution for the previous shift.

After a solution has been found for all ten shifts, the program offers to search for alternative timetables by continuing the narrowing process. By first trying to find an assignment for each shift, and backtracking to the previous shift if no such assignment exists, the described strategy implements a simple depth first search of the solution space. At this point we have to point out that this strategy does not guarantee an optimal solution from a global point of view. Indeed, the best choice for a shift may lead to a branch in which global cost is actually higher than if a less optimal solution would have been chosen.

The Hungarian Method returns a solution that minimizes the total cost of the assignments for one block based on the assignments in previous shifts. For a given cost matrix, there might exist several such optimal solutions with minimum cost. For each of those solutions we might obtain different cost matrices for the subsequent shifts. As we do not know, which of the optimal solutions will allow the least expensive assignments for the remaining shifts, and as we do not raise the cost by exchanging one optimal assignment for another, we have to consider equally all of the optimal assignments for a block.

But what about the suboptimal assignments? Do we have to consider them as well in order to guarantee the finding of an acceptable timetable whenever there exists one? The following example shows that even by considering all possible optimal solutions for the blocks, we might not find a globally optimal solution. If the cost of this globally optimal solution equals the maximum cost of an acceptable timetable, every timetable with higher cost would not be acceptable. Thus, an acceptable timetable will not be found without considering suboptimal solutions for certain blocks as well.

We consider an example with four shifts, two physicians $M_1$ and $M_2$, and one open service $S$. $M_1$ is absent in the first shift, $M_2$ is only available for the first two shifts. In order to prefer physicians that have not yet been assigned to a certain service, we define the cost for assigning a staff member to be the number of previous assignments to this service, if no other constraints are violated. For this example we allow every physician to be assigned only twice to a given service, penalizing a third assignment with a cost of 5000. With an underlined number
representing the selected assignment, we obtain the following cost matrices:

\[
\begin{bmatrix}
0 \\
M_2
\end{bmatrix}
\quad
\begin{bmatrix}
0 \\
M_1
\end{bmatrix}
\quad
\begin{bmatrix}
1 \\
M_1
\end{bmatrix}
\quad
\begin{bmatrix}
5000 \\
M_1
\end{bmatrix}
\]

The total cost for this timetable is the sum of all underlined numbers, which is \(0 + 0 + 1 + 5000 = 5001\). The only choice we have in our assignments is in the second block, where we could choose the suboptimal solution of assigning physician \(M_2\) to the service:

\[
\begin{bmatrix}
0 \\
M_2
\end{bmatrix}
\quad
\begin{bmatrix}
0 \\
M_1
\end{bmatrix}
\quad
\begin{bmatrix}
1 \\
M_1
\end{bmatrix}
\quad
\begin{bmatrix}
1 \\
M_1
\end{bmatrix}
\]

This solution has total cost of \(0 + 1 + 0 + 1 = 2\) and is therefore cheaper than the one found by only considering optimal solutions for each block. As the presented assignments are the only two assignments for our case, the second one has to be the globally optimal solution. One could argue that by changing the order of assignments for the blocks, we would have found this assignment by only considering optimal solutions, but that is only true for our example. By constructing a more complicated example, we could prove that there are cases in which for any order of blocks, the globally optimal solution for the problem can only be found by considering suboptimal solutions as well. We do not want to pursue this idea, but instead we conclude that for a fixed order of assignment of the blocks we have to consider suboptimal solutions for some shifts to be sure to find an acceptable timetable for the whole week in any case where such a timetable exists.

For the following, we assume that we can decide if a given timetable is valid or not by adding the cost for the violated constraints, even without knowing the order, in which the shifts have been assigned\(^1\). In this case, we obtain a complete timetabling algorithm that finds all acceptable schedules, if only we consider suboptimal solutions for the blocks until the cost for a block gets higher than the maximum cost for a timetable. If the optimal assignment for a block has a higher cost than the maximum cost for a timetable, we can be sure that already the subset of assigned shifts is violating more constraints than allowed. Hence, regardless of later assignments, we can never obtain a valid timetable and thus have to step back and consider an alternative assignment for a previous block.

### 4.3 The Solution Search Tree

With these observations, we are now ready to define the backtracking policy. First, we use a fixed order of assigning the shifts. We define two constants

\(^1\) This is actually our intention, even though we could construct examples, in which the total cost differs slightly depending on the order of assignment.
failure\_bound and backtrack\_bound. The meaning of the first constant is that we do not dispose of enough qualified staff for a shift, if the minimum cost of assignment is at least equal to failure\_bound. This problem cannot be solved by trying different assignments for previous shifts, and so we can stop the search for a valid timetable, as we can be sure that there is none. If the minimum cost for a certain shift equals or exceeds the second constant, backtrack\_bound, this means that the assignments that have been found so far cannot be part of any acceptable timetable, even though with different assignments there might exist one.

The search for a solution is then carried out in the following way. For every shift, we calculate the optimal assignment based on the previously assigned blocks. If for one shift the cost of this optimal assignment is greater or equal than backtrack\_bound, we return to the previous block and try an alternative optimal solution and continue with the next block. If no more optimal solutions exist, we try suboptimal solutions until the cost exceeds backtrack\_bound, in which case we step back another block. If for some shift the minimum cost of assignment equals or exceeds failure\_bound, we stop the search.

We give an example to illustrate the search of a timetable using the presented backtracking scheme. Figure 2 shows a possible search tree for a timetable consisting of five shifts. A solution $A_i$ stands for an assignment for one shift. The indices represent the order in which these solutions are found. For each solution, we state if the solution is optimal with respect to the previously assigned blocks ($o$), suboptimal ($s$) or if the assignment is not acceptable ($f$).

We start by finding an assignment $A_1$ of minimum cost for the first block. We find optimal assignments $A_2$ and $A_3$ for blocks 2 and 3. But in the fourth block it turns out that there is no valid timetable for the whole week containing the assignments $A_1$–$A_3$, because the least expensive solution $A_4$ already exceeds the value of backtrack\_bound. We step back to block three, find assignments $A_5$ and $A_7$, which also cannot be extended to a valid timetable. $A_3$, $A_5$ and $A_7$ represent the only optimal solutions for block three given the assignments $A_1$ and $A_2$. So we have to try suboptimal assignments. We find the next cheapest assignment $A_9$, which already exceeds backtrack\_bound. We now know that even the assignments $A_1$ and $A_2$ cannot be extended to an acceptable timetable, so we have to consider alternative substitutions for block 2. It turns out that $A_2$ was the only optimal assignment for block 2, hence we try suboptimal solutions. The next solution cannot be extended to a valid timetable either, but assignment $A_{12}$ can. Finally, we find the acceptable solution consisting of assignments $A_1$, $A_{12}$, $A_{13}$, $A_{14}$ and $A_{15}$. We could now try to find alternative solutions by first re-assigning one of the five blocks, and then completing the timetable by finding assignments for the subsequent shifts.

The solution search strategy is complete with respect to finding acceptable timetables whenever they exist. For the first block, we try all possible assignments in order of increasing cost, until the cost of a solution exceeds the value of the constant backtrack\_bound. Thus, every assignment for the first shift, that could be part of a timetable for the whole week, is considered. For each of these
Fig. 2. A possible search tree for a timetable for five shifts
assignments, we try all assignments for the next shift that are consistent with those of the first block and could possibly yield a valid timetable for the whole week. This strategy is pursued for each of the ten blocks. Thus, we only cut branches of the search tree, if the assignments in the current subset of shifts already violate too many constraints. We can therefore be sure not to ignore any acceptable solution in our search.

For a more precise presentation of the complete set of inference rules for goal transformation we refer to the appendix A. Let $G$ be a goal. Our proposition consists in giving a particular instance of the following rule where $C_{sol}(p(t_1, \ldots, t_n))$ is computed by the algorithm from section 3:

\[
\frac{|p(t_1, \ldots, t_n)| \cup G}{\sigma(G) \quad \text{if } \sigma \in C_{sol}(p(t_1, \ldots, t_n))}
\]

with $C_{sol}(p(t_1, \ldots, t_n))$ being a complete set of solutions for constraint $p(t_1, \ldots, t_n)$

where $p$ is an $n$-ary constraint. The modification has to take into account all of the solutions in $C_{sol}(p(t_1, \ldots, t_n))$, not only a single one, when $p$ corresponds to the constraint solvable by the algorithm described in section 3.

### 4.4 Practical results

We have tested the implementation on several generated test cases with real data from the planning of a week for a Parisian hospital. The real world example involved around 20 staff members that had to be assigned to 11 services. An implementation only using the Hungarian method would have failed finding a solution for this problem. The implementation using the complete LAP algorithm was able to find an acceptable timetable with one backtracking step in less than 20 seconds on a 500MHz Pentium III PC. For the generated cases, the program found a solution to a part of the problems within less than five minutes. For others, no solution was found within reasonable time. This was partly due to overconstrained problems, for which no acceptable timetables existed, and partly due to the backtracking strategy. The simple depth-first search that we have implemented performs very well for assignment conflicts that are caused by the assignments made in the previous few shifts. But it is bad for resolving constraints that require re-assignments for shifts that have been assigned relatively early in the goal solving process.

For the problem of timetable generation, the program has to be tested on more real data in order to determine its performance in practice. Also, it would give some material to try other heuristics for a better traversal of the search space. Random test cases are not sufficient for evaluating performance on real data. In cases, where a timetable cannot be found in reasonable time, the program can still be used to generate a timetable by allowing more constraints to be violated. This might already facilitate the process of finding acceptable timetables, if the program is used as a supporting tool.
5 Conclusion

We have proposed a combined algorithm for a complete enumeration of LAP solutions in order of increasing cost. We have discussed its integration into a constraint functional logic programming language. This system has been tested on a real world problem of timetabling for medical staff. It improves earlier approaches of this kind of problems (with a highly complicated hierarchy of constraints) by combining the expressive facility of declarative programming and the efficiency of operations research algorithms.

The presented depth-first traversal of the solution space performs well for constraints that can be resolved by re-assigning a shift that has been assigned only a few steps before, whereas it does not perform very well for global constraints. Thus, the search for solutions satisfying all of the stated constraints might be improved by employing different backtracking heuristics and strategies. Finding a generally applicable heuristic might be a difficult, if not impossible, task. Furthermore, in order to maintain the achieved completeness of the proposed solver, alternative backtracking approaches would either require much space for storing abandoned search paths, or the same paths would have to be computed more than once.

In the recent proposals for the solution of timetabling problems one can distinguish between two main approaches ([5, 19]). One is based on optimization techniques from operation research like simulated annealing, genetic algorithms and tabu search (e.g. [2, 4, 16, 26]). The second approach is based on constraint programming (e.g. [1, 21, 22]). But to our knowledge our proposition mixing declarative programming and the use of an extended version of the Hungarian method enumerating all solutions (thus retaining completeness) is a novelty.

Several directions for future works are possible. For instance taking into account dynamic constraints as in the on-line management of a fleet of vehicles. For this we would have to modify the operational semantics of a system combining declarative programming and concurrency such as [7]. Another direction would be to enhance the techniques for an efficient traversal of the search space. As we mentioned in section 4.4, we need more practical tests to develop such heuristics. It could be useful to consider other works in this field, for instance [10, 9].

References

3. N. Brauner, R. Echahed, G. Finke, F. Prost, and W. Serwe. Integration des methodes de reecriture et de recherche operationnelle pour la modelisation et la resolu-


Appendix

A Overview of narrowing rules

Narrowing is similar to rewriting. Instead of looking for occurrences of an instance $\sigma(l)$ of a left hand side of some rule in a given term $t$, a most general unifier $mgu(u, l)$ of some subterm $u$ of $t$ and a rule’s left hand side $l$ is computed. $u$ must not be a variable.

Let us start with the definition of narrowing for basic rewriting first. Let $t$ be a term containing at position $\alpha$ a non-variable subterm $t|_{\alpha}$ ($t|_{\alpha} \notin V$). If

1. there is a variant of a rewrite rule $l \rightarrow r \in R$ sharing no variables with $t$ ($\text{Var}(t) \cap (\text{Var}(l) \cup \text{Var}(r)) = \emptyset$)
2. there is a unifier $\sigma$ of $t|_{\alpha}$ and $l$ ($\sigma(t|_{\alpha}) \equiv \sigma(l)$)

we say that $t$ is narrowable to the term $s = \sigma(t[r]_{\alpha})$, and we write $t \leadsto_{\alpha, l \rightarrow r, \sigma} s$.

Similarly to rewriting, we can define the notions of a narrowing relation defined
by $\sim_{\alpha_1,l \rightarrow r_1}$ and of narrowing derivation consisting of a sequence of narrowing steps $t_0 \sim_{\alpha_1,l_1 \rightarrow r_1,\sigma_1} t_1 \sim_{\alpha_2,l_2 \rightarrow r_2,\sigma_2} t_2 \ldots$.

We define the notion of solution for a basic condition in the context of narrowing. A substitution $\sigma$ is a solution of $s_0 \Downarrow t_0$ computed by narrowing if and only if there exists a narrowing derivation $s_0 \Downarrow t_0 \sim_{\alpha_1,l_1 \rightarrow r_1,\sigma_1} \ldots \sim_{\alpha_n,l_n \rightarrow r_n,\sigma_n} s_n \Downarrow t_n$ such that $s_n$ and $t_n$ are unifiable constructor terms and $\sigma = \theta \circ \sigma_n \circ \ldots \circ \sigma_1$ where $\theta$ is a most general unifier of $s_n$ and $t_n$.

In order to allow the integration of external constraint solvers into the programming environment, we extend the notion of a basic condition by also allowing constraints of the form $p(t_1, \ldots, t_n)$. For each such constraint we assume that there is an implementation of an algorithm that can be used to solve the problem expressed in the form of the terms $t_1, \ldots, t_n$. The constraint is said to be solved, if the algorithm achieves to find a solution to the problem, consisting of a substitution $\sigma$ for the variables occurring in $t_1, \ldots, t_n$. A complete set of solutions for the constraint $p(t_1, \ldots, t_n)$ is a set from which one can deduce each substitution $\sigma$ that represents a solution to the constraint $p(t_1, \ldots, t_n)$.

A goal is defined as a multiset of basic conditions and therefore has the same form as the condition in a rewrite rule. A goal is said to be solved if it is empty. Provided a narrowing based goal solving mechanism, we obtain a framework for declarative programming, powerful enough to express each of the paradigms of functional, logic and constraint programming. It can also occur, that a stated goal is impossible to solve. If it is obvious that no solution exists, our goal solver should indicate failure. In general however, the decision problem whether a goal can or cannot be solved is undecidable. A complete set of solutions of a goal $G$ is composed of all substitutions $\sigma$ that solve the goal $G$.

Inference rules for goal solving are now defined. All substitutions occurring in the goal solving process are stored and represent the returned solution. In the following definitions $G$ stands for a goal, $C$ for the set of constructor symbols, $D$ for the set of defined functions, and $V$ for the set of variables.

**Rule 1** Constrained narrowing:

$$\left[ s \Downarrow t \right] \cup G \quad \overline{\sigma(s[r]_{\alpha}) \Downarrow \sigma(t) \cup \sigma(c) \cup \sigma(G)} \quad \text{if } \sigma(s) \equiv \sigma(l), \ s \notin V$$

and $(l \dashv c \rightarrow r)$ is a variant of a rule in $R$.

**Rule 2** Solution of constraints:

$$\left[ p(t_1, \ldots, t_n) \right] \cup G \quad \overline{\sigma(G)} \quad \text{if } \sigma \in C_{sol}(p(t_1, \ldots, t_n))$$

with $C_{sol}(p(t_1, \ldots, t_n))$ being a complete set of solutions for constraint $p(t_1, \ldots, t_n)$.

**Rule 3** If a basic condition is satisfied, we can eliminate it from the goal:

$$\left[ s \Downarrow t \right] \cup G \quad \overline{\sigma(G)} \quad \text{if } \sigma \text{ is a most general unifier of } s \text{ and } t$$

and if $s$ and $t$ are constructor terms.
Rule 4  If one of the two terms of a basic condition $x \downarrow t$ is a single variable $x$ that does not occur in $t$, and the other side $t$ is a constructor term, or a term that cannot be narrowed any further by application of rule 1, we can substitute $x$ by $t$:

\[
\frac{[x \downarrow t] \cup G}{\sigma(G)} \quad \text{with } \sigma = \{x \mapsto t\}, \text{ if } x \not\in \text{Var}(t)
\]

and $t$ is a constructor term or not narrowable

Rule 5  If one of the two terms of a basic condition $x \downarrow t$ is a single variable $x$ occurring in $t$, and the other side $t$ is a constructor term, or a term that cannot be narrowed any further by application of rule 1, no solution is possible:

\[
\frac{[x \downarrow t] \cup G \quad \text{if } x \in \text{Var}(t)}{\text{failure} \quad \text{and } t \text{ is a constructor term or not narrowable}}
\]

Rule 6  If one of the two terms of a basic condition $x \downarrow t$ is a single variable $x$, and the other side $t = c(t_1, \ldots, t_n)$ is a constructor rooted, narrowable term, we can substitute the basic condition as follows:

\[
\frac{\bigcup [x \downarrow c(t_1, \ldots, t_n)] \cup G \quad \sigma = \{x \mapsto c(x_1 \ldots x_n)\}, \\
[x_1 \downarrow \sigma(t_1)] \cup \ldots \cup [x_n \downarrow \sigma(t_n)] \quad x_1 \ldots x_n, \text{ freshes} \\
\cup \sigma(G) \quad c \in C \quad \text{and} \\
c(t_1, \ldots, t_n) \text{ narrowable}}{G}
\]

Rule 7  We can eliminate the outermost function symbol if it is a constructor and the same for the two terms of a basic condition $f(t_1, \ldots, t_n) \downarrow f(u_1, \ldots, u_n)$:

\[
\frac{[f(t_1, \ldots, t_n) \downarrow f(u_1, \ldots, u_n)] \cup G \quad \text{if } f \in C}{[t_1 \downarrow u_1] \cup \ldots \cup [t_n \downarrow u_n] \cup G}
\]

Rule 8  There is no solution, if the outermost function symbols $f$ and $g$ of the two terms of a basic condition $f(t_1, \ldots, t_m) \downarrow g(u_1, \ldots, u_n)$ are different and the two terms cannot be narrowed any further by application of rule 1:

\[
\frac{[f(t_1, \ldots, t_m) \downarrow g(u_1, \ldots, u_n)] \cup G \quad \text{if } f \in C \text{ or } (f \in D} \\
\quad \text{and } f(t_1, \ldots, t_m) \text{ not narrowable}, \\
g \in C \text{ or } (g \in D} \\
\quad \text{and } g(u_1, \ldots, u_n) \text{ not narrowable} \\
\quad \text{and } f \neq g}{\text{failure}}
\]
B Store hospital

This is the program for timetable generation containing the real planning data.

store hospital

(*
  ===============================================
  constants (controlling the hungarian assignment algorithm)
 *)

(* infinity: il est impossible de trouver une solution => EXIT *)
infinity :: integer.
infinity => 100000.

(* semi_infinity: ce n'est pas une solution correcte => BACKTRACK *)
semi_infinity :: integer.
semi_infinity => 5000.

(* backtrack_bound: borne a partir de laquelle il faut faire un backtrack *)
backtrack_bound :: integer.
backtrack_bound => semi_infinity.

(* failure_bound: borne a partir il faut signaler failure *)
failure_bound :: integer.
failure_bound => infinity.

(* max_distance: distance maximale toleree entre deux solutions *)
max_distance :: integer.
max_distance => 1000.

(*
  ===============================================
  booleans
  a enlever
 *)

not :: bool -> bool.
not true => false.
not false => true.
data Week_Day = Monday | Tuesday | Wednesday | Thursday | Friday.
data Day_Period = AM | PM.

data a = empty | (cons a (list a)).

hd :: (list a) -> a.
hd (cons h t) => h.

tail :: (list a) -> (list a).
tail (cons h t) => t.

is_empty :: (list a) -> bool.
is_empty empty => true.
is_empty (cons h t) => false.

is_in :: a -> (list a) -> bool.
is_in x empty => false.
is_in x (cons h t) | ((x = h) == true) => true.
is_in x (cons h t) | ((x = h) == false) => is_in x t.

(*
   lists
*)
dans les règles sur les staff.

Il indique que le service est ferme donc pas de personnel à affecter. Il apparaît dans les blocs schedules lors de la génération du timetable préaffecte.

*)

data Medecin = Abidat | Alerte | Attali | Brauner | Dumas | Elalouf | El_Jarrari | Hariz | Jelassi | Krief | Le_Van_Am | Mouffok | One | Rety | Safa | Touati | Vignot.

data Interne = Eiss | Valadier.

data Staff = (Medecin Medecin) | (Interne Interne) | no_staff_because_closed.

staff_list :: (list Staff).
staff_list =>
(cons (Medecin Abidat)
(cons (Medecin Alerte)
(cons (Medecin Attali)
(cons (Medecin Brauner)
(cons (Medecin Dumas)
(cons (Medecin Elalouf)
(cons (Medecin El_Jarrari)
(cons (Medecin Hariz)
(cons (Medecin Jelassi)
(cons (Medecin Krief)
(cons (Medecin Le_Van_Am)
(cons (Medecin Mouffok)
(cons (Medecin One)
(cons (Medecin Rety)
(cons (Medecin Safa)
(cons (Medecin Touati)
(cons (Medecin Vignot)
(cons (Interne Eiss)
(cons (Interne Valadier)
empty))})))))))))))

(*

======================================================================

services

A MODIFIER
data Service = Examens_Urgences | Echo1 | Echo2 | Vasculaire_vasculaire | Vasculaire_os | Vasculaire_os__TSA | Vasculaire_examens | Vasculaire_infiltrations | Mammographie | Scanner | IRM.

service_list :: (list service).
service_list =>
  (cons Examens_Urgences
   (cons Echo1
    (cons Echo2
     (cons Vasculaire_vasculaire
      (cons Vasculaire_os
       (cons Vasculaire_os__TSA
        (cons Vasculaire_examens
         (cons Vasculaire_infiltrations
          (cons Mammographie
           (cons Scanner
            (cons IRM
             empty)))))))))))).

data Staffing_Policy = SP_Interne | SP_Medecin | SP_Medecin_Interne.

(*
  =================================================================================================
  paires

  Ici il n’a pas ete defini de type generique de doublet (ie pair)
  Aussi il existe plusieurs constructeurs de doublets.
  A chaque doublet est associe une liste de ces doublets.
  *)

(*
  Pair correspond au couple Service Staff-list, representant la liste de
  medecins affectes a un service.
  *)
data Pair = (Pair Service (list Staff)).

get_staff_of_pair :: Pair -> (list Staff).
get_staff_of_pair (Pair s l) => l.

(*
  Pair_int_int corresponds to a pair of two integers
data Pair_int_int = (pair_int_int Int Int).

(*
  inc_first, inc_second, inc_both
  incrementing the first, second or both elements of a pair of integers *
*)
inc_first :: Pair_int_int -> Pair_int_int.
inc_first (pair_int_int x y) => (pair_int_int (x+1) y).
inc_second :: Pair_int_int -> Pair_int_int.
inc_second (pair_int_int x y) => (pair_int_int x (y+1)).
inc_both :: Pair_int_int -> Pair_int_int.
inc_both (pair_int_int x y) => (pair_int_int (x+1) (y+1)).

(*
  Pair_staff_int correspond au couple Staff int, representant une case de
  la matrice de couts associe a un staff.
*)
data Pair_staff_int = (pair_staff_int Staff integer).

(*
  Pair_serv_list represente le cout de l’affectation d’un
  staff a un service donne.
*)
data Pair_serv_list = (pair_serv_list Service (list Pair_staff_int)).

(*
  ==========================================================================
  blocks

  un block correspond au planning d’une demi-journee
  un planning est ainsi une liste de 11 blocks
  *)
data Block = (Block Week_Day Day_Period (list Pair)).

(*
  ==========================================================================
  time-tables (or schedules)
  *)
"main function": check/generate a time-table for a given date
*)
timetable :: (list Block) -> bool.
timetable schedule | (schedule == empty_schedule) & (is_timetable schedule)
 => true.

(*
 check a time-table for correctness
 *)
is_timetable :: (list Block) -> bool.
is_timetable empty => true.
is_timetable (cons (Block day period schedule) block_list) |
 (is_timetable block_list) &
 (hungarian_linear_assignment
 (compute_matrix
 (open_and_not_assigned service_list day period)
 (present_and_not_assigned staff_list day period)
 block_list
 day
 period)
 (open_and_not_assigned service_list day period)
 (present_and_not_assigned staff_list day period)
 schedule
 backtrack_bound
 failure_bound
 max_distance)
 => true.

(*
 au titre d’information

 appel a l’algorithme d’affectation
 Hungarian_Linear_assignment
 (m, seq1, seq2, bs, backtrack_bound, Failure_bound, max_eкарт)

corresponding type declaration:
 Hungarian_Linear_assignment ::
 (list Pair_serv_list) -> (list Service) -> (list Staff) ->
 (list Pair) -> Int -> Int -> Int -> (list Pair) -> bool_cond.
*)
(* generate an "empty" time-table  
this function is not "general", since it generates directly a time-table  
in the chronological order. might be improved to generate an arbitrary  
ordering of the blocks. see also the comments on the function  
already_assigned_same_day  
*)

empty_schedule :: (list Block).
empty_schedule =>
  (cons (empty_block Monday AM)
  (cons (empty_block Monday PM)
  (cons (empty_block Tuesday AM)
  (cons (empty_block Tuesday PM)
  (cons (empty_block Wednesday AM)
  (cons (empty_block Wednesday PM)
  (cons (empty_block Thursday AM)
  (cons (empty_block Thursday PM)
  (cons (empty_block Friday AM)
  (cons (empty_block Friday PM)
  empty))))))))).

(*  
generate an empty block  
*)

empty_block :: Week_Day -> Day_Period -> Block.
empty_block day period =>
  (Block day period (empty_block_sub day period service_list)).

empty_block_sub :: Week_Day -> Day_Period -> (list Service) -> (list Pair).
empty_block_sub day period empty => empty.
empty_block_sub day period (cons serv s_list) =>
  (cons (Pair serv (staff_preassigned serv day period))
  (empty_block_sub day period s_list)).

(*  
======================================================================  
interface to the hungarian assignment  
*)

(*  
compute the (cost-) matrix for one block  
* )
compute_matrix ::
(list Service) -> (list Staff) -> (list Block) -> Week_Day -> Day_Period ->
(list Pair_serv_list).

compute_matrix empty staffs blocks day period => empty.
(* one person required for service: *)
compute_matrix (cons service services) staffs blocks day period |
((staffing_policy_for_service service) = SP_Medecin_Interne) == false =>
cons (pair_serv_list service (row service staffs blocks day period
(staffing_policy_for_service service)))
(compute_matrix services staffs blocks day period).

(* two persons required for service: *)
compute_matrix (cons service services) staffs blocks day period |
((staffing_policy_for_service service) = SP_Medecin_Interne) =>
(cons (pair_serv_list service (row service staffs blocks day period SP_Medecin))
(cons (pair_serv_list service (row service staffs blocks day period SP_Interne))
(compute_matrix services staffs blocks day period))).

row ::
Service -> (list Staff) -> (list Block) -> Week_Day -> Day_Period ->
Staffing_Policy -> (list Pair_staff_int).

row service empty blocks day period sp => empty.
row service (cons person staffs) blocks day period sp |
not (person_matches_policy person sp) =>
(cons (pair_staff_int person infinity)
(row service staffs blocks day period sp)).
row service (cons person staffs) blocks day period sp |
(person_matches_policy person sp) &
(unqualified person service) =>
(cons (pair_staff_int person infinity)
(row service staffs blocks day period sp)).
row service (cons person staffs) blocks day period sp |
(person_matches_policy person sp) &
((unqualified person service) == false) &
(may_not_work_due_to_am_pm service person blocks day period) =>
(cons (pair_staff_int person semi_infinity)
(row service staffs blocks day period sp).
row service (cons person staffs) blocks day period sp |
(person_matches_policy person sp) &
((unqualified person service) == false) &
((may_not_work_due_to_am_pm service person blocks day period) == false) =>
(cons (pair_staff_int person (penalty_for_prev_assignments
(no_prev_assignments person service period blocks)))
(row service staffs blocks day period sp)).

(*

person_matches_policy
states, if a person matches the staff policy requirements for a service

Attention! Because there are only two interns, we accept physicians on
intern posts as well!
*)

person_matches_policy :: Staff -> Staffing_Policy -> bool.

person_matches_policy (Medecin x) SP_Medecin => true.
person_matches_policy (Medecin x) SP_Interne => true.
person_matches_policy (Interne x) SP_Medecin => false.
person_matches_policy (Interne x) SP_Interne => true.

(*

open_and_not_assigned
compute the list of open, unassigned services (for a given block)
note:
if the person assigned is not present, the assignment should be ignored
THIS IS TILL TO BE IMPLEMENTED !
*)

open_and_not_assigned ::
(list Service) -> Week_Day -> Day_Period -> (list Service).

open_and_not_assigned empty day period => empty.
open_and_not_assigned (cons s services) day period |
((open s day period) and (not (is_service_preassigned s day period))) =>
(cons s (open_and_not_assigned services day period)).
open_and_not_assigned (cons s services) day period |
(((open s day period) and
(not (is_service_preassigned s day period))) == false) =>
(open_and_not_assigned services day period).

(* present_and_not_assigned
   retourne la liste des medecins disponibles pour un block donne *)

present_and_not_assigned ::
   (list Staff) -> Week_Day -> Day_Period -> (list Staff).
present_and_not_assigned empty day period => empty .
present_and_not_assigned (cons person staffs) day period |
   ((present person day period) and
   (not (is_staff_preassigned person day period))) =>
   (cons person (present_and_not_assigned staffs day period)).
present_and_not_assigned (cons person staffs) day period |
   (((present person day period) and
   (not (is_staff_preassigned person day period))) == false) =>
   (present_and_not_assigned staffs day period).

(*
   is_staff_service_in_block
   verifie si person est deja affecte a service s dans block *)

is_staff_service_in_block :: Staff -> Service -> Block -> bool.
is_staff_service_in_block person s (block day period schedule) =>
   is_in person (staff_of_block_for_service s schedule).

(*
   is_staff_assigned_in_block
   verifie si person est deja affecte dans block *)

is_staff_assigned_in_block :: Staff -> (list Pair) -> bool.
is_staff_assigned_in_block person empty => false.
is_staff_assigned_in_block person (cons (Pair serv stf) bs_list) |
   (is_in person stf) =>
true.
is_staff_assigned_in_block person (cons (Pair serv stf) bs_list) | |((is_in person stf) == false) => (is_staff_assigned_in_block person bs_list).

(*
staff_of_block_for_service
*)

staff_of_block_for_service :: Service -> (list Pair) -> (list Staff).
staff_of_block_for_service serv empty => empty.
staff_of_block_for_service serv (cons (Pair pserv stf) bs_list) | |serv=pserv => stf.
staff_of_block_for_service serv (cons (Pair pserv stf) bs_list) | |(serv=pserv) == false => (staff_of_block_for_service serv bs_list).

(*
already_assigned_same_day :
indique pour period = AM si une personne a deja ete affecte au meme service pour l’apres-midi de la meme journee.
la definition actuelle de cette fonction necessite de construire la liste des blocks (voir fonction empty_schedule) tel que les apres-midis sont les successeurs immiedats des matinees correspondantes.
*)

already_assigned_same_day ::
Service -> Staff -> (list Block) -> Week_Day -> Day_Period -> bool.

already_assigned_same_day service person empty day period => false.
already_assigned_same_day service person (cons b blocks) day PM => false.
already_assigned_same_day service person (cons b blocks) day AM => (is_staff_service_in_block person service b).

(*
may_not_work_due_to_am_pm:
Indicates, if a person may not work on the given post due to the regulation that the staff has to be different/the same for morning and late shift.

Do not assign someone to a post requiring the same staff for the whole day, who isn't present all day long.
See also comment to already_assigned_same_day

*)

may_not_work_due_to_am_pm ::
  Service -> Staff -> (list Block) -> Weed_Day -> Day_Period -> bool.

may_not_work_due_to_am_pm service person empty day period => false.
may_not_work_due_to_am_pm service person blocks day PM |
  (same_staff_pm service) and (not ((present person day AM) and
  (not (is_staff_preassigned person day AM)))) => true.
may_not_work_due_to_am_pm service person blocks day PM |
  (same_staff_pm service) and (not ((present person day AM) and
  (not (is_staff_preassigned person day AM)))) == false => false.
may_not_work_due_to_am_pm service person blocks day AM |
  (same_staff_pm service) & (open service day PM) =>
  not (already_assigned_same_day service person blocks day AM).
may_not_work_due_to_am_pm service person blocks day AM |
  (same_staff_pm service) and (open service day PM)) == false =>
  already_assigned_same_day service person blocks day AM.

(*

    number_of_assignments_to_service:
    calcule le nombre de fois qu’un medecin a ete affecte pour un service donne
*)

(*
number_of_assignments_to_service :: Staff -> Service -> (list Block) -> Int.
number_of_assignments_to_service person s empty => 0.
number_of_assignments_to_service person s (cons b blocks) |
  (is_staff_service_in_block person s b) =>
  (1 + (number_of_assignments_to_service person s blocks)).
number_of_assignments_to_service person s (cons b blocks) |
  ((is_staff_service_in_block person s b) == false) =>
  (number_of_assignments_to_service person s blocks).
*)

(*
penalty_for_previous_assignments
checks the number of assignments of a person to a given service and the
number of assignments on afternoons. If both are smaller than 3, the max
is returned, otherwise a constraint is not satisfied, and semi_infinity
is returned.
*)

penalty_for_prev_assignments :: Pair_int_int -> Int.
penalty_for_prev_assignments (pair_int_int no_staff_service no_afternoons) |
(no_staff_service >= no_afternoons) & (no_staff_service < 3) =>
no_staff_service.
penalty_for_prev_assignments (pair_int_int no_staff_service no_afternoons) |
(no_staff_service < no_afternoons) & (no_afternoons < 3) =>
no_afternoons.
penalty_for_prev_assignments (pair_int_int no_staff_service no_afternoons) |
(((no_staff_service < 3) and (no_afternoons < 3)) == false) =>
semi_infinity.

(*
  no_prev_assignments
  calculates the number of previous assignments as a pair of the number of
  assignments to service serv and the number of assignments on afternoons
*)

no_prev_assignments :: Staff -> Service -> Day_Period -> (list Block) -> Int.
no_prev_assignments person serv period empty => (pair_int_int 0 0).
no_prev_assignments person serv AM blocks =>
  no_prev_am_sub person serv AM blocks.
no_prev_assignments person serv PM (cons (block day AM b) blocks) =>
  no_prev_am_sub person serv PM (cons (block day AM b) blocks).
no_prev_assignments person serv PM (cons (block day PM b) blocks) =>
  check_assignment_pm person serv PM (cons (block day PM b) blocks).

(*
  no_prev_am_sub
  adds one to no_staff_service, if person is assigned to service serv
*)

no_prev_am_sub ::
  Staff -> Service -> Day_Period -> (list Block) -> Pair_int_int.
no_prev_am_sub person serv period (cons b blocks) | 
  (is_staff_service_in_block person serv b) =>
    (inc_first (no_prev_assignments person serv period blocks)).
no_prev_am_sub person serv period (cons b blocks) |
check_assignment_pm ::
Staff -> Service -> Day_Period -> (list Block) -> Pair_int_int.
check_assignment_pm person serv period (cons (block day p b) blocks) |
  (is_staff_assigned_in_block person b) =>
  (no_prev_pm_sub person serv period (cons (block day p b) blocks)).
check_assignment_pm person serv period (cons (block day p b) blocks) |
  (is_staff_assigned_in_block person b) == false) =>
  (no_prev_assignments person serv period blocks).

(*
no_prev_pm_sub
increments both, if staff is assigned to service serv, otherwise
number of assignments on afternoons is incremented by one
*)

no_prev_pm_sub :::
Staff -> Service -> Day_Period -> (list Block) -> Pair_int_int.
no_prev_pm_sub person serv period (cons b blocks) |
  (is_staff_service_in_block person serv b) =>
  (inc_both (no_prev_assignments person serv period blocks)).
no_prev_pm_sub person serv period (cons b blocks) |
  ((is_staff_service_in_block person serv b) == false) =>
  (inc_second (no_prev_assignments person serv period blocks)).

(*
is_staff_preassigned
for an open service, returns the preassigned staff,
for a closed service, returns no_staff_because_closed
*)
staff_preassigned serv day period | (open serv day period) =>
    (find_preassigned_staff serv (preassignments day period)).
staff_preassigned serv day period | ((open serv day period) == false) =>
    (cons no_staff_because_closed empty).

(*
    find_preassigned_staff
    returns the staff preassigned to service serv
*)
find_preassigned_staff :: Service -> (list Pair) -> (list Staff).
find_preassigned_staff serv empty | ((staffing_policy_for_service serv) = SP_Medecin_Interne) == false =>
    (cons x empty).
find_preassigned_staff serv empty | (staffing_policy_for_service serv) = SP_Medecin_Interne =>
    (cons x (cons y empty)).
find_preassigned_staff serv (cons (pair pa_serv pa_stf) pa_list) |
    (serv=pa_serv) =>
    pa_stf.
find_preassigned_staff serv (cons (pair pa_serv pa_stf) pa_list) |
    (serv=pa_serv) == false =>
    (find_preassigned_staff serv pa_list).

(*
    is_service_preassigned
    checks, if service serv is preassigned for the specified block
*)
is_service_preassigned serv day period =>
    (is_service_in_pa_list serv (preassignments day period)).
is_service_in_pa_list :: Service -> (list Pair) -> bool.
is_service_in_pa_list serv empty => false.
is_service_in_pa_list serv (cons (pair pa_serv pa_stf) pa_list) |
    (serv=pa_serv) =>
    true.
is_service_in_pa_list serv (cons (pair pa_serv pa_stf) pa_list) |
    (serv=pa_serv) == false =>
(is_service_in_pa_list serv pa_list).

(*
   is_staff_preassigned
   checks, if person stf is preassigned for the specified block
*)

is_staff_preassigned stf day period =>
   (is_staff_in_pa_list stf (preassignments day period)).

is_staff_in_pa_list :: Staff -> (list Pair) -> bool.

is_staff_in_pa_list empty => false.
is_staff_in_pa_list (cons (pair pa_serv pa_stf) pa_list) |
   (is_in stf pa_stf) =>
      true.
is_staff_in_pa_list (cons (pair pa_serv pa_stf) pa_list) |
   (is_in stf pa_stf) == false =>
      (is_staff_in_pa_list stf pa_list).

(*
   ==========================================================================
   functions for the specifications
*)

open :: Service -> Week_Day -> Day_Period -> bool.
present :: Staff -> Week_Day -> Day_Period -> bool.
unqualified :: Staff -> Service -> bool.
staff_preassigned :: Service -> Week_Day -> Day_Period -> (list Staff).
is_service_preassigned :: Service -> Week_Day -> Day_Period -> bool.
is_staff_preassigned :: Staff -> Week_Day -> Day_Period -> bool.
same_staff_pm :: Service -> bool.
staffing_policy_for_service :: Service -> Staffing_Policy.
preassignments :: Week_Day -> Day_Period -> (list Pair).

(*
   ===================================================================
   specifications / donnees
*)

(*
   ==> open
*)

  open Examens_Urgences Monday AM => true.
  open Echo1 Monday AM => true.
  open Echo2 Monday AM => true.
  open Vasculaire_vasculaire Monday AM => false.
  open Vasculaire_os Monday AM => true.
  open Vasculaire_os__TSA Monday AM => false.
  open Vasculaire_examens Monday AM => false.
  open Mammographie Monday AM => true.
  open Scanner Monday AM => true.
  open IRM Monday AM => true.

  open Examens_Urgences Monday PM => true.
  open Echo1 Monday PM => true.
  open Echo2 Monday PM => true.
  open Vasculaire_vasculaire Monday PM => false.
  open Vasculaire_os Monday PM => false.
  open Vasculaire_os__TSA Monday PM => true.
  open Vasculaire_examens Monday PM => false.
  open Mammographie Monday PM => true.
  open Scanner Monday PM => true.
  open IRM Monday PM => true.

  open Examens_Urgences Tuesday AM => true.
  open Echo1 Tuesday AM => true.
  open Echo2 Tuesday AM => true.
  open Vasculaire_vasculaire Tuesday AM => true.
  open Vasculaire_os Tuesday AM => false.
  open Vasculaire_os__TSA Tuesday AM => false.
  open Vasculaire_examens Tuesday AM => false.
  open Mammographie Tuesday AM => true.
  open Scanner Tuesday AM => true.
<table>
<thead>
<tr>
<th>Procedure</th>
<th>Tuesday AM</th>
<th>Tuesday PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mammographie</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Scanner</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>IRM</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Examens_Urgences</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Echo1</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Echo2</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_vasculaire</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Vasculaire_os</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Vasculaire_os__TSA</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Vasculaire_examens</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_infiltrations</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Mammographie</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Scanner</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>IRM</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Examens_Urgences</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Echo1</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Echo2</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_vasculaire</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_os</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_os__TSA</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_examens</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_infiltrations</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Mammographie</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Scanner</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>IRM</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Examens_Urgences</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Echo1</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Echo2</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_vasculaire</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_os</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_os__TSA</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_examens</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_infiltrations</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Mammographie</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Scanner</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>IRM</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Examens_Urgences</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Echo1</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Echo2</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_vasculaire</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_os</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_os__TSA</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_examens</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Vasculaire_infiltrations</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Mammographie</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Scanner</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>IRM</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
open Vasculaire_os__TSA Thursday AM => false.
open Vasculaire_examens Thursday AM => false.
open Vasculaire_infiltrations Thursday AM => false.
open Mammographie Thursday AM => true.
open Scanner Thursday AM => true.
open IRM Thursday AM => true.

open Examens_Urgences Thursday PM => true.
open Echo1 Thursday PM => true.
open Echo2 Thursday PM => true.
open Vasculaire_vasculaire Thursday PM => false.
open Vasculaire_os Thursday PM => false.
open Vasculaire_os__TSA Thursday PM => false.
open Vasculaire_examens Thursday PM => false.
open Vasculaire_infiltrations Thursday PM => false.
open Mammographie Thursday PM => true.
open Scanner Thursday PM => true.
open IRM Thursday PM => true.

open Examens_Urgences Friday AM => true.
open Echo1 Friday AM => true.
open Echo2 Friday AM => true.
open Vasculaire_vasculaire Friday AM => true.
open Vasculaire_os Friday AM => false.
open Vasculaire_os__TSA Friday AM => false.
open Vasculaire_examens Friday AM => false.
open Vasculaire_infiltrations Friday AM => false.
open Mammographie Friday AM => true.
open Scanner Friday AM => true.
open IRM Friday AM => true.

open Examens_Urgences Friday PM => true.
open Echo1 Friday PM => true.
open Echo2 Friday PM => true.
open Vasculaire_vasculaire Friday PM => false.
open Vasculaire_os Friday PM => false.
open Vasculaire_os__TSA Friday PM => false.
open Vasculaire_examens Friday PM => true.
open Vasculaire_infiltrations Friday PM => false.
open Mammographie Friday PM => true.
open Scanner Friday PM => true.
open IRM Friday PM => true.
present (interne Eiss)  Monday AM => true.
present (interne Valadier)  Monday AM => true.
present (medecin Abidat)  Monday AM => true.
present (medecin Alerte)  Monday AM => true.
present (medecin Attali)  Monday AM => true.
present (medecin Brauner)  Monday AM => true.
present (medecin Dumas)  Monday AM => true.
present (medecin Elalouf)  Monday AM => true.
present (medecin El_Jarrari)  Monday AM => true.
present (medecin Le_Van_Am)  Monday AM => true.
present (medecin Mouffok)  Monday AM => true.
present (medecin One)  Monday AM => true.
present (medecin Rety)  Monday AM => true.
present (medecin Safa)  Monday AM => true.
present (medecin Touati)  Monday AM => false.
present (medecin Vignot)  Monday AM => true.
present (interne Eiss)  Monday PM => true.
present (interne Valadier)  Monday PM => true.
present (medecin Abidat)  Monday PM => true.
present (medecin Alerte)  Monday PM => true.
present (medecin Attali)  Monday PM => true.
present (medecin Brauner)  Monday PM => true.
present (medecin Dumas)  Monday PM => true.
present (medecin Elalouf)  Monday PM => true.
present (medecin El_Jarrari)  Monday PM => true.
present (medecin Hariz)  Monday PM => true.
present (medecin Jelassi)  Monday PM => true.
present (medecin Krief)  Monday PM => true.
present (medecin Le_Van_Am)  Monday PM => true.
present (medecin Mouffok)  Monday PM => true.
present (medecin One)  Monday PM => true.
present (medecin Rety)  Monday PM => true.
present (medecin Safa)  Monday PM => true.
present (medecin Touati)  Monday PM => false.
present (medecin Vignot)  Monday PM => true.
present (interne Eiss)  Tuesday AM => true.
| Present (interne Valadier) | Tuesday AM => false. |
| Present (medecin Abidat) | Tuesday AM => true. |
| Present (medecin Alerte) | Tuesday AM => true. |
| Present (medecin Attali) | Tuesday AM => true. |
| Present (medecin Brauner) | Tuesday AM => true. |
| Present (medecin Dumas) | Tuesday AM => true. |
| Present (medecin Elalouf) | Tuesday AM => true. |
| Present (medecin El_Jarrari) | Tuesday AM => true. |
| Present (medecin Hariz) | Tuesday AM => true. |
| Present (medecin Jelassi) | Tuesday AM => true. |
| Present (medecin Krief) | Tuesday AM => true. |
| Present (medecin Le_Van_Am) | Tuesday AM => true. |
| Present (medecin Mouffok) | Tuesday AM => true. |
| Present (medecin One) | Tuesday AM => true. |
| Present (medecin Rety) | Tuesday AM => true. |
| Present (medecin Touati) | Tuesday AM => false. |
| Present (medecin Safa) | Tuesday AM => true. |
| Present (medecin Vignot) | Tuesday AM => false. |

| Present (interne Eiss) | Tuesday PM => false. |
| Present (interne Valadier) | Tuesday PM => true. |
| Present (medecin Abidat) | Tuesday PM => true. |
| Present (medecin Alerte) | Tuesday PM => true. |
| Present (medecin Attali) | Tuesday PM => true. |
| Present (medecin Brauner) | Tuesday PM => true. |
| Present (medecin Dumas) | Tuesday PM => true. |
| Present (medecin Elalouf) | Tuesday PM => true. |
| Present (medecin El_Jarrari) | Tuesday PM => true. |
| Present (medecin Hariz) | Tuesday PM => true. |
| Present (medecin Jelassi) | Tuesday PM => true. |
| Present (medecin Krief) | Tuesday PM => true. |
| Present (medecin Le_Van_Am) | Tuesday PM => true. |
| Present (medecin Mouffok) | Tuesday PM => true. |
| Present (medecin One) | Tuesday PM => true. |
| Present (medecin Rety) | Tuesday PM => true. |
| Present (medecin Touati) | Tuesday PM => false. |
| Present (medecin Safa) | Tuesday PM => true. |
| Present (medecin Vignot) | Tuesday PM => false. |

| Present (interne Eiss) | Wednesday AM => true. |
| Present (interne Valadier) | Wednesday AM => true. |
| Present (medecin Abidat) | Wednesday AM => true. |
| Present (medecin Alerte) | Wednesday AM => true. |
| Present (medecin Attali) | Wednesday AM => true. |
| Present (medecin Brauner) | Wednesday AM => true. |
present (medecin Dumas) Wednesday AM => true.
present (medecin Elalouf) Wednesday AM => true.
present (medecin El_Jarrari) Wednesday AM => true.
present (medecin Hariz) Wednesday AM => true.
present (medecin Jelassi) Wednesday AM => true.
present (medecin Krief) Wednesday AM => true.
present (medecin Le_Van_Am) Wednesday AM => true.
present (medecin Mouffok) Wednesday AM => true.
present (medecin One) Wednesday AM => true.
present (medecin Rety) Wednesday AM => true.
present (medecin Safa) Wednesday AM => true.
present (medecin Touati) Wednesday AM => true.
present (medecin Vignot) Wednesday AM => true.

present (interne Eiss) Wednesday PM => true.
present (interne Valadier) Wednesday PM => false.
present (medecin Abidat) Wednesday PM => true.
present (medecin Alert) Wednesday PM => true.
present (medecin Attali) Wednesday PM => true.
present (medecin Brauner) Wednesday PM => true.
present (medecin Dumas) Wednesday PM => true.
present (medecin Elalouf) Wednesday PM => true.
present (medecin El_Jarrari) Wednesday PM => true.
present (medecin Hariz) Wednesday PM => true.
present (medecin Jelassi) Wednesday PM => true.
present (medecin Krief) Wednesday PM => true.
present (medecin Le_Van_Am) Wednesday PM => true.
present (medecin Mouffok) Wednesday PM => true.
present (medecin One) Wednesday PM => true.
present (medecin Rety) Wednesday PM => true.
present (medecin Touati) Wednesday PM => true.
present (medecin Safa) Wednesday PM => true.
present (medecin Vignot) Wednesday PM => true.

present (interne Eiss) Thursday AM => true.
present (interne Valadier) Thursday AM => true.
present (medecin Abidat) Thursday AM => true.
present (medecin Alert) Thursday AM => true.
present (medecin Attali) Thursday AM => true.
present (medecin Brauner) Thursday AM => true.
present (medecin Dumas) Thursday AM => true.
present (medecin Elalouf) Thursday AM => true.
present (medecin El_Jarrari) Thursday AM => true.
present (medecin Hariz) Thursday AM => true.
present (medecin Jelassi) Thursday AM => true.
present (medecin Krief) Thursday AM => true.
present (medecin Le_Van_Am) Thursday AM => true.
present (medecin Mouffok) Thursday AM => true.
present (medecin One) Thursday AM => true.
present (medecin Rety) Thursday AM => true.
present (medecin Safa) Thursday AM => true.
present (medecin Touati) Thursday AM => true.
present (medecin Vignot) Thursday AM => false.

present (interne Eiss) Thursday PM => false.
present (interne Valadier) Thursday PM => true.
present (medecin Abidat) Thursday PM => true.
present (medecin Alerte) Thursday PM => true.
present (medecin Attali) Thursday PM => true.
present (medecin Brauner) Thursday PM => true.
present (medecin Dumas) Thursday PM => true.
present (medecin Elalouf) Thursday PM => true.
present (medecin El_Jarrari) Thursday PM => true.
present (medecin Hariz) Thursday PM => true.
present (medecin Jelassi) Thursday PM => true.
present (medecin Krief) Thursday PM => true.
present (medecin Le_Van_Am) Thursday PM => true.
present (medecin Mouffok) Thursday PM => true.
present (medecin One) Thursday PM => true.
present (medecin Rety) Thursday PM => true.
present (medecin Touati) Thursday PM => true.
present (medecin Safa) Thursday PM => true.
present (medecin Vignot) Thursday PM => false.

present (interne Eiss) Friday AM => true.
present (interne Valadier) Friday AM => true.
present (medecin Abidat) Friday AM => true.
present (medecin Alerte) Friday AM => true.
present (medecin Attali) Friday AM => true.
present (medecin Brauner) Friday AM => true.
present (medecin Dumas) Friday AM => true.
present (medecin Elalouf) Friday AM => true.
present (medecin El_Jarrari) Friday AM => true.
present (medecin Hariz) Friday AM => true.
present (medecin Jelassi) Friday AM => true.
present (medecin Krief) Friday AM => true.
present (medecin Le_Van_Am) Friday AM => true.
present (medecin Mouffok) Friday AM => true.
present (medecin One) Friday AM => true.
present (medecin Rety) Friday AM => true.
present (medecin Touati) Friday AM => true.
present (medecin Safa) Friday AM => true.
present (medecin Vignot) Friday AM => false.
present (interne Eiss) Friday PM => true.
present (interne Valadier) Friday PM => false.
present (medecin Abidat) Friday PM => true.
present (medecin Alerte) Friday PM => true.
present (medecin Attali) Friday PM => true.
present (medecin Brauner) Friday PM => true.
present (medecin Dumas) Friday PM => true.
present (medecin Elalouf) Friday PM => true.
present (medecin El_Jarrari) Friday PM => true.
present (medecin Hariz) Friday PM => true.
present (medecin Jelassi) Friday PM => true.
present (medecin Krief) Friday PM => true.
present (medecin Le_Van_Am) Friday PM => true.
present (medecin Mouffok) Friday PM => true.
present (medecin One) Friday PM => true.
present (medecin Rety) Friday PM => true.
present (medecin Touati) Friday PM => true.
present (medecin Safa) Friday PM => true.
present (medecin Vignot) Friday PM => false.

(*
  => unqualified
*)

unqualified (medecin Abidat) Examens_Urgences => false.
unqualified (medecin Alerte) Examens_Urgences => false.
unqualified (medecin Attali) Examens_Urgences => false.
unqualified (medecin Brauner) Examens_Urgences => false.
unqualified (medecin Dumas) Examens_Urgences => true.
unqualified (medecin Elalouf) Examens_Urgences => false.
unqualified (medecin El_Jarrari) Examens_Urgences => false.
unqualified (medecin Hariz) Examens_Urgences => false.
unqualified (medecin Jelassi) Examens_Urgences => false.
unqualified (medecin Krief) Examens_Urgences => false.
unqualified (medecin Le_Van_Am) Examens_Urgences => false.
unqualified (medecin Mouffok) Examens_Urgences => false.
unqualified (medecin One) Examens_Urgences => false.
unqualified (medecin Rety) Examens_Urgences => false.
unqualified (medecin Safa) Examens_Urgences => false.
<table>
<thead>
<tr>
<th>Medical Doctor</th>
<th>Examens_Urgences</th>
<th>Echo1</th>
<th>Echo2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Touati</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vignot</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eiss</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Valadier</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abidat</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alerte</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attali</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brauner</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dumas</td>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elalouf</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>El_Jarrari</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hariz</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jelassi</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Krief</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Le_Van_Am</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mouffok</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rety</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safa</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Touati</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vignot</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eiss</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Valadier</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abidat</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alerte</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attali</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brauner</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dumas</td>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elalouf</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>El_Jarrari</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hariz</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jelassi</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Krief</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Le_Van_Am</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mouffok</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rety</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safa</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Touati</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vignot</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eiss</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Valadier</td>
<td>False</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
unqualified (medecin Abidat) Vasculaire_vasculaire => true.
unqualified (medecin Alerte) Vasculaire_vasculaire => false.
unqualified (medecin Attali) Vasculaire_vasculaire => false.
unqualified (medecin Brauner) Vasculaire_vasculaire => false.
unqualified (medecin Dumas) Vasculaire_vasculaire => false.
unqualified (medecin Elalouf) Vasculaire_vasculaire => true.
unqualified (medecin El_Jarrari) Vasculaire_vasculaire => true.
unqualified (medecin Hariz) Vasculaire_vasculaire => true.
unqualified (medecin Jelassi) Vasculaire_vasculaire => true.
unqualified (medecin Krief) Vasculaire_vasculaire => false.
unqualified (medecin Le_Van_Am) Vasculaire_vasculaire => true.
unqualified (medecin Mouffok) Vasculaire_vasculaire => false.
unqualified (medecin One) Vasculaire_vasculaire => true.
unqualified (medecin Rety) Vasculaire_vasculaire => true.
unqualified (medecin Safa) Vasculaire_vasculaire => false.
unqualified (medecin Touati) Vasculaire_vasculaire => false.
unqualified (medecin Vignot) Vasculaire_vasculaire => true.
unqualified (interne Eiss) Vasculaire_vasculaire => true.
unqualified (interne Valadier) Vasculaire_vasculaire => true.

unqualified (medecin Abidat) Vasculaire_os => true.
unqualified (medecin Alerte) Vasculaire_os => false.
unqualified (medecin Attali) Vasculaire_os => false.
unqualified (medecin Brauner) Vasculaire_os => false.
unqualified (medecin Dumas) Vasculaire_os => false.
unqualified (medecin Elalouf) Vasculaire_os => true.
unqualified (medecin El_Jarrari) Vasculaire_os => true.
unqualified (medecin Hariz) Vasculaire_os => true.
unqualified (medecin Jelassi) Vasculaire_os => true.
unqualified (medecin Krief) Vasculaire_os => false.
unqualified (medecin Le_Van_Am) Vasculaire_os => true.
unqualified (medecin Mouffok) Vasculaire_os => false.
unqualified (medecin One) Vasculaire_os => true.
unqualified (medecin Rety) Vasculaire_os => true.
unqualified (medecin Safa) Vasculaire_os => false.
unqualified (medecin Touati) Vasculaire_os => false.
unqualified (interne Eiss) Vasculaire_os => false.
unqualified (interne Valadier) Vasculaire_os => true.

unqualified (medecin Abidat) Vasculaire_os__TSA => true.
unqualified (medecin Alerte) Vasculaire_os__TSA => false.
unqualified (medecin Attali) Vasculaire_os__TSA => false.
unqualified (medecin Brauner) Vasculaire_os__TSA => false.
unqualified (medecin Dumas) Vasculaire_os__TSA => false.
unqualified (medecin Elalouf) Vasculaire_os__TSA => true.
unqualified (medecin El_Jarrari) Vasculaire_os__TSA => true.
unqualified (medecin Hariz) Vasculaire_os__TSA => true.
unqualified (medecin Jelassi) Vasculaire_os__TSA => true.
unqualified (medecin Krief) Vasculaire_os__TSA => false.
unqualified (medecin Le_Van_Am) Vasculaire_os__TSA => true.
unqualified (medecin Mouffok) Vasculaire_os__TSA => false.
unqualified (medecin One) Vasculaire_os__TSA => true.
unqualified (medecin Rety) Vasculaire_os__TSA => true.
unqualified (medecin Safa) Vasculaire_os__TSA => false.
unqualified (medecin Touati) Vasculaire_os__TSA => false.
unqualified (medecin Vignot) Vasculaire_os__TSA => true.
unqualified (interne Eiss) Vasculaire_os__TSA => false.
unqualified (interne Valadier) Vasculaire_os__TSA => false.
unqualified (medecin Elalouf)    Vasculaire_os__TSA => true.
unqualified (medecin El_Jarrari) Vasculaire_os__TSA => true.
unqualified (medecin Hariz)      Vasculaire_os__TSA => true.
unqualified (medecin Jelassi)    Vasculaire_os__TSA => true.
unqualified (medecin Krief)      Vasculaire_os__TSA => false.
unqualified (medecin Le_Van_Am)  Vasculaire_os__TSA => true.
unqualified (medecin Mouffok)    Vasculaire_os__TSA => false.
unqualified (medecin One)        Vasculaire_os__TSA => true.
unqualified (medecin Rety)       Vasculaire_os__TSA => true.
unqualified (medecin Safa)       Vasculaire_os__TSA => false.
unqualified (interne Eiss)       Vasculaire_os__TSA => false.
unqualified (interne Valadier)   Vasculaire_os__TSA => false.

unqualified (medecin Abidat)     Vasculaire_examens => true.
unqualified (medecin Alerte)     Vasculaire_examens => false.
unqualified (medecin Attali)     Vasculaire_examens => false.
unqualified (medecin Brauner)    Vasculaire_examens => false.
unqualified (medecin Dumas)      Vasculaire_examens => false.
unqualified (medecin Elalouf)    Vasculaire_examens => true.
unqualified (medecin El_Jarrari) Vasculaire_examens => true.
unqualified (medecin Hariz)      Vasculaire_examens => true.
unqualified (medecin Jelassi)    Vasculaire_examens => true.
unqualified (medecin Krief)      Vasculaire_examens => false.
unqualified (medecin Le_Van_Am)  Vasculaire_examens => true.
unqualified (medecin Mouffok)    Vasculaire_examens => false.
unqualified (medecin One)        Vasculaire_examens => true.
unqualified (medecin Rety)       Vasculaire_examens => true.
unqualified (medecin Safa)       Vasculaire_examens => false.
unqualified (interne Eiss)       Vasculaire_examens => false.
unqualified (interne Valadier)   Vasculaire_examens => false.

unqualified (medecin Abidat)     Vasculaire_infiltrations => true.
unqualified (medecin Alerte)     Vasculaire_infiltrations => false.
unqualified (medecin Attali)     Vasculaire_infiltrations => false.
unqualified (medecin Brauner)    Vasculaire_infiltrations => false.
unqualified (medecin Dumas)      Vasculaire_infiltrations => false.
unqualified (medecin Elalouf)    Vasculaire_infiltrations => true.
unqualified (medecin El_Jarrari) Vasculaire_infiltrations => true.
unqualified (medecin Hariz)      Vasculaire_infiltrations => true.
unqualified (medecin Jelassi)    Vasculaire_infiltrations => true.
unqualified (medecin Krief)      Vasculaire_infiltrations => false.
unqualified (medecin Le_Van_Am) Vasculaire_infiltrations => true.
unqualified (medecin Mouffok) Vasculaire_infiltrations => false.
unqualified (medecin One) Vasculaire_infiltrations => true.
unqualified (medecin Rety) Vasculaire_infiltrations => true.
unqualified (medecin Safa) Vasculaire_infiltrations => false.
unqualified (medecin Touati) Vasculaire_infiltrations => false.
unqualified (medecin Vignot) Vasculaire_infiltrations => true.
unqualified (interne Eiss) Vasculaire_infiltrations => false.
unqualified (interne Valadier) Vasculaire_infiltrations => false.

unqualified (medecin Alerte) Mammographie => false.
unqualified (medecin Abidat) Mammographie => false.
unqualified (medecin Attali) Mammographie => false.
unqualified (medecin Brauner) Mammographie => true.
unqualified (medecin Dumas) Mammographie => false.
unqualified (medecin Elalouf) Mammographie => false.
unqualified (medecin El_Jarrari) Mammographie => false.
unqualified (medecin Hariz) Mammographie => false.
unqualified (medecin Jelassi) Mammographie => true.
unqualified (medecin Krief) Mammographie => false.
unqualified (medecin Le_Van_Am) Mammographie => false.
unqualified (medecin Mouffok) Mammographie => false.
unqualified (medecin One) Mammographie => false.
unqualified (medecin Rety) Mammographie => false.
unqualified (medecin Safa) Mammographie => false.
unqualified (medecin Touati) Mammographie => false.
unqualified (medecin Vignot) Mammographie => false.
unqualified (interne Eiss) Mammographie => false.
unqualified (interne Valadier) Mammographie => false.

unqualified (medecin Abidat) Scanner => false.
unqualified (medecin Alerte) Scanner => false.
unqualified (medecin Attali) Scanner => false.
unqualified (medecin Brauner) Scanner => false.
unqualified (medecin Dumas) Scanner => false.
unqualified (medecin Elalouf) Scanner => false.
unqualified (medecin El_Jarrari) Scanner => false.
unqualified (medecin Hariz) Scanner => false.
unqualified (medecin Jelassi) Scanner => false.
unqualified (medecin Krief) Scanner => false.
unqualified (medecin Le_Van_Am) Scanner => false.
unqualified (medecin Mouffok) Scanner => false.
unqualified (medecin One) Scanner => false.
unqualified (medecin Rety) Scanner => false.
unqualified (medecin Safa) Scanner => false.
unqualified (medecin Touati)  Scanner => false.
unqualified (medecin Vignot) Scanner => false.
unqualified (interne Eiss) Scanner => false.
unqualified (interne Valadier) Scanner => false.
unqualified (medecin Alerte)  IRM => false.
unqualified (medecin Abidat)  IRM => false.
unqualified (medecin Attali)   IRM => false.
unqualified (medecin Brauner) IRM => true.
unqualified (medecin Dumas)   IRM => false.
unqualified (medecin Elalouf) IRM => false.
unqualified (medecin El_Jarrari) IRM => false.
unqualified (medecin Hariz)   IRM => false.
unqualified (medecin Jelassi) IRM => false.
unqualified (medecin Krief)   IRM => false.
unqualified (medecin Le_Van_Am) IRM => false.
unqualified (medecin Mouffok) IRM => true.
unqualified (medecin One)     IRM => false.
unqualified (medecin Rety)    IRM => false.
unqualified (medecin Safa)    IRM => false.
unqualified (medecin Touati)  IRM => true.
unqualified (medecin Vignot)  IRM => false.
unqualified (interne Eiss)    IRM => false.
unqualified (interne Valadier) IRM => false.

(* => service staffing policies *)

staffing_policy_for_service Examens_Urgences => SP_Medecin.
staffing_policy_for_service Echo1 => SP_Medecin.
staffing_policy_for_service Echo2 => SP_Medecin.
staffing_policy_for_service Vasculaire_vasculaire => SP_Medecin.
staffing_policy_for_service Vasculaire_os => SP_Medecin.
staffing_policy_for_service Vasculaire_os__TSA => SP_Medecin.
staffing_policy_for_service Vasculaire_examens => SP_Medecin.
staffing_policy_for_service Vasculaire_infiltrations => SP_Medecin.
staffing_policy_for_service Mammographie => SP_Medecin.
staffing_policy_for_service Scanner => SP_Medecin, Interne.
staffing_policy_for_service IRM => SP_Medecin.
same_staff_pm Examens_Urgences => true.
same_staff_pm Echo1 => false.
same_staff_pm Echo2 => false.
same_staff_pm Vasculaire_vasculaire => false.
same_staff_pm Vasculaire_os => false.
same_staff_pm Vasculaire_os__TSA => false.
same_staff_pm Vasculaire_examens => false.
same_staff_pm Vasculaire_infiltrations => false.
same_staff_pm Mammographie => false.
same_staff_pm Scanner => false.
same_staff_pm IRM => false.

preassignments Monday AM =>
  (cons (pair IRM (cons (medecin Vignot) empty))
  empty).
preams Monday PM => empty.
preams Tuesday AM => empty.
preams Tuesday PM => empty.
preams Wednesday AM =>
  (cons (pair Scanner (cons (interne Eiss) empty))
  empty).
preams Wednesday PM => empty.
preams Thursday AM =>
  (cons (pair Mammographie (cons (medecin Touati) empty))
  (cons (pair Scanner (cons (interne Valadier) empty))
  empty)).
preams Thursday PM => empty.
preams Friday AM => empty.
preams Friday PM =>
  (cons (pair Scanner (cons (medecin Touati) empty))
  empty).
end hospital
Les Cahiers du laboratoire Leibniz

Le Laboratoire Leibniz est fortement pluridisciplinaire. Son activité scientifique couvre un large domaine qui comprend des thèmes fondamentaux aussi bien en informatique qu'en mathématiques, avec une ouverture sur l'apprentissage machine, la modélisation de systèmes complexes adaptatifs, et les applications aux environnements informatiques pour l'apprentissage humain.

Les Cahiers du laboratoire Leibniz ont pour vocation la diffusion de rapports de recherche, de supports de cours, de textes de séminaires ou de projets de publications réalisés par des membres du laboratoire. Ils peuvent accueillir aussi des textes de chercheurs n'appartenant pas au laboratoire Leibniz mais qui travaillent sur des thèmes proches et ne disposent pas de tels supports de publication. Ces chercheurs sont priés de contacter un des membres du comité éditorial ; le comité décidera de l'acceptation du texte proposé.

Le contenu des textes publiés dans les Cahiers du laboratoire Leibniz relève de la seule responsabilité de leurs auteurs.

The research at Laboratoire Leibniz is multidisciplinary. It covers a large domain of fundamental and applied subjects in informatics an mathematics, with openings to machine learning, the modelisation of adaptive complex systems, and applications to teaching software.

The Cahiers du laboratoire Leibniz aim at diffusing research reports, lectures, and texts of conferences or pre-prints of the members of the laboratory. Moreover, the Cahiers welcome manuscripts of researchers belonging to other laboratories, working on subjects close to ours, but not disposing of such a medium. These researchers should contact one of the members of the editorial board; the latter will decide whether to accept the proposal.

The responsibility of the contents of the Cahiers lies exclusively with the authors.

Comité éditorial
Mirta B. Gordon (responsable), Annie Bessot, Gerd Finke, Humbert Fiorino, Denise Grenier, Philippe Jorrand, Andras Sëbo

Directeur de la publication
Nicolas Balacheff

Réalisation : Jacky Coutin

ISSN : 1298-020X - © laboratoire Leibniz