Algorithms for the substrate of MGR

Roger T. Hartley

March 18, 2003

1 The MGR architecture

The architecture of MGR consists of the two operators *Specialize* and *Fragment* that manipulate three databases, *Facts, Definitions and Models*, that are three partitions of the overall database of *Graphs*. The relationship among these is illustrated in Figure 1. Functionally the operators are:

$$Sp : 2^F \times 2^D \rightarrow 2^M$$
$$Fr : M \times 2^F \rightarrow 2^M$$

where

$$F + D + M = G$$

Each operator is composed of more primitive operations which are the subject of this paper. These operations comprise a substrate for MGR on which all of the high level operators are founded.

Specialize is composed of the operations *Choose, Cover and Join*. Fragment is composed of the operations *Choose, Project and Uncover*. Each of these[1] will be discussed in turn, and an algorithm presented in 'set pseudocode'. Since all of the algorithms are manipulations on sets and functions, the basic set operations of union and intersection are all that are needed, along with looping and conditional forms, and assignment for control. The pseudocode should therefore be self-explanatory.

The functionality of the operations is:

$$\begin{aligned} Ch &: G \rightarrow 2^G \\ Cv &: 2^F \times 2^M \times 2^D \rightarrow 2^D \\ Jn &: 2^G \rightarrow 2^G \\ Pr &: 2^G \rightarrow 2^G \\ Uc &: M \times 2^F \rightarrow 2^M \end{aligned}$$

and their composition into the higher operators is

$$Sp = Jn \circ Cv \circ Ch$$
$$Fr = Uc \circ Pr \circ Ch$$

2 How graphs are represented

Let a *box* be a *node-label* pair: $\langle n, l \rangle$ where $l \in L$ and n is a natural number, used as a unique identifier. We will call these identifiers *nodes*. Boxes are elements of a mapping $NodeLabel : N \rightarrow L$ where every n has a unique l, but every l may have multiple n's.

[1] The operation Choose will not be presented here, since it is the operation most affected by problem-specific considerations, and is in any case a stochastic operation with no real algorithm

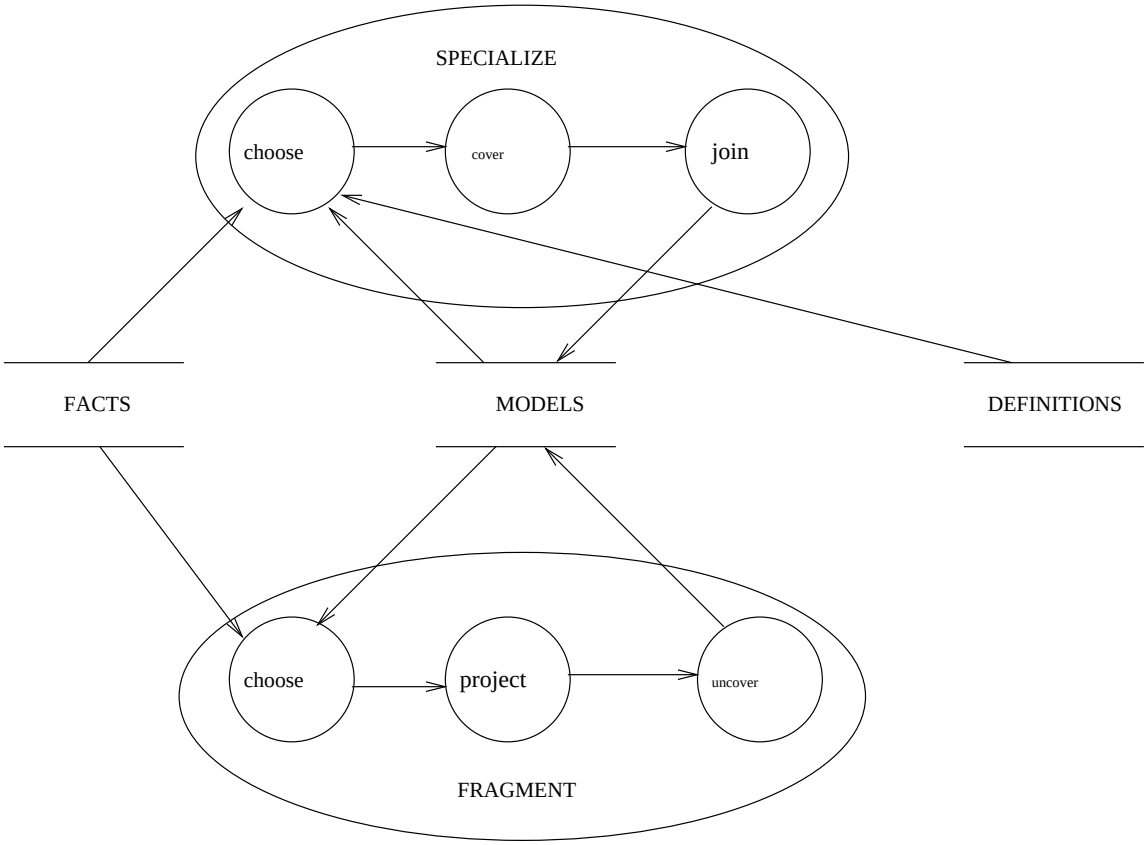


Figure 1: The basic MGR architecture

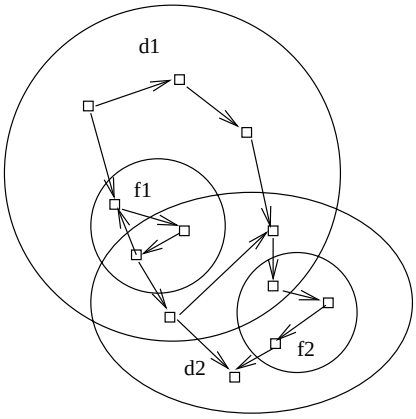


Figure 2: Facts $f1$ and $f2$ covered by definitions $d1$ and $d2$

The links are then a mapping between nodes, together with a flag that indicates the direction of the connection between the nodes: $Conn : N \rightarrow N \times \{f, b\}$ (f means a forward link and b a back link).

A graph is then a triple

$$\langle N, B, Conn \rangle$$

3 Cover and Uncover

The first step of the operator *Specialize* is to look for a conceptual closure of the concepts in the selected facts that it takes as input. It does this by covering each concept in the set of fact graphs by at least one definition (see Figure 3.1). The second step of the operator *Fragment* is to break a model apart by removing the minimal number of concept nodes and/or relation nodes in order to disconnect the graph, preserving one or more fact projections into the model as wholes (see Figure 3.2).

3.1 Cover

Let $F' \subseteq F + M$ be the set of chosen facts[2] and D be the set of definitions. F, M and D are distinguished members of the set of graphs G, i.e. $G = F + D + M$. Let $Label(g)$ be the set of concept labels in any graph $g \in G$. These terms come from a set L of concept labels. A cover of F' is obtained if:

$$T(\cup F') \subseteq \cup D' \subseteq D$$

and a cover is *minimal* if D' is the smallest such subset of D.

3.1.1 The algorithm for cover

The representation system (CP) maintains a hash table of concepts and the definitions that contain them. Call this mapping $Contained : L \rightarrow D$. It is easy then to form a set of alternative covers for each label in F' in the following way:

MakeAlternatives

1 $C = \emptyset$

[2]for the purposes of cover, we can choose F or M as the source of graphs

2 **for each** $f_i \in F'$
3 **for each** $l_j \in Label(f_i)$
4 $C = C \cup \{Contained(l_j)\ \}$

The set C can be seen as a conjunct of disjuncts. Where each element of C contains at least one necessary definition. Viewed as a Boolean expression it is

$$\bigwedge_i \bigvee_j d_{ij}$$

Minimal cover is then a process of minimizing this expression so that there are the smallest number of alternatives and each alternative is as small as possible.

The minimization algorithm is:

MinimizeAlternatives

1 $A^{cover} = \{\emptyset\}$
2 **for each** $c_i \in C$ [the set of alternatives from above]
3 $N = \emptyset$
4 **for each** $d_j \in c_i$
5 $T = \emptyset$
6 **for each** $a_k \in A^{cover}$
7 $T = T \cup \{a_k \cup \{d_j\}\}$
8 $N = N \cup T$, where $\neg\exists\ n_p, n_q \in N : n_p \supset n_q$
[hence N does not contain a set which is a superset of any other]
9 $A^{cover} = N$

3.2 Uncover

Assume a graph M and a set of projections into it P. M is the triple:

$$\langle N^M, B^M, Conn^M \rangle$$

and the $P_i \in P$ are:

$$\langle N_i^P, B_i^P, Conn_i^P, NodeMap_i^P \rangle$$

The $NodeMap_i^P$ are mappings from nodes in the graph M to the nodes in the projection graph, as returned by the operation project *q.v.*. There is one each for the original graphs that make the projection. These original graphs are not needed for uncover since the idea is to 'grow' each projection until they almost, but not quite, touch.

3.2.1 The algorithm for uncover

The strategy followed is to search breadth-first from each projection, in parallel, looking for any node intersection between the growing projections. The search has to be carried out, however, in the graph M, so the first step is to use the $NodeMap_i^P$ to find these nodes:

1 **for each** $P_i \in P$
2 $FN_i = \{n_j : NewNode_i^P(n_j) \in N^M\}$

The sets FN_i now contain the nodes in M that map to each node in the projections P_i. The multiple parallel breadth-first search is then:

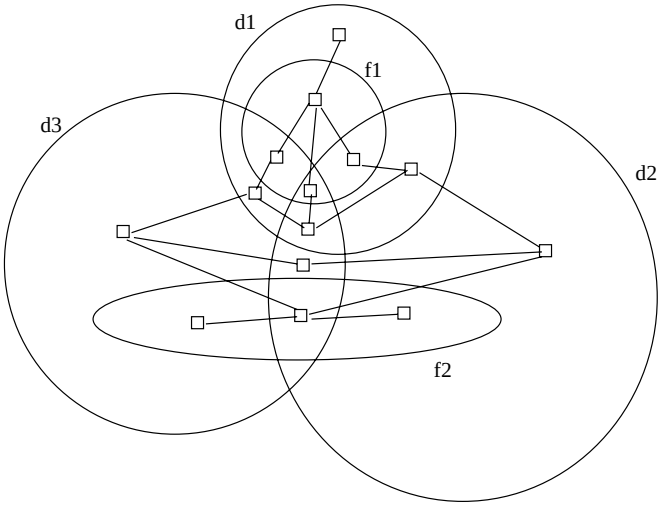


Figure 3: A model formed from $d1$, $d2$ and $d3$ uncovered with $f1$ and $f2$. The shaded area is removed.

Uncover

```
3  repeat
4      for each P_i
5          FN'_i = {n_j : Conn^M(n_j) ↑ 1 ∈ Fn_i}
6          FN_i = FN_i ∪ FN'_i
7
8      for each P_i
9          for each P_j, i ≠ j
10             Int = FN_i ∩ FN_j
11             when Int ≠ ∅
12                 for each n_k ∈ Int
13                     for each n_l ∈ Int, k ≠ l
14                         if ⟨n_k, n_l, f⟩ ∈ Conn^M ∨ ⟨n_k, n_l, b⟩ ∈ Conn^M
15                             if n_k ∈ FN_i
16                                 FN_i = FN_i − n_l
17                                 FN_j = FN_j − n_k
18                             else
19                                 FN_i = FN_i − n_k
20                                 FN_j = FN_j − n_l
21                             Linked = True
22                         else
23                             Linked = False
24                     when Linked = False
25                         FN_i = FN_i − n_k
26                         FN_j = FN_j − n_k
27 until ∀i FN'_i = ∅
```

Lines 4-6 augment the set of nodes for the fragment by following the links in $Conn^M$ (since both forward and backward links are represented explicitly, this is no problem). Lines 8-26 check for any intersection among the node sets. There are two possibilities: either the intersection set contains links nodes that are

linked to others in the set (the searches have crossed over) or the set contains nodes that are not linked to any others. In the first case, the 'cross-over' nodes are removed from the 'other' set. In the second, the node is removed from both. This process will ensure that the resultant fragments are fully disconnected (i.e. are not joinable).

Having determined the node sets for the fragments, it remains to create graphs from them. This is done by:

24 **for each** P_i
25 $\quad NodeMap_i = \emptyset$
26 $\quad n_{new} = |N_i^P| + 1$
27 $\quad$ **for each** $\langle n_1, n_2, c\rangle_j \in Conn_i^P$
28 $\quad\quad$ **if** $n_1 \in FN_i \wedge n_2 \in FN_i$
29 $\quad\quad\quad l_1 = NodeLabel^M(n_1)$
30 $\quad\quad\quad l_2 = NodeLabel^M(n_2)$
31 $\quad\quad\quad$ **if** $NewNode(n_1) \neq \epsilon$
32 $\quad\quad\quad\quad n_1 = NewNode(n_1)$
33 $\quad\quad\quad$ **else**
34 $\quad\quad\quad\quad n_1 = n_{new}$
35 $\quad\quad\quad\quad$ increment n_{new}
36 $\quad\quad\quad$ **if** $NewNode(n_2) \neq \epsilon$
37 $\quad\quad\quad\quad n_2 = NewNode(n_2)$
38 $\quad\quad\quad$ **else**
39 $\quad\quad\quad\quad n_2 = n_{new}$
40 $\quad\quad\quad\quad$ increment n_{new}
41 $\quad\quad\quad Conn_i^P = Conn_i^P \cup \{\langle n_1, n_2, c\rangle\}$
42 $\quad\quad\quad N_i^P = N_i^P \cup \{n_1, n_2\}$
43 $\quad\quad\quad B_i^P = B_i^P \cup \{\langle n_1, l_1\rangle, \langle n_2, l_2\rangle\}$
44 $\quad Fr_i = \langle Conn_i^P, N_i^P, B_i^P\rangle$

The result is a set of graphs Fr_i, each one of which contiains the corresponding projection P_i as a subgraph. The fragments are disconnected one from another, but each is a projection from the original graph M.

4 Maximal join and project

Given two conceptual graphs [3], produce a graphs representing their maximal join and project. The graphs contain labels taken from a type lattice on the relation subtype. The graphs may have duplicate labels (e.g. more than one occurrence of any given label), but the maximal common subtype of any two labels must be unique.

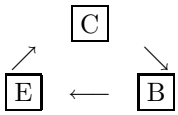
We will use the same example to illustrate the algorithms for both join and project. They use a type hierarchy where B and $C < A$, $E < B$ and C [4]. The two graphs are

$\boxed{A_1} \rightarrow \boxed{\text{C}} \rightarrow \boxed{A_2}$ and $\boxed{\text{B}} \rightarrow \boxed{\text{E}}$ (subscripts indicate duplicate labels)

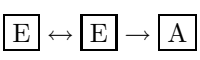
4.1 Maximal join

From the above graphs produce the following alternative maximal joins:

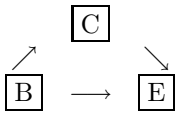
[3] in this paper we use the reduced form of conceptual graph where relation nodes are deleted
[4] < means 'subtype'

1. 

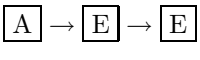
   joining A_1 to E and A_2 to B.

2. 

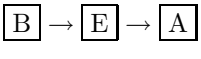
   joining A_1 to E and C to B.

3. 

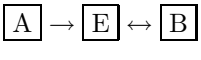
   joining A_2 to E and A_1 to B.

4. 

   joining A_2 to E and C to B.

5. 

   joining C to E and A_1 to B.

6. 

   joining C to E and A_2 to B.

4.2 The algorithm for maximal join

The algorithm is in two parts:

1. Produce the set of alternative pairings of nodes from the two graphs.
2. Merge the graphs on paired nodes, producing one resultant graph.

4.2.1 Pair alternatives

Let the two graphs to be joined contain sets of nodes N_1 and N_2, and the joined graph contain a set N_3. Let the connection mappings be $Conn_1$, $Conn_2$ and $Conn_3$ respectively.

Let labels be taken from the set of lowercase letters *viz* a,b,c etc. [5] Let the labels in the first graph be a set U and the labels in the second be a set V. The possible joins of two graphs are then governed by the number of ways in which individual node pairings can be combined, subject to allowable pairs of labels.

A node can be paired with another if the maximal common subtype of their labels (the function $M_b : L \times L \rightarrow L$, where L is the domain of labels.) is not $\perp$, e.g.

$$M_b(a, b) = b,\ M_b(b, c) = e\ etc.$$

An alternative pairing a_i taken from a set A^{pair} of all such pair combinations is the largest set of possible individual node pairings. a_i is a set $\{\ldots p_{ij} \ldots\}$ where

$$p_{ij} \quad = \quad \langle n_i, n_j \rangle : n_i \in N_1, n_j \in N_2,$$

[5] these correspond to the uppercase 'nodes' in the example, i.e. A, B, C etc.

$$\begin{aligned} u_i &= NodeLabel(Conn_1(n_i) \uparrow 1) \in U, \\ v_j &= NodeLabel(Conn_2(n_j) \uparrow 1) \in V, \\ M_b(u_i, v_j) &\neq \bot \end{aligned}$$

Alternative joins of the two graphs are then obtained by forming the Cartesian product of the a_i, but disallowing combinations where nodes are mentioned in more than one pair. Depending on these constraints, there may be alternatives of differing cardinality, i.e. when a particular node pairing stops one of its nodes being paired with nodes from the other graph.

Let this set be A^{join}. It elements are $a_i^{join} \in a_1 \times a_2 \times \ldots$ where

$$a_i^{join} = \{\langle n_i, n_j \rangle : \neg\exists kl \ (n_k = n_l \vee n_k = n_l, k \neq l)\}$$

The algorithm is:

PairNode

```
1  A^pair = ∅
2  for each n_i ∈ N_1
3       a_i = ∅
4       for each n_j ∈ N_2
5            if M_b(u_i, v_j) ≠ ⊥,
6                 [where u_i = NodeLabel(Conn_1(n_i) ↑ 1), and
7                  v_j = NodeLabel(Conn_2(n_j) ↑ 1)]
8                 a_i = a_i ∪ {⟨n_i, n_j⟩}
9       A^pair = A^pair ∪ a_i
10
11 A^join = {∅}
12 for each a_i ∈ A^pair
13      C = copy A^join
14      A^join = ∅
15      for each p_j ∈ a_i
16           for each c_k ∈ C
17                AddPair = TRUE
18                c_temp = ∅
19                for each p_l ∈ c_k
20                     if (p_j ↑ 1 = p_l ↑ 1) ∨ (p_j ↑= p_l ↑ 2)
21                          AddPair = FALSE
22                     else
23                          c_temp = c_temp ∪ {p_l}
24                if AddPair = TRUE
25                     A^join = A^join ∪ {c_k ∪ {p_j}}
26                else
27                     when c_temp ≠ ∅
28                          A^join = A^join ∪ {c_temp} ∪ {c_k}
```

Lines 2-9 find all pairs of nodes from N_1 and N_2 whose labels have a maximal common subtype that is not $\bot$. The result is s set of pairs A^{pair}. The next part of the algorithm produces the all possible combinations of these node pairs. The first pass, when C is empty produces a set of singleton sets each consisting of a pair in a_1. The subsequent passes add pairs from $a_i, i > 1$ to each of these sets unless either node in a pair is already present (i.e. it is already paired with another, and cannot be joined to anything else). The result is a set A^{join} of all the possible node pair combinations. If a pair has a node already present, a residue set

(c_{temp}) contains the rest of the pairs with which it can co-exist. If a pair can be added, it is (line 25). If it cannot, the original (c_k), and the residue (c_{temp}) are added to A^{join} for the next pass. This ensures that combinations not including the current pair are kept alive for further additions. The sets are also kept at maximal size, eventually producing maximal joins. In our example we have:

$$\begin{aligned} B_1 &= \{\langle 1,a\rangle, \langle 2,a\rangle, \langle 3,c\rangle\} \\ B_2 &= \{\langle 1,b\rangle, \langle 2,e\rangle\} \\ N_1 &= \{1,2,3\} \\ N_2 &= \{1,2\} \\ Conn_1 &= \{\langle 1,3,f\rangle, \langle 3,2,f\rangle, \langle 3,1,b\rangle, \langle 2,3,b\rangle\} \\ Conn_2 &= \{\langle 1,2,f\rangle, \langle 2,1,b\rangle\} \\ M_b &= \{\langle a,b,b\rangle, \langle a,e,e\rangle, \langle c,b,e\rangle, \langle c,e,e\rangle, \ldots\} \end{aligned}$$

The set A^{pair} is then

$$\{\{\langle 1,1\rangle, \langle 1,2\rangle\}, \{\langle 2,1\rangle, \langle 2,2\rangle\}, \{\langle 3,1\rangle, \langle 3,2\rangle\}\}$$

since all pairings are possible.

A^{join} starts out containing the empty set (line 11). After the first pass (a_1), A^{join} is:

$$\{\{\langle 1,1\rangle\}, \{\langle 1,2\rangle\}\}$$

After the second (a_2), it is

$$\{\{\langle 1,2\rangle, \langle 2,1\rangle\}, \{\langle 1,1\rangle, \langle 2,2\rangle\}\}$$

Finally, after the third (a_3), A^{join} is

$$\{\{\langle 3,1\rangle, \langle 1,2\rangle\}, \{\langle 1,2\rangle, \langle 2,1\rangle\}, \{\langle 3,1\rangle, \langle 2,2\rangle\}, \{\langle 2,2\rangle, \langle 1,1\rangle\}, \{\langle 3,2\rangle, \langle 2,1\rangle\}, \{\langle 3,2\rangle, \langle 1,1\rangle\}\}$$

All of these alternatives represent maximal joins.

4.2.2 Merge graphs

Having produced the set A^{join} of all possible label pairings, the remaining task is to make a graph for each pairing set from the two input graphs. This amounts to producing a new mapping in which the original node pairings are replaced by ones reflecting the merged nodes. Along the way, two sets of old-node to new-node maps are made, one for each input graph. They are called $NodeMap_1$ and $NodeMap_2$. They are both subsets of a function $NewNode : N \rightarrow N \times L + \epsilon$. NewNode returns ϵ if a node has no mapping. The algorithm is:

MergeNodes

1 $AltJoins = B_3 = N_3 = Temp = NodeMap_1 = NodeMap_2 = \emptyset$
2 **for each** $a_i \in A^{join}$
3 $\quad n_{new} = 1$
4 $\quad$ **for each** $p_j \in a_i$
5 $\quad\quad Temp = \langle n_{new}, l\rangle$, [where $l = M_b(NodeLabel_1(p_j \uparrow 1), NodeLabel_2(p_j \uparrow 2))$]
6 $\quad\quad NodeMap_1 = NodeMap_1 \cup \{\langle p_j \uparrow 1, Temp\rangle\}$
7 $\quad\quad NodeMap_2 = NodeMap_2 \cup \{\langle p_j \uparrow 2, Temp\rangle\}$
8 $\quad\quad B_3 = B_3 \cup \{Temp\}$
9 $\quad\quad N_3 = N_3 \cup \{n_{new}\}$
10 $\quad\quad$ increment n_{new}

11
12 $Conn_3 = \emptyset$
13 **for each** $c_i \in Conn_1$
14 $c_1 = c_i \uparrow 1$
15 $c_2 = c_i \uparrow 2$
16 $c_3 = c_i \uparrow 3$
17 **if** $NewNode_1(c_1) = \epsilon$
18 $B_3 = B_3 \cup \{\langle n_{new}, NodeLabel_1(c_1)\rangle\}$
19 $c_1 = n_{new}$
20 $N_3 = N_3 \cup \{n_{new}\}$
21 increment n_{new}
22 **else**
23 $c_1 = NewNode_1(c_1) \uparrow 1$
24 **if** $NewNode_2(c_2) = \epsilon$
25 $B_3 = B_3 \cup \{\langle n_{new}, NodeLabel_1(c_2)\rangle\}$
26 $c_2 = n_{new}$
27 $N_3 = N_3 \cup \{n_{new}\}$
28 increment n_{new}
29 **else**
30 $c_2 = NewNode_1(c_2)$
31 $Conn_3 = Conn_3 \cup \{\langle c_1, c_2, c_3\rangle\}$
32
33 **for each** $c_i \in Conn_2$
34 $c_1 = c_i \uparrow 1$
35 $c_2 = c_i \uparrow 2$
36 $c_3 = c_i \uparrow 3$
38 **if** $NewNode_1(c_1) = \epsilon$
39 $B_3 = B_3 \cup \{\langle n_{new}, NodeLabel_2(c_1)\rangle\}$
40 $c_1 = n_{new}$
41 $N_3 = N_3 \cup \{n_{new}\}$
42 increment n_{new}
43 **else**
44 $c_1 = NewNode_2(c_1)$
45 **if** $NewNode_2(c_2) = \epsilon$
46 $B_3 = B_3 \cup \{\langle n_{new}, NodeLabel_2(c_2)\rangle\}$
47 $c_2 = n_{new}$
48 $N_3 = N_3 \cup \{n_{new}\}$
49 increment n_{new}
50 **else**
51 $c_2 = NewNode_2(c_2)$
52 $Conn_3 = Conn_3 \cup \{\langle c_1, c_2, c_3\rangle\}$
53
54 $AltJoins = AltJoins \cup \{\langle N_3, B_3, Conn_3\rangle\}$

The two input graphs' links are processed in two sections. Lines 12-31 handle graph 1, lines 33-52 handle graph two, in exactly similar fashion. Each node in each graph is replaced by its merge from the $NodeMaps$, if present, or by a new (renumbered) node if not. The new graph in $Conn_3$ and B_3 is built incrementally. Set union ensures that links are not duplicated (lines 31 and 52). $AltJoins$ collects the alternative joins as they are produced (line 54).

The first alternative is the graph given by:

$$\begin{aligned} N_3 &= \{1,2,3\} \\ B_3 &= \{\langle 1,d\rangle, \langle 2,e\rangle, \langle 3,a\rangle\} \\ Conn_3 &= \{\langle 1,2,f\rangle, \langle 2,1,f\rangle, \langle 2,1,b\rangle, \langle 1,2,b\rangle\ \langle 1,3,f\rangle, \langle 3,1,b\rangle\} \end{aligned}$$

The other alternatives are similarly obtained.

5 Maximal project

The same two graphs used in the join section project in the following alternative ways:

1. $\boxed{\text{A}} \leftarrow \boxed{\text{A}}$

   projecting A_1 into E and A_2 into B.

2. $\boxed{\text{A}} \leftrightarrow \boxed{\text{A}}$

   projecting A_1 into E and C into B.

3. $\boxed{\text{A}} \rightarrow \boxed{\text{A}}$

   projecting A_2 into E and either A_1 into B, or C into B.

4. $\boxed{\text{A}} \rightarrow \boxed{\text{C}}$

   projecting C into E and A_1 into B.

5. $\boxed{\text{A}} \leftrightarrow \boxed{\text{C}}$

   projecting C into E and A_2 into B.

5.1 The algorithm for maximal project

The algorithm is again in two parts, the first one being identical to the first part of join, except that the criterion for pairing nodes is the miminal common supertype function, $M_p : L \times L \rightarrow L$. Just as $\perp$ is not allowed in join, so $\top$ is not allowed in project, e.g.

$$M_p(a,b) = a,\ M_p(b,c) = a\ etc.$$

. The set of alternative pairings produced by **PairNode**, $A^{project}$ is the same as A^{join}.

The merge operation is also very similar, except that links from $Conn_1$ are only copied into $Conn_3$ if

$$NewNode_1(c_1) \neq \epsilon \wedge NewNode_1(c_2) \neq \epsilon$$

There is no need to take links from $Conn_2$ since project only looks for common elements between the two graphs. The code corresponding to lines 17-52 is:

17 **when** $NewNode_1(c_1) \neq \epsilon \wedge NewNode_2(c_2) \neq \epsilon$
18 $B_3 = B_3 \cup \{\langle n_{new}, NodeLabel_1(c_1)\rangle\}$
19 $c_1 = n_{new}$
20 $N_3 = N_3 \cup \{n_{new}\}$
21 increment n_{new}
22 $B_3 = B_3 \cup \{\langle n_{new}, NodeLabel_1(c_2)\rangle\}$
23 $c_2 = n_{new}$
24 $N_3 = N_3 \cup \{n_{new}\}$
25 increment n_{new}
26 $Conn_3 = Conn_3 \cup \{\langle c_1, c_2, c_3\rangle\}$

Unlike the join algorithm which guarantees at least one join if $A^{join} \neq \emptyset$, there could be no such projection if the paired nodes in question have no link in either graph, other than those to non-paired nodes. Furthermore, there is no guarantee that the resultant graph is connected, merely that it does not contain isolated nodes. A check for connectedness can be done by traversing $Conn_3$ before it is added (with N_3 and B_3) to $AltProjections$, the accumulating set corresponding to $AltJoins$ in the join algorithm. Thus:

27 $connected = TRUE$
28 $N = copy\ N_3$
29 $V = \{$ any element of $N\}$
30 $N = N - V$
29 **while** $N \neq \emptyset$
30 $n =$ any element of V
31 $V = \emptyset$
32 **for each** $c_i \in Conn_3$
33 $V = V \cup \{n_j : (n_j = c_i \uparrow 1 \wedge c_i \uparrow 2 \in V) \vee (n_j = c_i \uparrow 2 \wedge c_i \uparrow 1 \in V\}$
32 **when** $V = \emptyset$
33 $connected = FALSE$
34 **exit while**
35 $N = N - V$
36 **when** $connected = TRUE$
37 $AltProjections = AltProjections \cup \{\langle N_3, B_3, Conn_3\rangle\}$

If these additions are applied to the example case, the first alternative is the graph given by:

$$\begin{aligned} N_3 &= \{1, 2\} \\ B_3 &= \{\langle 1, a\rangle, \langle 2, a\rangle\} \\ Conn_3 &= \{\langle 2, 1, f\rangle, \langle 1, 2, b\rangle\} \end{aligned}$$