Chapter 8
Other Specialization/Generalization Operations

Overview

This chapter is about more complex specialization/generalization operations than the elementary specialization/generalization operations studied in Chap. 2. In order to focus on the main ideas, the conceptual graphs considered here are BGs, and not SGs. Moreover, for operations involving compatibility notions (i.e., maximal join and extended join) we consider conjunctive concept types.

In Sect. 8.1, we show how the greatest specialization (or greatest lower bound) and the least generalization (or least upper bound) of two BGs can be computed. Computing the least generalization of two formulas is a fundamental problem in inductive inference. Computing a common specialization of two (or more) BGs is required in various applications. The greatest lower bound is usually not a "good" notion since it does not merge BGs, so several specific common specializations of two graphs were defined. They are often designed for specific applications, and we give here the main ideas that can be used to define operations fitted for a given application. Computing a common specialization of two graphs G and H consists of establishing a correspondence between nodes in G and nodes in H, then merging corresponding nodes. The operations can be distinguished by the kind of correspondence between G and H (e.g., bijection between subsets of concept nodes or general correspondence between a subgraph of G and a subgraph of H), and by a maximality property of the correspondence.

In Sect. 8.2, the notion of a compatible set of concept nodes of a graph is reviewed and compatible sets of relation nodes are introduced. These notions are used for defining maximal join operations. A maximal join operation between two BGs G and H consists of first merging a concept node in G and a concept node in H, and then merging as far as possible neighbors of previously merged nodes. Different neighborhood search strategies lead to different generalized join operations. Usually a generalized join operation stops when it is no longer possible to merge two nodes, so the term "maximal join" is used even though this term represents a set of operations rather than a precisely defined one.

The third section is devoted to the study of compatible partitions of node sets. We define and study compatible partitions of the concept node set of a BG, compatible partitions of the relation node set and compatible partitions of the whole node set of a BG. This last notion is strongly related to surjective homomorphisms. These notions are used for the extended join operation, which generalizes maximal join operations.

The main result in Sect. 8.4 concerns characterization of BGs that can be obtained from a set of given BGs using elementary specialization operations (sometimes known as “canonical” BGs). As a corollary, one obtains an inductive definition of the BGs built on a given vocabulary . This study is done using surjective homomorphisms and the union of a set of BGs.

Finally, in Sect. 8.5, the expansion and contraction operations used when considering defined concept types are presented.

This section is not an in-depth study of problems related to type definitions. The aim is only to provide other examples of conceptual graph operations.

8.1 The Least Generalization and Greatest Specialization of Two BGs

Computing a (or *the*) least generalization of two or more descriptions is a fundamental problem in inductive inference, which occurs particularly in machine learning. This operation can be offered by knowledge-based systems along with classical inference services. Consider, for instance, the tasks of building concept or relation definitions, or schemata typically associated with certain concepts or relations, or rules expressing general properties of certain entities. It may help to start from a set of descriptions assumed to be examples of the same concept (or relation) and consider their least generalization as a working basis. Least generalizations can also be used to organize a large set of descriptions in a hierarchical structure.

If the description language is a BG language, this problem in its basic form takes two BGs as input, say G and H, and asks for a least generalization of G and H, i.e., a BG K such that $K \succeq G$ and $K \succeq H$ and for all BG K', if $K' \succeq G$ and $K' \succeq H$ then $K' \succeq K$. If we restrict ourselves to irredundant BGs, K is unique (up to ismorphism): it is the least upper bound of G and H in the irredundant BG lattice (cf. Sect. 2.3.2). The *categorial product* of two graphs (a graph theoretic notion which can be found in the literature under a variety of other names, e.g., weak product) is used to compute a least generalization of two BGs. Let us review the categorial product of two ordinary graphs before considering BGs.

Definition 8.1 (Categorial Product). The (categorial) product of two (ordinary) graphs $G = (X,U)$ and $H = (Y,V)$ is the graph $G \times H = (Z,W)$ with $Z = \{(x,y)|x \in X \text{ and } y \in Y\}$ and $W = \{((x,y),(z,t))|(x,z) \in U \text{ and } (y,t) \in V\}$.

Example. Figure 8.1 shows an example of the product of two graphs.

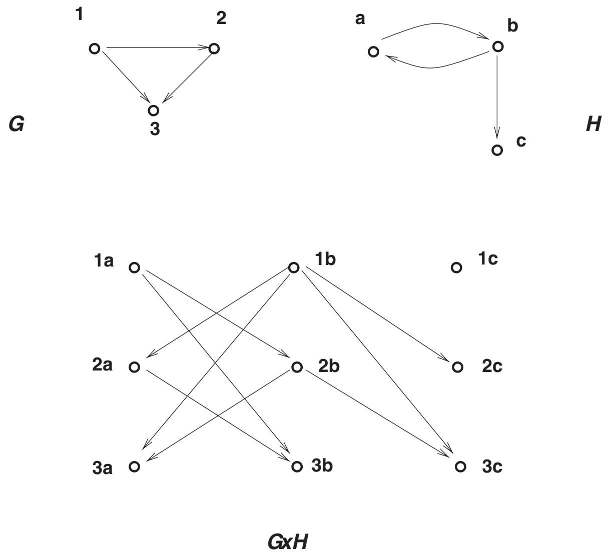

Fig. 8.1 The categorial product of two graphs

Note that $G \times H$ and $H \times G$ are isomorphic. The following properties about homomorphism can be easily observed.

Property 8.1 (Standard properties of the categorial product).

1. There is a homomorphism from $G \times H$ to G and a homomorphism from $G \times H$ to H.
2. If there is a homomorphism from K to G and a homomorphism from K to H, then there is a homomorphism from K to $G \times H$.
3. There is a homomorphism from G to $G \times H$ if and only if there is a homomorphism from G to H.

Proof. 1. Take the homomorphism from $G \times H$ to G (resp. from $G \times H$ to H), which maps each vertex (x,y) to vertex x (resp. y),
2. Take two homomorphisms $f : K \rightarrow G$ and $f' : K \rightarrow H$. Then the mapping $h : K \rightarrow G \times H$, which maps every vertex x to $(f(x), f'(x))$, is a homomorphism,
3. It follows from (1) and (2): If there is a homomorphism from G to $G \times H$, then by (1) there is a homomorphism from G to H; reciprocally, if there is a homomorphism from G to H then, since there is a homomorphism from G to G, by (2) there is a homomorphism from G to $G \times H$. □

Let us extend the definition of the graph product to BGs. Labels have to be taken into account. For ordinary graphs, we had exactly one vertex for each pair of vertices (x,y), with x in G and y in H. Now the number of nodes created from (x,y), where x and y are two concept nodes or two relation nodes in G and H respectively, is the number of minimal upper bounds of x and y labels. Since the concept type set has a greatest element, and individual markers are less than the generic marker, every pair of concept node labels has at least an upper bound. If the concept type set is a sup-semi-lattice (for instance if conjunctive concept types are considered), all pairs have a unique minimal upper bound (called their least upper bound, *lub*); otherwise, some pairs have several minimal upper bounds.

Pairs of relation labels may have any number of minimal upper bounds (including zero). Therefore, if no assumption is made on the structure of the concept and relation type sets, the size of $G \times H$ is no longer bounded by the product of the size of G and H (which is a rough upper bound) but involves the size of the vocabulary, and more precisely the maximal number of minimal upper bounds of two concept or relation types.

For the next definition, we consider the particular case where two concept labels have a least upper bound (i.e., a single least generalization) and two relation labels have either a least upper bound or do not have an upper bound. Then the number of concept nodes in $G \times H$ is $|C_G| \times |C_H|$, the number of relation nodes is bounded by the sum of $(|R^i_G| \times |R^i_H|)$, where R^i_G and R^i_H denote the set of relation nodes with arity i occurring respectively in G and in H, and the number of edges is bounded by the sum of $(|R^i_G| \times |R^i_H| \times i)$.

Definition 8.2 (BG product). Let $\mathcal{V}$ be a vocabulary where the concept type set is a sup-semi-lattice and two relation types have at most one minimal upper bound, and let G and H be two BGs on $\mathcal{V}$. The product of G and H is the BG $K = G \times H$ built as follows:

- $C_K = \{(c,d) | c \in C_G \text{ and } d \in C_H\}$, and the label of (c,d) is the lub of $l_G(c)$ and $l_H(d)$,
- $R_K = \{(r,s) | r \in R_G,\ s \in R_H$ and there is a relation label t with $t \geq l_G(r)$ and $t \geq l_H(s)\}$, and the label of (r,s) is the lub of $l_G(r)$ and $l_H(s)$,
- $E_K = \{((r,s),i,(c,d)) | (r,i,c) \in E_G \text{ and } (s,i,d) \in E_H\}$.

It can be easily checked that $G \times H$ is a BG and that the BG product satisfies the standard properties (cf. property 8.1). This operation can be extended to the product of n BGs.

Property 8.2. The least upper bound, i.e., the least generalization, of two BGs G and H is the irredundant form of $G \times H$.

Example. Consider G and H in Fig. 8.2. These BGs are obtained from the graphs in Fig. 8.1 by transforming the vertices into concept nodes and edges (u,v) into binary relation nodes, with the node assigned to u for first neighbor and the node assigned to v for second neighbor. All pairs of concept node labels have a lub. Concerning relation labels, one assumes that r and s have a lub, as well as r and t, but s and t have no upper bound. Compare the skeleton of $G \times H$ to the product graph in Fig. 8.1: the relation nodes corresponding to the edges $(1b,3a)$ and $(2b,3a)$ do not exist (since s and t have no upper bound). It can be easily checked that $G \times H$ maps to G and to H (assign to each node ij in $G \times H$ the node i in G and the node j in H). One can check that $G \times H$ is not irredundant, the gray part composed of the concept node $1c$,

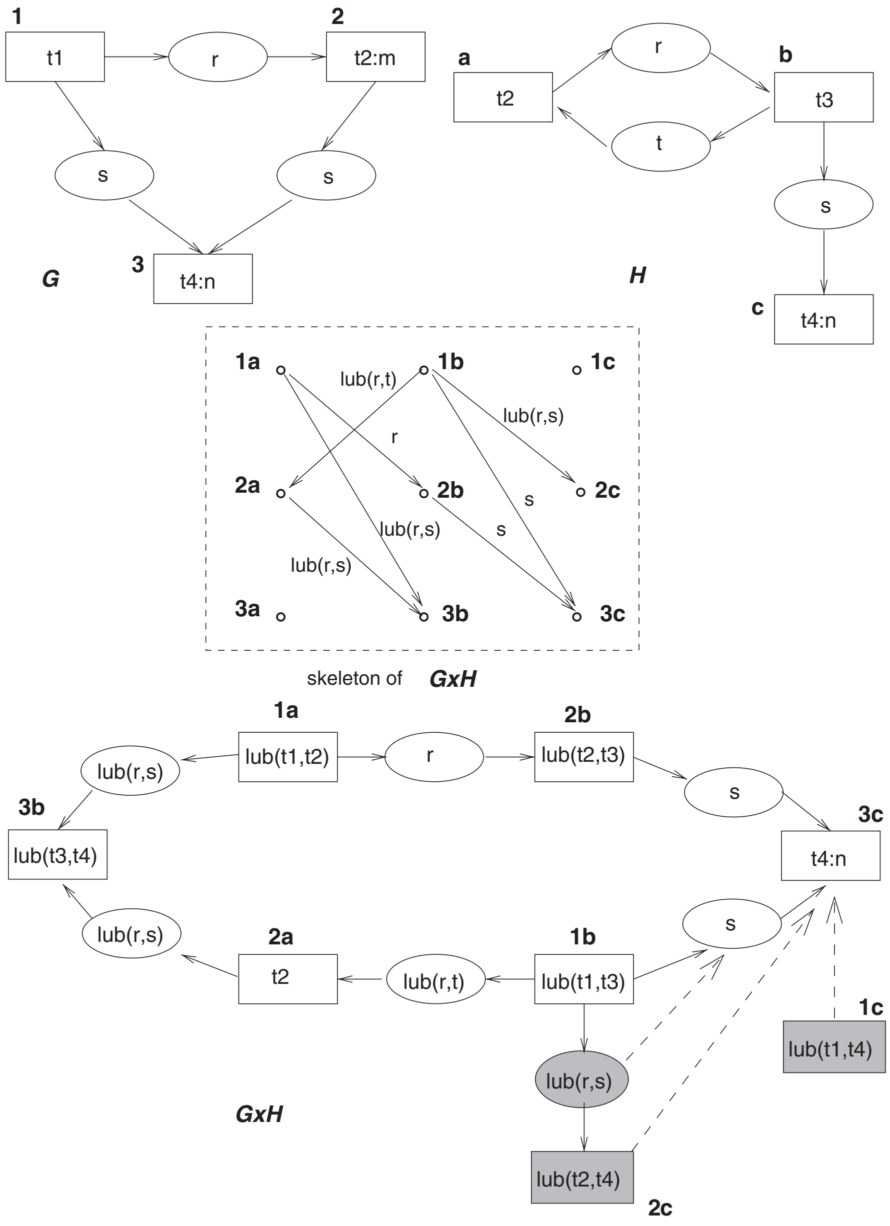

Fig. 8.2 The least generalization of two BGs

the concept node $2c$ and its neighbor relation node can be removed (cf. the dashed arrows in Fig. 8.2). The BG obtained after these deletions is the least generalization of G and H (check that it is irredundant even if some labels might actually be the same in some vocabulary).

With the assumption that the concept type set is a sup-semi-lattice, $G \times H$ is normal if G and H are normal. Indeed, an individual concept node (c,d) with marker m may be created in $G \times H$ only if c and d are both individual nodes with marker m; If G and H are normal, they may each possess at most one node with marker m. As illustrated in Fig. 8.2, $G \times H$ is generally not irredundant, even if G and H are irredundant. As far as we know, the conditions under which $G \times H$ is irredundant have not been characterized.

Computing a common specialization of two (or more) graphs is required in various applications. A *greater specialization* of two BGs is easily built: It suffices to compute their disjoint sum (see Property 2.10). Hence the property:

Property 8.3. The greatest lower bound of two BGs G and H is the irredundant form of their disjoint sum $G+H$.

$G+H$ is generally not normal or irredundant, even though G and H are normal and irredundant. However, note that given G and H irredundant, $G+H$ is irredundant if and only if there is no connected component of one of the two graphs that maps to a connected component of the other. Thus, in the case where G and H are connected irredundant graphs, the greatest lower bound (*glb*) of G and H is G if H maps to G, H if G maps to H, and otherwise it is exactly $G+H$.

The disjoint sum is easily computed but it is generally considered as unsatisfying because it does not "merge" the two graphs. This has led to the introduction of other notions of common specializations which rely on more complex operations than the elementary specialization operations studied in Chap. 2. Among them, the so-called "maximal join" is the most popular. It is studied in the next section.

8.2 Basic Compatibility Notions and Maximal Joins

In this section, we define compatible sets of concept nodes and of relation nodes. A node set can be merged into a single node if and only if it is a compatible set. Compatible sets are then used for defining the maximal join operation.

8.2.1 Compatible Node Set

Let us review the notion of compatible (or mergeable) concept nodes studied in Chap. 3, i.e., when conjunctive concept types are considered.

A set of *compatible concept nodes* (cf. Definition 3.9) is a set of concept nodes such that the set of their labels is compatible, i.e., these labels possess a greatest lower bound. The *label of a compatible concept node set* S is $l(S) = glb(\{l(x) \mid x \in S\})$.

Compatibility is strongly related to homomorphism: If a set of concept nodes is compatible, then these nodes may all be mapped to the same node by a homomorphism, and conversely.

More precisely, let us consider two BGs G and H and a homomorphism π from G to H. If y is a concept node in $\pi(G)$, then $S = \pi^{-1}(y)$ is a compatible concept node set of G. Indeed, for any $x \in S$ one has $l_G(x) \geq l_H(y)$, thus the label set of S has a lower bound (i.e., the conjunction of all types appearing in S is not a banned type and at most one individual marker appears in S), and it has a greatest lower bound.

A set of concept nodes S is compatible if and only the nodes in S can be merged into a single node. Merging these nodes consists of replacing them by a single node labeled $l(S)$ and having for neighbors the union of the neighbors of all nodes in S. In terms of elementary specialization operations: first restricting their type to the conjunction of their types, secondly, if an individual marker m appears in S, restricting their marker to m, and thirdly joining them.

Property 8.4. Let G be a BG and S be a compatible concept node set of G. The graph G/S obtained from G by merging S is a BG and there is a (surjective) homomorphism from G to G/S.

We shall now define the compatibility of relation nodes, which is more complex than the concept node compatibility.

The set of partitions of the integer set $\{1,\ldots,k\}$ is partially ordered by the usual order on partitions. Given two relations r_1 and r_2 of the same arity, P_{r_2} (i.e., the edge partition associated with r_2, cf. Definition 6.2) is thinner than P_{r_1} (notation $P_{r_2} \subseteq P_{r_1}$) if each class of P_{r_2} is included in (or equal to) a class of P_{r_1}. Let π be a BG homomorphism mapping a relation node r in G to a relation node r' in G'. Then, P_r is thinner than $P_{r'}$. Indeed, two concept nodes can have the same image, whereas a node cannot have two images. Moreover, let c' be a neighbor of r'. For any neighbor c of r such that $P_r[c] \subseteq P_{r'}[c']$, one has and $\pi(c) = c'$.
Example. Consider Fig. 8.3, where π is a homomorphism from G to G', $r(a,b,c,d,e,f)$ is in G, $\pi(r) = r'$ and $r'(u,v,u,v,w,z)$ is in G'. $P_r = \{\{1\},\{2\},\ldots,\{6\}\}$ (it is the discrete partition), and $P'_r = \{\{1,3\},\{2,4\},\{5\},\{6\}\}$. As $P_r[a] = \{1\} \subseteq \{1,3\} = P_{r'}[u]$ and $P_r[c] = \{3\} \subseteq \{1,3\} = P_{r'}[u]$, one has $\pi(a) = \pi(c) = u$.

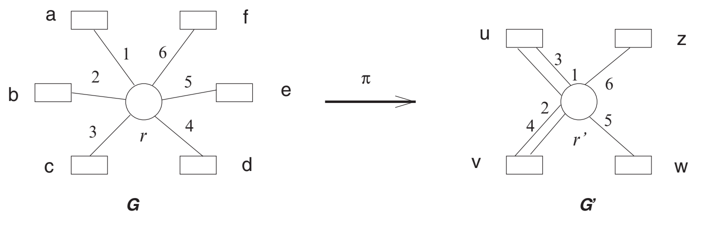

Fig. 8.3 P_r and homomorphism

Similar to concept nodes, a set of relation nodes is said to be compatible if these nodes can be merged, i.e., they may all be mapped to the same node by a homomorphism.

More precisely, let G and H be two BGs, π a homomorphism from G to H and y a relation node in $\pi(G)$. The set of relation nodes $A = \pi^{-1}(y) = \{r_1, \ldots, r_k\}$ satisfies:

1. All r_i have the same arity and furthermore for any $i = 1, \ldots, k$ one has $l_G(r_i) \geq l_H(y)$, therefore the set $\{l_G(r_1), \ldots, l_G(r_k)\}$ has a lower bound.
2. For any $i = 1, \ldots, k$, the partition P_{r_i} is thinner than or equal to P_y, and thus the least upper bound $P(A)$ of all the P_{r_i} is thinner than or equal to P_y.
3. Let $C(A)$ denotes the set of concept nodes which is the union of the neighbors of all nodes in A. For any class X in $P(A)$, let $C(X)$ be the subset of $C(A)$ containing concept nodes linked to a node in A by an edge numbered with an integer in X. Let $\mathcal{C}(A) = \{C(X) \mid X \in P(A)\}$. Every concept node in $C(A)$ belongs to at least a subset $C(X) \in \mathcal{C}(A)$ and possibly to several such subsets. $\mathcal{C}(A)$ is thus a covering of the set $C(A)$. Let $P_C(A)$ denote the thinnest partition of $C(A)$ greater than (or equal to) this covering, $P_C(A)$ is a compatible partition of $C(A)$.

Example. Let us assume that π is the homomorphism from G to H represented in Fig. 8.4, with $\pi(r_1) = \pi(r_2) = r'$. $A = \pi^{-1}(r') = \{r_1, r_2\}$. $P_{r_1} = \{\{1,2\},\{3\}\}$, $P_{r_2} = \{\{1\},\{2\},\{3\}\}$, $C(A) = \{a,b,c,d\}$, $P(A)$=$\{\{1,2\},\{3\}\}$, $\mathcal{C}(A)$=$\{\{a,b,d\},\{b,c\}\}$, and $P_C(A) = \{\{a,b,c,d\}\}$.

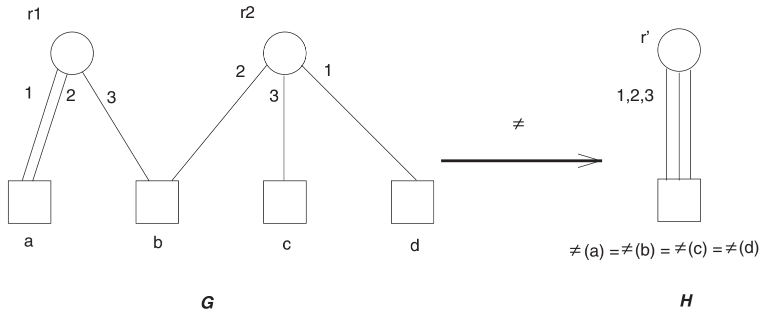

Fig. 8.4 Two relation nodes with the same image

A set of relation nodes satisfying all the previous properties is called a *compatible relation node set*.

Definition 8.3 (Compatible relation nodes). A *compatible relation node set* of a BG G is a set $A = \{r_1, \ldots, r_k\}$ of relation nodes in G satisfying the following constraints:

- All relations in A must have the same arity; moreover, the set of labels of A must have a lower bound (in the vocabulary upon which G is built),
- let $C(A)$ denote the set of concept nodes which are neighbor of at least one relation node in A; let $P(A)$ denote the lub of the partitions P_{r_i}, $i = 1, \ldots, k$; let $P_C(A)$

be the least partition containing the covering $\mathcal{C}(A)$ of $C(A)$ associated with $P(A)$; each class of $P_C(A)$ must be a compatible set of concept nodes.

As for the concept nodes, if a set of relation nodes is compatible, one can define a specialization operation that consists of merging these relation nodes.

Definition 8.4 (Compatible Relation Nodes Merging). Let $A = \{r_1, \dots, r_k\}$ be a compatible relation node set of a BG G. The graph G/A obtained from G by merging A is defined as follows.

- Compute the partition $P(A)$, i.e., the lub of the partitions P_{r_i}.
- Compute $\mathcal{C}(A) = \{C(X) \mid X \in P(A)\}$ and $P_C(A)$, i.e., the thinnest partition of $C(A)$ greater than or equal to $\mathcal{C}(A)$.
- Merge each class of $P_C(A)$ (which involves a sequence of concept restrict and join).
- Restrict the types of the relation nodes in A to $l(A)$, a maximal lower bound of $\{l(r_1), \dots, l(r_k)\}$ (now, all relation nodes in A are twin relation nodes).
- Merge A into a single relation node labeled $l(A)$ (i.e. remove all but one of the twin relation nodes in A).

Note that if the relation type set is not an inf-semi-lattice, then a set of types may have several maximal lower bounds and the operation is not deterministic.

Property 8.5. Let A be a compatible relation node set of a BG G. The graph G/A obtained from G by merging A is a BG and there is a (surjective) homomorphism from G to G/A.

Proof. Let $A = \{r_1, \dots, r_k\}$. Each class of $P_C(A)$ is a compatible concept node set; thus it can be merged (while keeping a BG). Let us check that after step 4 all nodes in A are twin nodes. They have the same label $l(A)$, so it remains to verify that if x and y are i-th neighbors of r_j and r_l, respectively, then they are in the same class of $P_C(A)$ (thus they are merged into a single node in step 3). Indeed, let X be the class of i in $P(A)$; x and y belong to $C(X)$, which is included in a class of $P_C(A)$. Thus step 5 keeps a BG. $\square$

8.2.2 Maximal Join

Maximal join is an important operation in conceptual graph applications. Intuitively, the effect of the maximal join operation is to maximally join, or merge, connected subgraphs of two graphs. It is mainly used to do plausible inference by applying conceptual schemata, typical patterns, etc., to facts.

Although there is agreement on the intuitive purpose of this operation, a lot of variants are considered in practice. In this section, we will thus present the general ideas behind the maximal join, some variations, and give a simple algorithm that illustrates one way of implementing this operation. Section 8.3 will provide the formal foundations required to define the notion of extended join, with maximal join (and its variations) being a particular case of it.

Let us start with the simplest way of joining two graphs, the *external join* operation (by distinction with the elementary join operation already defined, which is also sometimes called *internal* join). The external join consists of merging two concept nodes of two disjoint graphs, say G and H.

Definition 8.5 (External join). Let G and H be two (disjoint) BGs and c and d be two compatible concept nodes in G and H, respectively. The *external join* of c in G and d in H is the BG obtained by first, computing $G+H$, secondly, restricting the labels of c and d to their glb l, then by identifying c and d.

Since an external join can be decomposed into elementary specialization operations, the graph obtained is a common specialization of G and H.

A way of extending an external join of c in G and d in H consists of searching mergeable neighbors (i.e., relation nodes) of c and d, then to check if new concept nodes neighbors of these relations can be merged, and so on. Said otherwise, starting from a pair of compatible concept nodes, the idea is to search, in a greedy way, mergeable neighbors of previously identified mergeable nodes. The algorithm stops when it is impossible to find new mergeable nodes. The result is thus locally "maximal," hence the name *maximal join* for this class of operations.

In order to specify a maximal join operation, one has to define a condition for merging concept nodes and a condition for merging relation nodes (the nodes must be at least compatible, but stronger conditions may be enforced) as well as a strategy for exploring the graphs. Given two mergeable concept nodes as a starting point, there may be several maximal joins, but computing one of them can be done in polynomial time, whereas computing a maximal join with a *maximum* number of nodes is NP-hard (indeed it admits homomorphism or injective homomorphism as a special case, cf. Sect. 5.2).

We now describe a very simple maximal join algorithm. The graph exploration involves extending the initial external join by a breadth-first strategy. One seeks *strongly compatible* pairs of relation nodes defined as follows:

Definition 8.6 (Strongly compatible pair of relation nodes). Two relation nodes $r \in G$ and $s \in H$ are strongly compatible with respect to compatible concept nodes $c \in G$ and $d \in H$ if they fulfill:

- $type(r) = type(s)$,
- there is an integer i with $(r,i,c) \in G$ and $(s,i,d) \in H$,
- $P_r = P_s$ and for each class of P_r (or P_s) if c' (resp. d') is the neighbor of r (resp. s) associated with this class, then $\{c',d'\}$ is a pair of compatible concept nodes.

Starting from the concept node pair (c,d), the algorithm seeks in a greedy way a maximal set of strongly compatible pairs of relations with respect to c and d: a first pair (r,s) is sought, then another pair disjoint from (r,s) is sought and so on until no new pair can be built in the neighborhood of c and d. The found relation pairs yield concept node pairs, which are used as starting points for a new step.

An important point is that the compatibility of concept nodes evolves during the process. For instance, let (c_1, c_3) and (c_2, c_3) be two pairs of initially compatible concept nodes; assume that the process leads to add the pair (c_1, c_3) to f; the label of c_2 may be incompatible with the label of the node that will be obtained by merging c_1 and c_3 (i.e., $l_G(c_2)$ may be incompatible with glb$(\{l_G(c_1), l_H(c_3)\})$. That is why the algorithm maintains two labeling functions memorizing the "current label" of concept nodes: These functions are initially equal to those of the input graphs and, when a pair of concept nodes is added to f, the current label of these nodes becomes the glb of their former current labels. A concept node may appear in several pairs, thus its current label may change several times.

The result of the algorithm is first, the pair (f, g), and secondly, new labeling functions l'_G and l'_H for concept nodes. Effectively computing the maximal join of the input graphs consists of merging the nodes of f and g pair by pair. No computation is necessary to obtain the labels of the new nodes: Merged relation nodes have the same label, and merged concept nodes have the same label by l'_G and l'_H, which is exactly the label of the new node.

If the relation nodes are first merged according to g, there is no need to consider f, since concept merging follows from relation merging. If the concept nodes are first merged according to f, relation nodes of the same pair become twins, and a twin per pair has to be removed (i.e., by a sequence of relation simplify operations).

Hereafter we give the schema of a maximal join algorithm.

The set *ToExplore* contains pairs of concept nodes that have been added to f but whose neighborhood has not yet been explored. The boolean *explored* is used to explore the neighborhood of a given pair (x, y): Initially false, it becomes true when no new pairs of relation neighbors of x and y can be built.

Example. Figure 8.5 presents two BGs G and H. For simplicity, it is assumed that all concept nodes have the same label. Starting with concept nodes 1 and a, two maximal joins that do not have the same number of nodes can be obtained. The first one is defined by the mapping $1 \mapsto a$, $2 \mapsto b$, $3 \mapsto c$, and the second one by $1 \mapsto a$, $2 \mapsto d$, $3 \mapsto e$, $4 \mapsto f$, $5 \mapsto g$.

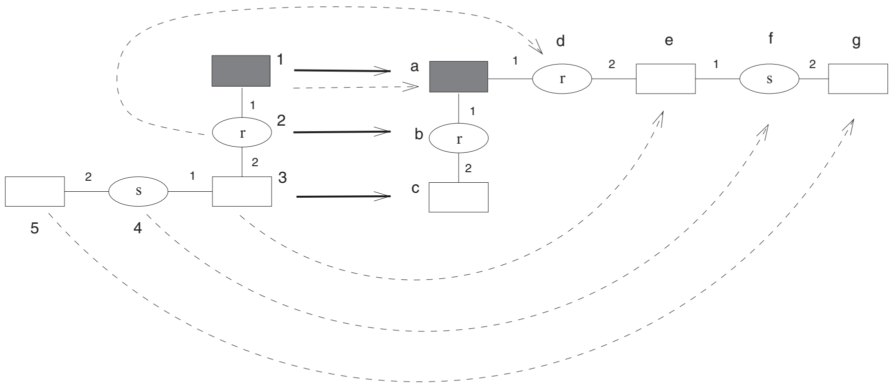

Fig. 8.5 Maximal join

Algorithm 23: Maxjoin(G,H,c,d)

```
Input: two BGs $G$ and $H$, two compatible concept nodes $c \in G$ and $d \in H$
Output: a pair $(f,g)$ defining a maximal join of $G$ and $H$ with respect to $(c,d)$, with $f$ being
        the set of concept node pairs and $g$ being the set of relation node pairs; $l'_G$ and $l'_H$
        are labeling functions
begin
    $l'_G \leftarrow l_G$
    $l'_H \leftarrow l_H$
    $g \leftarrow \emptyset$
    $f \leftarrow \{(c,d)\}$
    $ToExplore \leftarrow \{(c,d)\}$
    while $ToExplore \neq \emptyset$ do
        Remove $(x,y)$ from $ToExplore$
        $explored \leftarrow false$
        while not $explored$ do
            if there is a pair $(r,r')$ of strongly compatible relation nodes with respect to
            $(x,y)$, $l'_G$ and $l'_H$, such that $r \notin g$, $r' \notin g$ then
                $g \leftarrow g \cup \{(r,r')\}$
                foreach neighbor $e$ of $r$, with $e \neq x$ do
                    let $e' = corresponding(e,r,r')$
                    if $(e,e') \notin f$ then
                        $f \leftarrow f \cup \{(e,e')\}$
                        $ToExplore \leftarrow ToExplore \cup \{(e,e')\}$
                        $l'_G(e) \leftarrow glb(l'_G(e), l'_H(e'))$
                        $l'_G(e') \leftarrow glb(l'_G(e), l'_H(e'))$
            else
                $explored \leftarrow true$
    return $(f,g)$, $l'_G$, $l'_H$
end
```

Figure 8.6 shows that the joined subgraphs are not necessarily isomorphic. Assume that all concept nodes in G and H have the same label. Starting from $(1,a)$, each relation neighbor of 1 is matched with the relation of the same label neighbor of a. Thus $g = \{(2,b),(3,c),(4,d)\}$ and $f = \{(1,a),(5,e),(6,e),(7,f),(7,g)\}$. The concept nodes 5, 6 and e will be merged into a single node, as well as the concept nodes 7, 8 and f. In this particular example, G and H are completely joined. Thus they map entirely to the obtained graph.

Let us use this figure to illustrate the role of concept node labels. Assume that $(5,e)$ and $(6,e)$ are pairs of compatible concept nodes, but that $\{5,6,e\}$ is not a compatible set. E.g., $l_G(5) = (t,a)$, $l_G(6) = (t,b)$ and $l_H(e) = (t,*)$. If $(2,b)$ is the first relation pair added to g, then the pair $(5,e)$ is added to f, and the current label of 5 and e becomes (t,a). Consequently, $(6,e)$ is no longer a compatible pair. Thus $(3,c)$ is no longer strongly compatible and cannot be added to f. If $(3,c)$ is considered before $(2,b)$, then $(5,e)$ becomes an incompatible pair. Thus, depending on the order in which the relation neighbors of 1 and a are considered, different maximal joins are obtained.

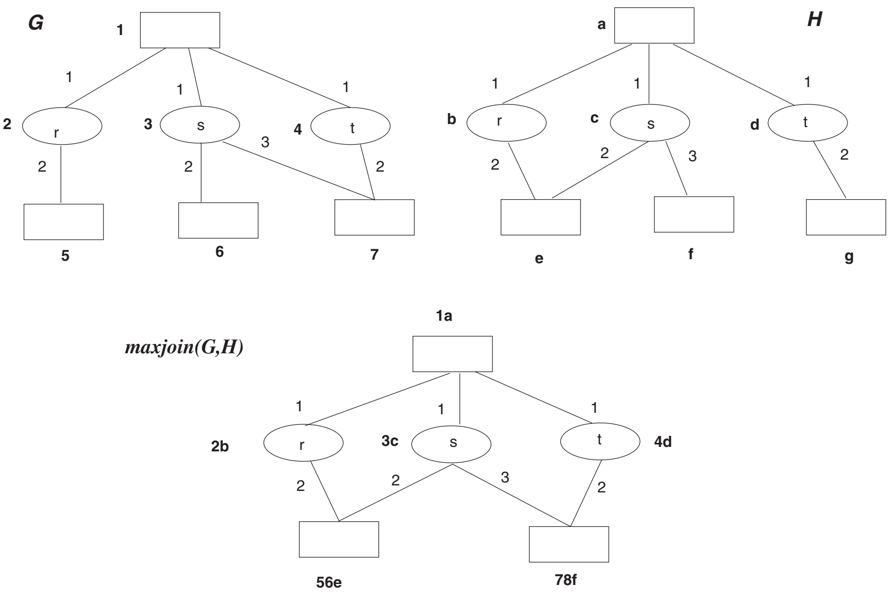

Fig. 8.6 Maximal Join of non isomorphic subgraphs

The algorithm can be seen as defining two homomorphisms from subgraphs of G and H to the same graph. More precisely, let G' (resp. H') be the subgraph of G defined by all nodes which are first (resp. second) components of an ordered pair in f or g. First, let us point out that G' and H' are subBGs of G and H respectively. Indeed, let r be a relation node in G'. There is a single relation node s in H' such that (r,s) is in g and all neighbors of r (resp. s) are first (resp. second) components of f. Thus, all neighbors of a relation node in G' are also in G'. Therefore G' is a BG (the same holds for H'). Let K denote the BG obtained by merging the corresponding nodes in f and g. K is obtained from G' and H' by specialization operations, therefore G' and H' map to K. Furthermore, these homomorphisms are injective on relation nodes, but not necessarily on concept nodes (Indeed, a concept node may appear in several pairs, as illustrated by Fig. 8.6).

Depending on the properties desired for homomorphisms from G' and H' to K, one can adapt the condition for merging relation or concept nodes. By considering stronger conditions for node merging, one obtains homomorphisms with stronger properties, but the number of nodes in G' and H' can be smaller. By considering weaker conditions for node merging, one obtains homomorphisms with weaker properties, but the number of nodes in G' and H' can be greater. Let us give some examples.

Assume we want to make f injective, so that the joined subgraphs are isomorphic. In MAXJOIN, f is initially injective (It is restricted to the ordered pair (c,d) and $c \neq d$). The following condition concerning a pair of relation nodes guarantees that f remains injective after an execution of the while loop.

Let G and H be two BGs, and let f be an injective correspondence between some concept nodes in G and some in H. A pair (r,s) of relation nodes, r in G and s in H, is *strongly compatible with respect to* f if (r,s) is strongly compatible and if, for any neighbor c of r and $c' = corresponding(c,r,s)$, either (c,c') is in f or neither c nor c' appears in f. In MAXJOIN, instead of considering "strongly compatible and disjoint pairs of relation nodes with respect to (x,y)," let us consider "strongly compatible *with respect to* f and disjoint pairs of relation nodes with respect to (x,y)". The correspondence f built by the algorithm is now injective and the restrictions of f and g to nodes in G' and H' are bijective. Thus, there are bijective homomorphisms from G' to K and from H' to K that, furthermore, do not restrict the relation node labels. Note also that the compatibility of concept nodes does not evolve during the algorithm.
Example. Applying this modified algorithm to G and H in Fig. 8.6, assuming that all concept nodes are pairwise compatible, and starting from $(1,a)$ one obtains either K or K' drawn Fig. 8.7. K is obtained with $g = \{(2,b),(4,d)\}$ and $f = \{(1,a),(5,e),(7,g)\}$, and K' with $g = \{(3,c)\}$ and $f = \{(1,a),(6,e),(7,f)\}$.

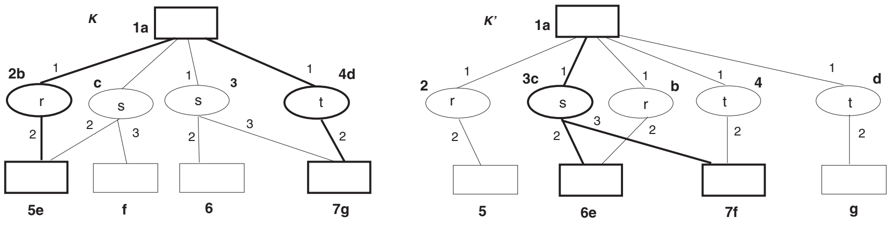

Fig. 8.7 Maximal Joins of isomorphic subgraphs

Instead of reinforcing the condition for merging relation nodes (to obtain bijective homomorphisms keeping relation node labels), it can be relaxed. For instance, one can replace the second point in the definition of a strongly compatible pair of relation nodes by " $type(r)$ and $type(s)$ have a lower bound." In this case (f,g) defines homomorphisms that are still injective on the relation nodes, but the relation node labels can be restricted.

Let us end with an example showing how the maximal join can be used to add plausible information to a BG representing a fact. A schema is a BG associated with a concept type t gathering typical or plausible information commonly accompanying the occurrence of an entity of type t (cf. Chap. 13). In Fig. 8.8, H is a schema for the concept type *Drink*, with h being its privileged concept node. This schema for *Drink* takes meaning in a "baby stories" context, i.e., usually, in a baby stories context, a drink action has for object milk and for instrument a feeding bottle that contains the milk. G is a fact containing a concept node c of type *Drink*. Computing a maximal join between G and H from an external join between c and h adds plausible relevant information to G. In the obtained graph, the soft drink has been specialized into milk, the bottle into a feeding bottle which contains the milk.

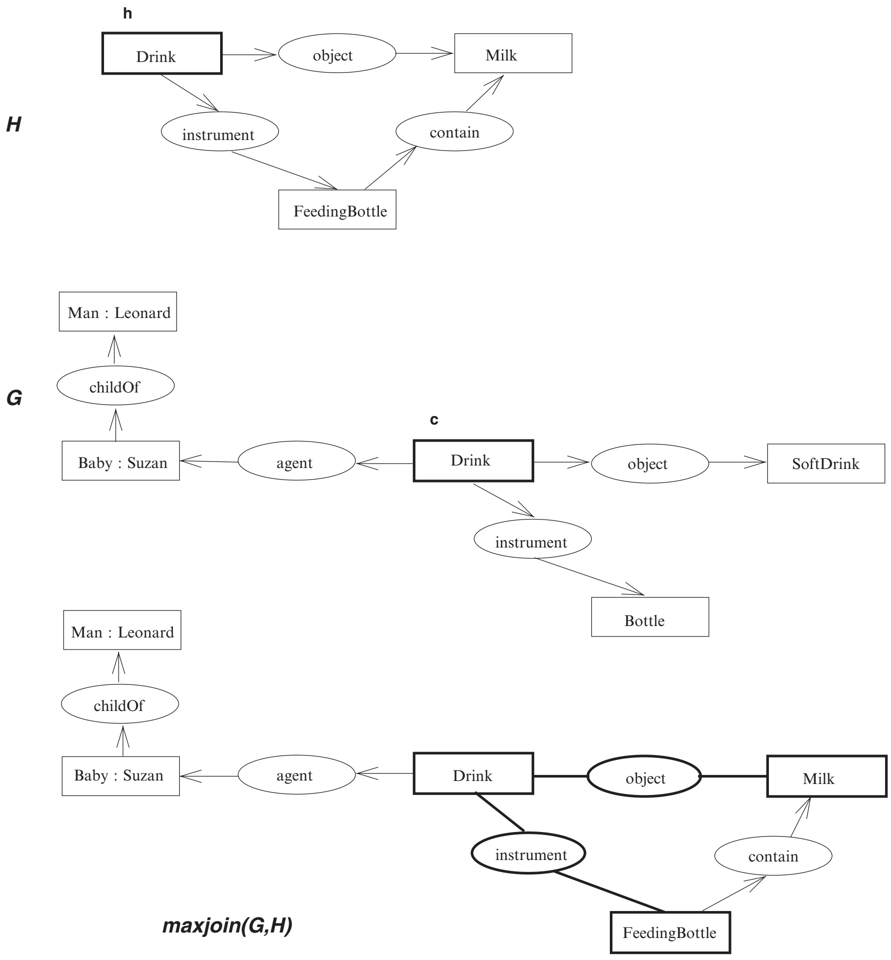

Fig. 8.8 Maximal join with a schema

When several schemas are associated with the same concept type, the situation is more complex. For instance, let us assume that the previous schema H and the schema H' shown Fig. 8.9 are both associated with the concept type *Drink*, with this second schema being related to a context "detective novels." Then maximal join can be used to determine a plausible context of the fact G in Fig. 8.8 as follows. If the number of matched nodes in $maxjoin(G,H)$ is greater than the number of matched nodes in $maxjoin(G,H')$ then "babies stories" is a plausible context of G, otherwise a plausible context of G is "detective novels." As the number of matched nodes in $maxjoin(G,H)$ is 5 whereas the number of matched nodes in $maxjoin(G,H')$ is 1, then the fact G more plausibly concerns a babies story than a detective novel.

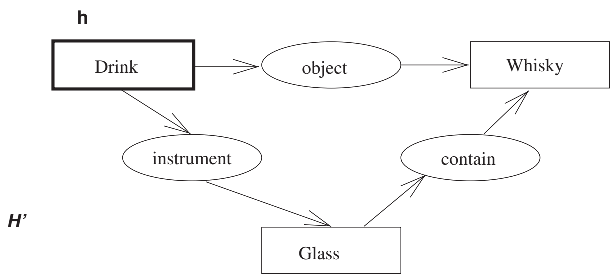

Fig. 8.9 Another schema for the concept type *Drink*

Maximal join operations are particular cases of the extended join that will be defined at the end of the next section.

8.3 Compatible Partitions and Extended Join

In this section, we first study the partition of the node set of a BG induced by a homomorphism. This notion is then used to define the extended join of two BGs. A maximal join of two BGs G and H induces two particular homomorphisms from G and H to the same graph. Similarly, an extended join of G and H induces homomorphisms from G and H to the same graph, but these homomorphisms are more general. Finally, we consider a particular extended join that merges concept node only, and is useful, for instance, in rule processing.

8.3.1 Compatible C-Partition and R-Partition

Let us now consider the partitions of the concept and relation node sets of a BG induced by a homomorphism. Let G and H be two BGs and let π be a homomorphism from G to H. A partition P_π of the node set of G can be defined as follows: Two nodes are in the same class of P_π if they have the same image by π, i.e., two nodes x and y in G are equivalent modulo P_π if $\pi(x) = \pi(y)$. P_π is composed of a partition of the concept node set of G such that any class of this partition is a compatible set of concept nodes and a partition of the relation node set of G such that any class of this partition is a compatible set of relation nodes (cf. Sect. 8.2.1).

Conversely, if P_C (resp. P_R) is a partition of the concept (resp. relation) node set of a BG G, then we give conditions for the quotient graph G/P_C (resp. G/P_R) to be a BG. In this case, there is a (surjective) homomorphism from G to G/P_C (resp. G/P_R).

If P_R is a partition of the relation node set of a BG G, G/P_R is defined like G/P_C (cf. Definition 2.22). Note that if P is a partition of a subset of the node set of a BG, one can still consider G/P by supplementing P with classes having a single node for all nodes that are not in a class of P

Definition 8.7 (Compatible C-partition). A *compatible C-partition* of a BG is a partition P of its concept node set such that any class in P is a compatible concept node set.

Property 8.6. Let G be a (normal) BG and π be a homomorphism from G to H, then π induces a compatible C-partition of G. Conversely, let P_C be a compatible C-partition of G, then the graph G/P_C obtained by merging each class of P_C is a (normal) BG and there is a surjective homomorphism from G to G/P_C.

Proof. The first part comes from the definition of a compatible set of relation nodes. The second part is obtained by recurrence on the number of classes in P_C. Let $P_C = \{S_1,\dots,S_k\}$. Then G/S_1 satisfies property 8.5. One concludes by noting that $G/\{S_1,\dots,S_k\} = (G/S_1)/\{S_2,\dots,S_k\}$. □

Let us now consider the relation nodes.

Definition 8.8 (Compatible R-partition). A *compatible R-partition* of a BG is a partition $P = \{A_1,\dots,A_k\}$ of its relation node set such that:

- Any class in P is a compatible relation node set,
- the partition $P_C(P_R)$ which is the lub of all the $P_C(A_i)'$-s is a compatible C-partition, where $P_C(A_i)'$ denotes the partition of the whole concept node set obtained by completing $P_C(A_i)$ by a trivial class for each concept node that does not belong to a class of $P_C(A_i)$.

Example. Consider in the BG of Fig. 8.10, $A = \{r_1, r_2\}$. Then $C(A) = \{a,b,c,d\}$, $P_{r_1} = \{\{1\},\{2,4\},\{3,5\}\}$, $P_{r_2} = \{\{1,2\},\{3\},\{4\},\{5\}\}$, $P_A = \{\{1,2,4\},\{3,5\}\}$ and the covering induces by P_A is $\{\{b,c,d\},\{a,c\}\}$. Thus, $P_C(A)$ is equal to $C(A) = \{a,b,c,d\}$. As $C(A)$ is compatible, A is a compatible set of relation nodes.

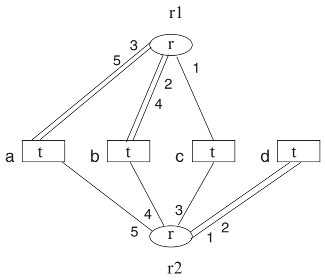

Fig. 8.10 $\{r_1, r_2\}$ is a compatible set of relation nodes

The following property is obtained with a proof similar to that given for Property 8.6:

Property 8.7. Let G be a (normal) BG and let π be a homomorphism from G to H, then π induces a compatible R-partition of G. Conversely, let P_R be a compatible R-partition of G, then the graph G/P_R obtained by merging each class of P_R is a (normal) BG and there is a surjective homomorphism from G to G/P_R.

After considering compatible C-partitions and compatible R-partitions, let us now consider both the concept and relation node sets.

A compatible partition of a BG is composed of a compatible C-partition and a compatible R-partition which are not independent.

Definition 8.9 (Compatible partition of a BG). Let G be a BG, P_C a partition of the concept node set of G and P_R a partition of relation node set of G. $P_C \cup P_R$ is a *compatible partition of G* if:

- P_R is compatible,
- P_C is compatible and $P_C \supseteq P_C(P_R)$.

Let $P = \{P_C, P_R\}$ be a compatible partition of a (normal) BG $G = (C, R, E, l)$. A quotient BG G/P is obtained from G and P as follows. First, the (ordinary) quotient graph G/P is built. Then the node associated with a class of P_C is labeled by the glb of the labels of the class, and the node associated with a class of P_R is labeled by a maximal lower bound of the labels of the class. If there are several such maximal lower bounds, then several G/P can be defined. They all have the same structure but the labels of the relation nodes can be different.

Example. Let G be the BG presented Fig. 8.11. It is assumed that $P_C = \{\{a,b,c,d,f\}, \{e\}\}$ is a compatible C-partition and that $type(r_1) = type(r_2)$ and $type(r_3) = type(r_4)$. Let us check that $P_R = \{\{r_1,r_2\},\{r_3,r_4\}\}$ is a compatible R-partition. $P_{r_1} = \{\{1\}, \{2,4\},\{3,5\}\}$, $P_{r_2} = \{\{1,2\},\{3\},\{4\},\{5\}\}$, $P(r_1,r_2) = \{\{1,2,4\}, \{3,5\}\}$. The covering associated with $\{r_1,r_2\}$ is $\{bcd, ac\}$ thus $P_C(\{r_1,r_2\}) = \{abcd\}$.

$P_{r_3} = \{\{1\},\{2\}\}$, $P_{r_4} = \{\{1\},\{2\}\}$, $P(r_3,r_4) = \{\{1\},\{2\}\}$. The covering associated with $\{r_3,r_4\}$ is $\{df, e\}$ thus $P_C(\{r_3,r_4\}) = \{df, e\}$. Finally, $P_C(R) = P_C$; therefore P_R is a compatible R-partition and $\{P_C, P_R\}$ is a compatible partition of G.

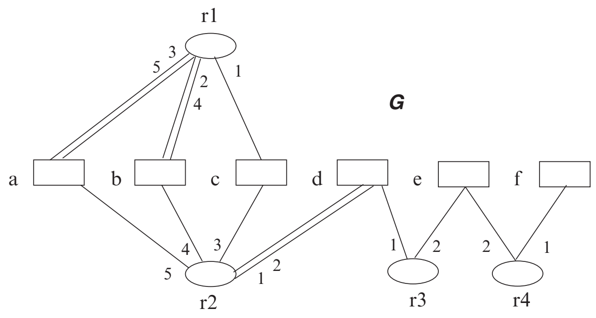

Fig. 8.11 $\{\{r_1,r_2\},\{r_3,r_4\}\}$ is a compatible partition of the relation node set

The quotient graph G/P is presented Fig. 8.12.

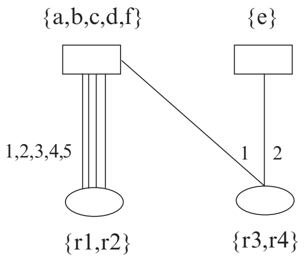

Fig. 8.12 The quotient graph G/P associated with G and P in Fig. 8.11

Theorem 8.1. *1. If P is a compatible partition of a BG G then there is a surjective homomorphism from G to (any) G/P.*

2. Let G and H be two BGs. A mapping π from G to H is a BG homomorphism if and only if the partition of the node set of G induced by the equality of the images by π is a compatible partition of G.

3. If the only compatible partition of a BG is the trivial partition, i.e., each class is restricted a single node, then it is irredundant.

Proof. 1. The mapping which associates to any node of G its class in P is a homomorphism from G to G/P.

2. If there is a homomorphism from G to H, then the partition of the node set of G induced by the equality of the images by π is a compatible partition of G. Indeed, P_C and P_R are compatible partitions by definition of a homomorphism, and P_C is greater than $P_C(P_R)$ by definition of the underlying graph homomorphism. Furthermore, there is a bijective homomorphism from G/P to $\pi(G)$, and for any class $\{c_1,\dots,c_k\}$ of P_C let y be the concept node in H such that for $i=1,\dots,p$, $\pi(c_i)=y$ then $glb(\{l(c_i) \mid i=1,\dots,p\} \geq l(y)$. The same applies for any class of P_R.

3. If the only compatible partition of a BG G is the trivial partition, i.e., each class is restricted a single node, then G is irredudant. Indeed, if there is a non-injective homomorphism from G to itself, then there is a non-trivial compatible partition. □

Note that even if a graph is irredundant, it can have non-trivial compatible partitions, as shown in Fig. 8.13 where G is irredundant; nevertheless there is a surjective homomorphism from G to H, and G has a non-trivial compatible partition.

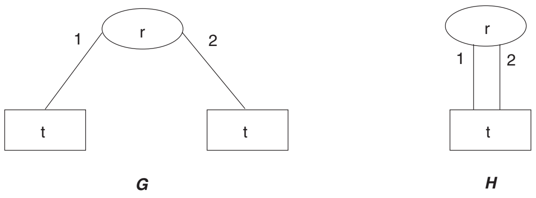

Fig. 8.13 An irredundant graph with a non-trivial compatible partition

8.3.2 Extended Join

We have considered so far two join operations, which build a common specialization of two BGs by merging parts of them. The external join can be considered as a merging of two trivial subgraphs (they are restricted to a single node). In the maximal join algorithm described in Sect. 8.2.2, two connected subgraphs, obtained by maximally extending an external join, can be merged. In this section, we consider the *extended join* operation, which generalizes the previous joins, by allowing subgraphs of any form to be merged.

Definition 8.10 (Fusionnable BGs). Let G_1 and G_2 be two BGs. They are *fusionnable* if there are partitions P_1 and P_2 respectively compatible on G_1 and G_2, such that there is an isomorphism f from G_1/P_1 to G_2/P_2, and f fulfills: For any node x in G_1/P_1, $\{x, f(x)\}$ is compatible.

The fusion of two fusionnable BGs G_1 and G_2 consists of merging any x in G_1/P_1 with $f(x)$. Let H be the BG obtained by such a fusion. H can be obtained from G_1/P_1 by replacing the label of any node x by $glb(l(x), l(f(x)))$. G_1/P_1 and G_2/P_2 are more general or equal to H, and we have:

Property 8.8. If G_1 and G_2 are fusionnable then there is a BG H and two surjective homomorphisms from G_1 to H and from G_2 to H.

Definition 8.11 (Extended join). An *extended join* operation between two BGs consists of fusionning two of their fusionnable subBGs.

The Fig. 8.14 illustrates the extended join between two BGs L and M. f_1 and f_2 are the canonical surjective homomorphisms from G_1 to G_1/P_1 and from G_2 to G_2/P_2, g_1 and g_2 are bijective homomorphisms. $g_1 \circ f1$ and $g_2 \circ f2$ are the surjective homomorphisms of Prop. 8.8.

8.3.3 Join According to a Compatible Pair of C-Partitions

The join defined in this section can be seen as a generalization of the external join, and as a particular extended join. With respect to the external join, instead of only

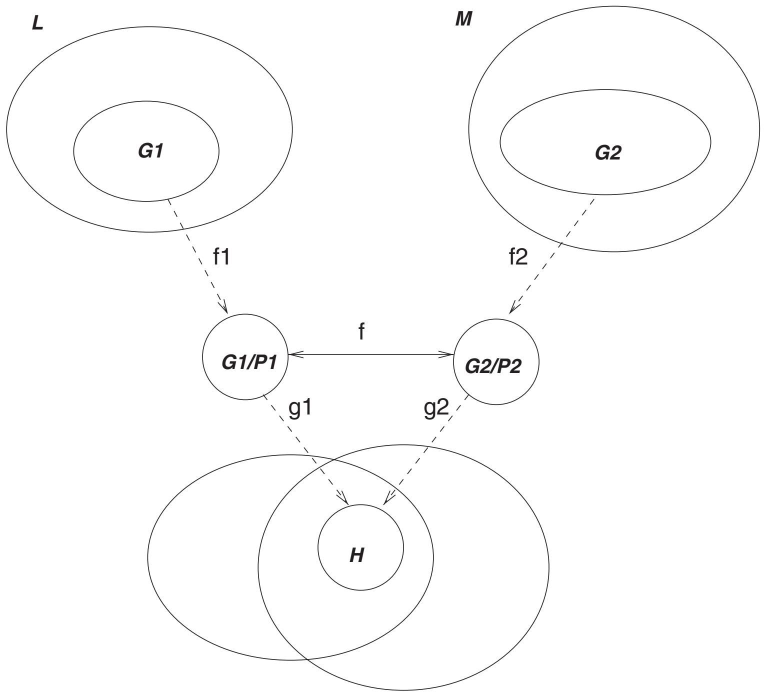

Fig. 8.14 Extended join between the BGs L and M

joining a concept c_1 in G_1 and a concept c_2 in G_2, several compatible sets of concept nodes in G_1 and in G_2 are joined. With respect to the extended join, only concept nodes are joined.

Definition 8.12 (Compatible Pair of C-partitions). Let $P_1 = (p_{1_1}, \ldots, p_{1_k})$ and $P_2 = (p_{2_1}, \ldots, p_{2_k})$ be two ordered partitions with the same cardinality over two subsets of the concept node sets of two BGs G_1 and G_2. $\{P_1, P_2\}$ is a *compatible pair of C-partitions* if for all i, $i = 1, \ldots, k$, the set $p_{1_i} \cup p_{2_i}$ is compatible.

Note that if $\{P_1, P_2\}$ is a compatible pair of C-partitions, then P_1 and P_2 are compatible C-partitions of subsets of the concept node sets of G_1 and G_2.

The join of two graphs with respect to a compatible pair of C-partitions is described in two steps. The first step consists of merging in G_1 (resp. G_2) the classes of P_1 (resp. P_2) using the labels of classes of P_2 (resp. P_1). More precisely,

Definition 8.13 (Specialization of a BG with respect to a compatible pair of C-partitions). Let $P_1 = (p_{1_1}, \ldots, p_{1_k})$ and $P_2 = (p_{2_1}, \ldots, p_{2_k})$ be a compatible pair of C-partitions of G_1 and G_2. The *specialization of* G_1 *with respect to* P_1 *and* P_2 is the graph $G_1(P_1, P_2)$ built from G_1 as follows:

- the label of any concept node in a class p_{1j} of P_1 is specialized into $glb(p_{1j} \cup p_{2j})$,
- each class of P_1 is merged.

The join of two graphs with respect to a compatible pair of C-partitions can now be defined.

Definition 8.14 (Join of two BGs with respect to a compatible pair of C-partitions). Let P_1 and P_2 be a compatible pair of C-partitions of G_1 and G_2. The *join of G_1 and G_2 with respect to P_1 and P_2* is the graph obtained from G_1 and G_2 as follows. First, build $G_1(P_1,P_2)$ and $G_2(P_2,P_1)$. Then, for all i join c_{1i} and c_{2i}, which are respectively the node of $G_1(P_1,P_2)$ obtained by merging the i-th class of P_1 and the node of $G_2(P_2,P_1)$ obtained by merging the i-th class of P_2.

Said otherwise, the join of G_1 and G_2 with respect to P_1 and P_2 is obtained from G_1+G_2 by merging the nodes of each set $p_{1_i} \cup p_{2_i}$, $1 \leq i \leq k$.

If the C-partitions are empty, the join is simply a disjoint sum of G'_1 and G'_2. If each C-partition is restricted to a single class with a single node, the join is simply an external join. Classes of P_1 and P_2 may contain concept nodes of several connected components. In such cases, the join can stick several connected components.

The join of two graphs with respect to a compatible pair of C-partitions will be used for processing rules in backward chaining (see Chap. 10).

8.4 $\mathcal{G}$-Specializations

Let $\mathcal{G}$ be a set of BGs. The main goal of this section is to give a computational characterization of the set of $\mathcal{G}$-specializations. A $\mathcal{G}$-specialization is simply a BG which can be obtained from $\mathcal{G}$ by using a finite number of elementary specialization operations (cf. Sect. 3.4). Such a set $\mathcal{G}$ is sometimes called a "canonical base" of BGs and, in this case, a $\mathcal{G}$-specialization is called a "canonical BG." BGs built on a given vocabulary can be be considered as canonical BGs generated by star BGs. Thus an inductive definition of BGs is stated. In order to achieve Theorem 8.3, which is the main result of this section, we study surjective homomorphisms and define the union of a set of (non-necessarily disjoint) BGs.

8.4.1 Surjective Homomorphism

In this section, the unary specialization operations and their relationships with homomorphism are only considered. We also state an inductive definition of a BG that differs from the definition given in Chap. 2.

Definition 8.15 (G^α). Let $\alpha = \{s,r,j\}$ be the set of the unary strict specialization operations (s for relation simplify, r for restrict, and j for join) and let G be a BG. G^α is the set of BGs inductively defined by:

- (basis) $G \in G^\alpha$,
- (rules) If $o \in \alpha$ and $H \in G^\alpha$ then $o(H) \in G^\alpha$.

One has:

Property 8.9. Let G and H be two BGs. There is a surjective homomorphism from G to H if and only if H is isomorphic to an element of G^{α}.

Proof. Let H be isomorphic to an element of G^{α}. If $H = o(G)$ with $o \in \alpha$, then there is a surjective homomorphism π from G to H defined as follows.

1. If $o = s$ and r' a twin node of r is deleted then $\pi(r') = r$ and for all node $x \neq r'$ $\pi(x) = x$,
2. if $o = r$ then π is the identity,
3. if $o = j$ and c and c' are identified then $\pi(c') = c$ and for all node $x \neq c'$ $\pi(x) = x$.

The composition of surjective homomorphisms is a surjective homomorphism, so one obtains the sufficient condition. Reciprocally, let π be a surjective homomorphism from G to H. The following sequence of unary specialization operations transforms G into H:

1. Let G_1 be the BG obtained from G by restricting the label of any concept x to the label of its image $\pi(x)$ in H,
2. let G_2 be the BG obtained from G_1 by successively joining all concepts having the same image in H,
3. let G_3 be the BG obtained from G_2 by successively simplifying all relations in G_2 having the same image in H, G_3 is isomorphic to H.

□

Property 8.10. Let G and H be BGs, then $G \succeq H$ if and only if there is a subBG of H which is isomorphic to an element of G^{α}.

Proof. If $G \succeq H$, let us consider a homomorphism π from G to H. Then there is a surjective homomorphism from G to $\pi(G)$, which is a subBG of H. One concludes with the preceding property for the necessary part. Conversely, let us suppose that K is a subBG of H, with K being isomorphic to $L \in G^{\alpha}$. Therefore, $G \succeq L$, K isomorphic to L, and $K \succeq H$, yield $G \succeq H$. □

In graph theory, a homomorphism is called *complete* if it is faithful and surjective. As a BG homomorphism is faithful, one can replace surjective homomorphism by complete homomorphism.

8.4.2 Union

The union of two (not necessarily disjoint) BGs is a partial operation defined as follows.

Definition 8.16 (Union of two BGs). Let $G=(C_G, R_G, E_G, l_G)$ and $H=(C_H, R_H, E_H, l_H)$ be two BGs such that any node common to G and H has the same label in G and H. The union of G and H is the BG $(C_G \cup C_H, R_G \cup R_H, E_G \cup E_H, l_{G \cup H})$ where $l_{G \cup H}$ is defined by:

- if x is a node in G which is not in H, then $l_{G\cup H}(x) = l_G(x)$,
- if x is a node in H which is not in G, then $l_{G\cup H}(x) = l_H(x)$,
- if x is a node in G and H, then $l_{G\cup H}(x) = l_H(x) = l_G(x)$.

Property 8.11. Let $G = (C,R,E,l)$ and $G' = (C',R',E',l')$ be two BGs such that for any node or edge x in $(C,R,E) \cap (C',R',E')$ $l(x) = l'(x)$. $(C\cup C', R\cup R', E\cup E', l\cup l')$ is a BG if and only if $graph(G) \cap graph(G')$ provided with the labeling function l (or l') is a BG.

Proof. If $(C\cup C', R\cup R', E\cup E', l\cup l')$ is a BG and r is a relation in $graph(G) \cap graph(G')$ all its neighbors must be within G and G'. Otherwise there are two edges (r,i,c) and (r,i,c') with $c \neq c'$; thus r has two i-ith neighbors in the union of G and H. Therefore, the intersection of G and G' is a BG. Reciprocally, let us suppose that $graph(G) \cap graph(G')$ is not a BG, even if for any node or edge x in $graph(G) \cap graph(G')$ $l(x) = l'(x)$. Then there is a relation r in $graph(G) \cap graph(G')$ that does not have all its neighbors in $graph(G) \cap graph(G')$, and the union of G and G' is not a BG because r does not fulfill the condition of BG. □

Whenever the union operation is defined on a set of BGs it is a specialization operation. More precisely,

Property 8.12. If a BG G is equal to the union of a set of BGs $\{G_1, \ldots, G_k\}$, then G is a specialization of the extended disjoint sum $G'_1 + \ldots + G'_k$, such that for all $i \neq j$ G'_i and G'_j are disjoint and G_i and G'_i are isomorphic for every $i = 1, \ldots, k$.

Proof. By recurrence on k. The property is true if $k = 1$. Let us consider a BG G which is the union of $k+1$ BGs $G_1, \ldots, G_{k+1}$ and let H denote the union of $G_1, \ldots, G_k$. H is a BG and it is a specialization of $G'_1 + \ldots + G'_k$. Let us consider the union of H and G_{k+1}. The intersection K of H and G_{k+1} is a BG K (cf. Property 8.11). Let us consider the disjoint sum $H + G'_{k+1}$. By joining the concepts in H and in G'_{k+1} corresponding to a concept in K, and by deleting twin relations, one obtains the union of H and G_{k+1} which is equal to G. □

Property 8.13. Let T be a specialization tree of G with k leaves labeled by $H_1, \ldots,$ H_k. For all $i = 1, \ldots, k$ there is a homomorphism π_i from H_i to G such that $G = \cup_{1,\ldots,k}\pi_i(H_i)$.

Proof. For each $i = 1, \ldots, k$ H_i is a generalization of G, therefore there is a homomorphism π_i from H_i to G, and $\pi_i(H_i)$ is a subBG of G. Let us prove by recurrence on the number n of nodes of T that the union of these sub-BGs is equal to G. The property holds if $n = 1$. If G has only one predecessor H in T then $G = op(H)$, where op is one of the unary specialization operations. By the recurrence hypothesis, H is equal to $\cup_{1,\ldots,k}\pi'_i(H_i)$. One concludes by considering $\pi_i = \pi \circ \pi'_i$ where π is the surjective homomorphism associated with op as defined in property 8.9. If G is the disjoint sum of two BGs H and K, the set $\{H_1, \ldots, H_k\}$ can be partitioned into $\{K_1, \ldots, K_l\}$ and $\{M_1, \ldots, M_m\}$, and by the recurrence hypothesis, $H = \cup_{1,\ldots,l}\alpha_i(K_i)$ and $K = \cup_{1,\ldots,m}\beta_i(M_i)$. Then G is equal to the union of these two unions, which is precisely $\cup_{1,\ldots,k}\pi_i(H_i)$. □

8.4.3 Inductive Definition of BGs

In this paragraph, BGs that can be obtained with specialization operations from a set of BGs are studied.

Definition 8.17 ($\mathcal{G}$-specialization). Let $\mathcal{G}$ be a set of BGs. A BG G is called a $\mathcal{G}$-specialization if it is obtained by a specialization tree whose leaves are all labeled by BGs in $\mathcal{G}$. Stated otherwise, the set of $\mathcal{G}$-specializations is the set of BGs inductively defined with:

- (basis) the set $\mathcal{G}$,
- (rules) the elementary specialization operations.

In Sect. 2.1.2 a star BG is defined as a BG restricted to a relation node and its neighbors (cf. Definition 2.3). Elementary star BGs are specific star BGs defined as follows:

Definition 8.18 (Elementary Star BG). The *elementary star BG* associated with a relation type r of arity k is a BG restricted to a relation node labeled by r and k neighbors, each labeled by $(\top, *)$.

The BGs on $\mathcal{V}$ can now be inductively defined as follows.

Theorem 8.2. *Let $\mathcal{G}_{\mathcal{V}}$ denote the set of elementary star BGs associated with a vocabulary $\mathcal{V}$ plus a BG, denoted $[\top]$, reduced to a single concept node labeled by $(\top, *)$. The set of $\mathcal{G}_{\mathcal{V}}$-specializations is equal to the set of (non-empty) BGs on the vocabulary $\mathcal{V}$.*

Proof. Any $\mathcal{G}_{\mathcal{V}}$-specialization is a BG on $\mathcal{V}$. Let us prove, by recurrence on the number of relation nodes, that if G is a BG on $\mathcal{V}$ then it is a $\mathcal{G}_{\mathcal{V}}$-specialization. If G has no relation node and k concept nodes x_i labeled by a_i, then G can be obtained from the BG $[\top]$ with $k-1$ disjoint sums, then restricting the labels of k nodes to $a_1, \ldots, a_k$. If a BG G has $k \geq 1$ relation nodes, then let us consider one relation node r. If H is obtained from G by deleting r, then H can be obtained by a $\mathcal{G}_{\mathcal{V}}$-specialization (recurrence hypothesis). Let us consider the subBG K of G induced by r. This BG can be obtained from the star BG associated to the type of r by a sequence of concept restrictions and joins. Finally, one makes the disjoint sum of H and K, and if (r, i, c) is in G, the node c in H is joined to the i-th neighbor of the relation in K. $\square$

Note that if the relation types are equipped with signatures (cf. Sect. 2.1.1), then the elementary star BG associated with the relation type r is the star BG having its relation node labeled r and its i-th neighbor, for $i = 1, \ldots, arity(r)$, labeled by $\sigma(r)$, i.e., the maximal concept type of the i-th neighbor of a relation node labeled r.

8.4.4 $\mathcal{G}$-Specializations

$\mathcal{G}$-specializations can occur in design problems, particularly in the synthesis or analysis of complex objects. Let us consider a box with *k* legs which may represent an abstraction of a simple physical object, e.g., a piece of a jigsaw, an electronic circuit, a mechanical device, an atom (see Fig. 8.15). The box type indicates the type of device, and leg types represent the connections of a box.

Let us assume that boxes can be glued together by means of legs for building complex objects. In Fig. 8.15, two boxes are represented as well as their gluing by identical colors.

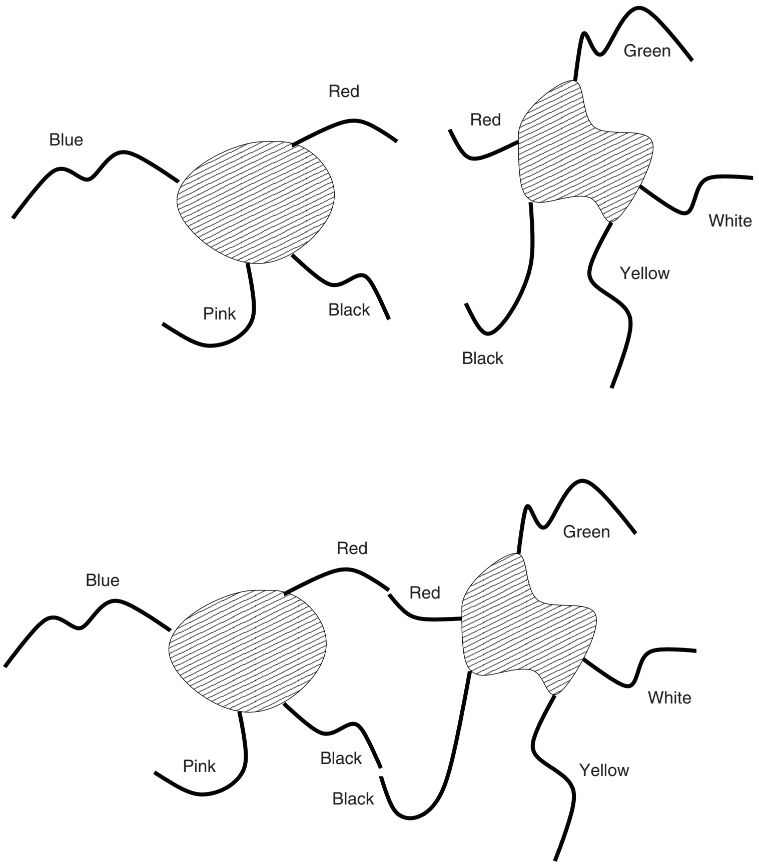

Fig. 8.15 Boxes with legs

A BG representation of boxes with legs in Fig. 8.15 is given Fig. 8.16. In this representation, colors are considered as individuals thus gluing by identical colors corresponds to normalization.

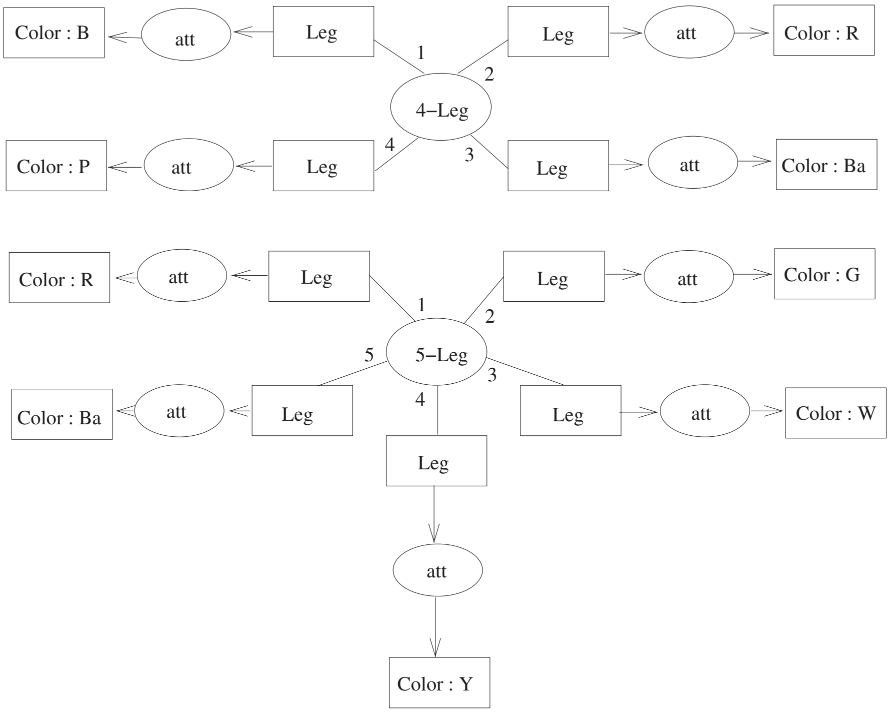

Fig. 8.16 The boxes with legs example represented as BGs

If the rules gathering boxes are elementary specialization rules (or can be defined by composition from specialization rules), then the complex objects obtainable from the boxes correspond to the set of BGs over the vocabulary consisting of the box and leg types (or a part of that BG set). Indeed, such a box with its legs can be represented by a star BG. If the elementary pieces from which more complex objects have to be built are not necessarily boxes but any set $\mathcal{G}$ of BGs, then the complex objects are $\mathcal{G}$-specializations.

A BG that is a $\mathcal{G}$-specialization is sometimes called a *canonical* BG with respect to the set $\mathcal{G}$, which is called a *canon*.

The following property can be used to build an algorithm for recognizing a $\mathcal{G}$-specialization, which is polynomial in the complexity of a BG homomorphism algorithm.

Theorem 8.3. *Let $\mathcal{G}$ be a set of BGs. A $\mathcal{G}$-*cover *of a BG G is a set $\{G_1,\ldots,G_p\}$ of elements of $\mathcal{G}$ such that: $G = \cup_{i=1,\ldots,p} G_i$. If a BG G has no isolated concepts, then the three following properties are equivalent:*

1. *G is a $\mathcal{G}$-specialization*
2. *there is a $\mathcal{G}^{\alpha}$-cover of G*
3. *for all relations r of G there is a homomorphism π from a $B_i \in \mathcal{G}$ to G with $r \in \pi(B_i)$*

Proof.

- $1 \Rightarrow 2$
  Let us consider a $\mathcal{G}$-specialization G. There is a specialization tree T of G with all its leaves labeled with elements of $\mathcal{G}$. Thus, with property 8.9, $\pi(B_i)$ is an element of B_i^{α}, and one concludes with Property 8.13 in which the union is a $\mathcal{G}^{\alpha}$-cover.
- $2 \Rightarrow 3$
  Let us consider a BG G having a $\mathcal{G}^{\alpha}$-cover $\{G_1, \ldots, G_k\}$. Any relation r of G belongs to some G_i, a subBG of G which is an element of $\mathcal{G}^{\alpha}$. One concludes with Property 8.9.
- $3 \Rightarrow 1$
  Let $B_{r_1}, \ldots, B_{r_k}$ denote the elements of $\mathcal{G}$ associated with relations $r_1, \ldots, r_k$ of G, and π_i a homomorphism such that $r \in \pi_i(B_{r_i})$. Each B_{r_i} is specialized into $\pi_i(B_{r_i})$. Then one makes the union of these BGs, which is equal to G (as there are no isolated concepts in G, a cover of the relations is a cover of the whole BG). One concludes with property 8.12.
  □

The third characterization of a $\mathcal{G}$-specialization in Theorem 8.3 leads to the following CANONICAL algorithm for $\mathcal{G}$-specialization recognition.

Let us assume that there is a function $HOM(B_i, G, r)$, where $B_i \in \mathcal{G}$, G is a BG and r is a relation node in G, which returns $\emptyset$ if there is no homomorphism from B_i to G that maps a relation node in B_i to r, otherwise it returns such a homomorphism π (and one says that "π covers r").

Algorithm 24: CANONICAL (G is a BG, $\mathcal{G}$ is a canonical base)

```
Input: a BG $G$ not reduced to a single concept node
Output: returns True if $G$ is a $\mathcal{G}$-specialization, otherwise False
begin
    $R$ is the relation node set of $G$
    $Rel \leftarrow R$
    // relation nodes not yet covered
    while $Rel \neq \emptyset$ do
        $covered \leftarrow$ False
        Let $r \in Rel$
        forall $B_i \in \mathcal{G}$ do
            $\pi \leftarrow HOM(B_i, G, r)$
            if $\pi \neq \emptyset$ then
                $Rel \leftarrow Rel \setminus \{$relation nodes appearing in $\pi\}$
                $covered \leftarrow$ True
                Exit(For loop)
        if $\neg covered$ then
            return $False$
    return $True$
end
```

Initially, *Rel* contains all relation nodes in G. The algorithm iteratively tries to cover a relation node r that has not already been covered (by a graph B_i). Whenever a homomorphism covering r is found, then other relation nodes can be covered by the same homomorphism and all these nodes are removed from *Rel*. If r cannot be covered, the algorithm returns `False`. If *Rel* becomes empty, all relation nodes have been covered and the algorithm returns `True`.

The time complexity of CANONICAL can be roughly bounded by $|R| \times |\mathcal{G}| \times$ *hom*, where *hom* is the maximum complexity of computing a homomorphism from a BG to another BG that covers a specific relation node. For known algorithms, when computing a homomorphism from a BG to another BG the covering condition does not increase the complexity. Since the problem of a homomorphism between two graphs is NP-Complete (cf. Sect. 5.2), CANONICAL is exponential here. Nevertheless, one important point is that the complexity of CANONICAL is polynomially related to the complexity of *hom*. Thus, each time *hom* is polynomial, so is CANONICAL.

8.5 Type Expansion and Contraction

The concept and relation types considered so far are primitive types. They are not explicitly defined, and their meaning is only given by their position in the type hierarchies and their possible occurrences in relation signatures, rules and other sorts of knowledge.

If a type t is less than a type t', this means that every entity of type t is also of type t', but nothing is explicitly said about the properties of entities of type t, nor about what distinguishes a t from any t'. Two kinds of type definitions have been proposed for conceptual graphs: By necessary and sufficient conditions, or with a set of prototypes representing typical instances of the type. With the first approach, an entity is of type t if and only if it fulfills the necessary and sufficient conditions. With the second approach, an entity is of type t if it is sufficiently similar to one of this prototype (We find here one application of the maximal join operations: The similarity can be related to the ratio of nodes in both graphs that are matched by a maximal join). The logical translation of the first kind of definition is simply a logical equivalence, while this is not the case for the second kind of definition.

In this section, we consider type definitions by necessary and sufficient conditions. Note that we shall not conduct an in-depth study of type definitions. Studying how to classify a defined type, i.e., how to find its position in the concept type hierarchy, or extending the specialization/generalization relation between graphs and studying its relationships with FOL semantics, studying recursive type definitions and so on, are beyond the scope of this section. Briefly, in this section we do not extend the conceptual graph model presented in this book with type definitions, we only consider expansion and contraction operations of a defined type.

Furthermore, we describe these operations for concept type definitions only. Similar operations can be provided for relation type definitions (see the bibliographical notes). Finally, let us point out that we consider a classical BG vocabulary, thus without conjunctive types.

Definition 8.19 (Concept type definition). Let $\mathcal{V}$ be a BG vocabulary and t be a symbol which does not belong to $\mathcal{V}$. A *concept type definition* of t is a unary λ-BG $(c)D_t$, where D_t is connected and c is called the *head* of D_t. Such a type definition is noted $t = (c)D_t$.

Example. Assume, for instance, that the vocabulary contains the concept type $Woman$ and the relation type $childOf$. One wants to define a new concept type $Mother$, by the λ-graph $(c)G$ in Fig. 8.17. This definition says that an entity m is of type $Mother$ if and only if m is of type $Woman$ and has a child.

Intuitively, a concept type definition is a necessary and sufficient condition for an entity to belong to this defined type. A definition $t = (c)D_t$ can be considered as an Aristotelician type definition by genus and difference. The genus of t is the type of c; its difference is what completes c in D_t. Logically, the definition $t = (c)D_t$ is naturally translated by the formula $\forall x(t(x) \leftrightarrow \Phi((c)D_t))$, with the term assigned to c in $\Phi((c)D_t)$ being x. For instance, the definition of the concept type Mother (Fig. 8.17) is translated into $\forall x(Mother(x) \leftrightarrow \exists y(Woman(x) \wedge Person(y) \wedge childOf(y,x)))$.

Even if one does not describe the modification of the ordered type set due to the addition of a defined type, let us assume that, in the enriched vocabulary, the defined type t is $\leq$ the type of c (and in general t might be less than other types which are specializations of, or incomparable with, the type of c).

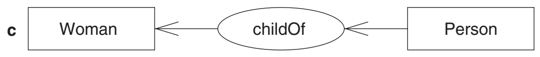

Fig. 8.17 Definition graph of the concept type *Mother*

In the same way, a relation type definition is defined as follows.

Definition 8.20 (Relation type definition). Let $\mathcal{V}$ be a BG vocabulary, r be a symbol that does not belong to $\mathcal{V}$ and k an integer ≥ 1. A *k-ary relation type definition* of r is a k-ary λ-BG $(c_1,\ldots,c_k)D_r$.

For instance, if the vocabulary contains the relation types $parentOf$ and $sibling$, then the binary relation type $cousin$ can be defined by the λ-graph $(c_1,c_2)G$ in Fig. 8.18. A definition of a relation type of arity k is a necessary and sufficient condition for k entities to be linked by this defined relation type, e.g., the entities represented by c_1 and c_2 are cousins if and only if they have parents who are siblings. It is translated into logics in the same way as a concept type definition: The formula assigned to the definition $r = (c_1,\ldots,c_k)D_r$ is $\forall x_1 ... x_k(r(x_1 ... x_k) \leftrightarrow \Phi(((c_1,\ldots,c_k)D_r)))$, where the variables assigned to $c_1,\ldots,c_k$ are $x_1,\ldots,x_k$ respectively. For instance, the definition of the concept type cousin (Fig. 8.18) is translated into $\forall x_1 \forall x_2(cousin(x_1,x_2) \leftrightarrow \exists y \exists z(Person(x_1) \wedge Person(x_2) \wedge Person(y) \wedge Person(z) \wedge parentOf(y,x_1) \wedge parentOf(z,x_2) \wedge sibling(y,z)))$.

Note that *cousin* and *sibling* are both symmetrical relations, which can be expressed by BG rules (cf. Chap. 10).

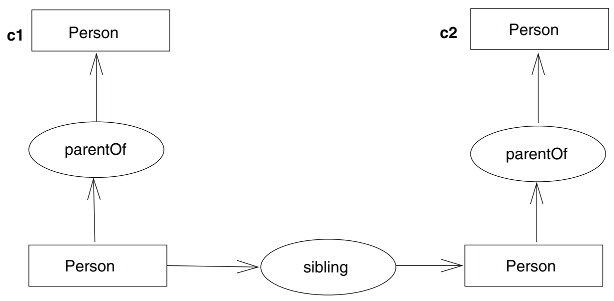

Fig. 8.18 Definition graph of the relation type *cousin*

The expansion of a concept type definition consists of replacing a concept node with a defined type by the graph defining the type. More precisely:

Definition 8.21 (Concept type expansion). Let G be a graph containing a concept node x with a defined type $t = (c)D_t$. The expansion of t at x is the BG $exp(G,x,D_t)$ obtained by merging x and c, the label of the new node being (t',m) where t' is the type of c and m is the marker of x.

An example is given in Fig. 8.19.

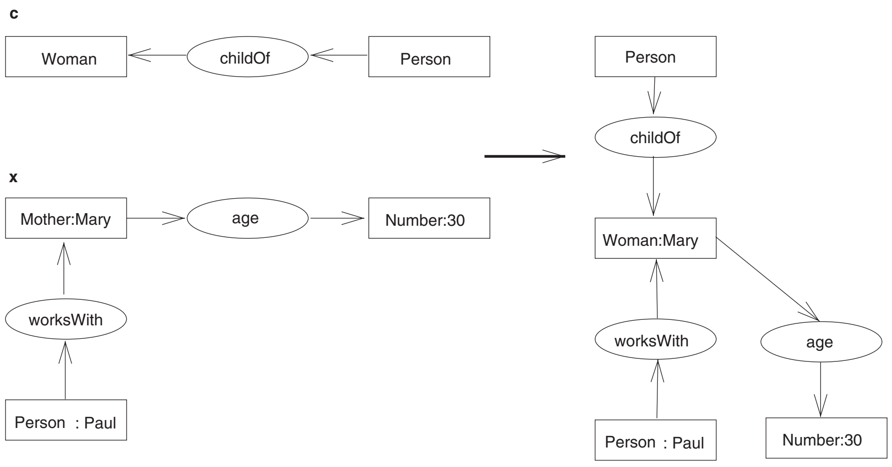

Fig. 8.19 A concept type expansion

Let us point out the relationships with rules. From a logical viewpoint, a concept type definition $t = (c)D_t$ can be considered as equivalent to two specific BG rules (and more precisely, λ-rules, cf. Chap. 10), say R_1 and R_2. The hypothesis of the first rule is a single generic concept node labeled t, its conclusion is D_t, and the node in the hypothesis is in correspondence with the head of D_t; the second rule is the reciprocal rule. See for instance Fig. 8.20, which shows both rules corresponding to the definition of the concept type *Mother*.

At first glance, a type expansion looks like an application of the rule R_1. However, there are two differences. First, R_1 can be applied to concept nodes with a type less than t, whereas an expansion can occur only if the concept node has exactly the type t. Secondly, a rule application cannot generalize the type of an existing node.

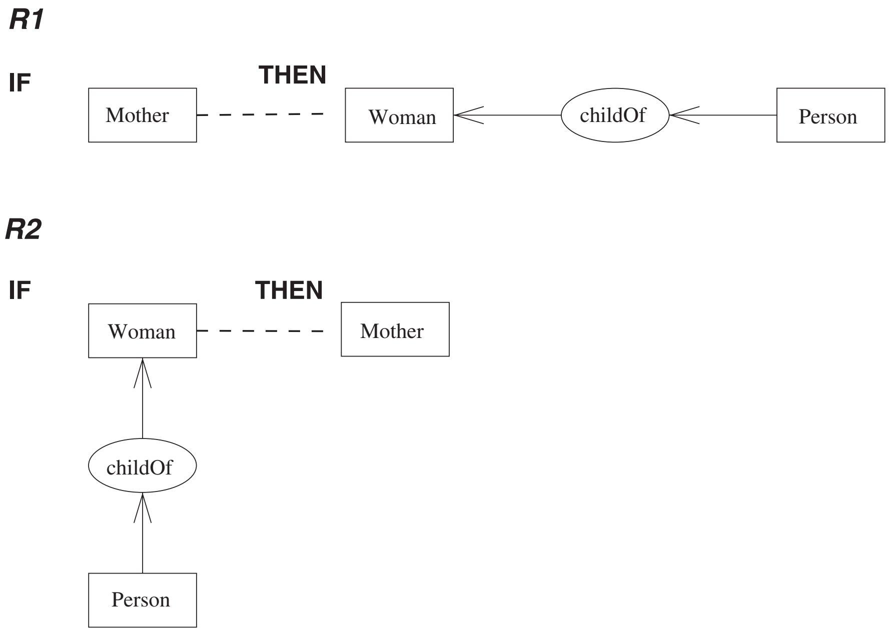

Fig. 8.20 Rules associated with the definition of the concept type Mother

A concept type expansion may add redundant information. This added redundant information can be avoided by a more complex type expansion operation relying on the piece notion that is defined hereafter. This notion will be generalized in Chap. 10, where pieces are used in backward chaining of rules.

Definition 8.22 (Piece). Let G be a BG and let x be a concept node in G. Two nodes u and v in G belong to the same *piece* of x if they are in the connected component of G containing x and if there is a chain between them that does not go through x, i.e., there is a chain $w_0(=u),\dots,w_i,\dots,w_k(=v)$ such that $w_i \neq x$ for all $i = 1,\dots,k-1$.

Note that x belongs to all pieces of x and that x has more than one piece if and only if x is a cut node of G (i.e., the deletion of x strictly increases the number of connected components).

The pieces of x can be built as follows. Let H be the connected component of G containing x, the connected components $H_1,\dots,H_k$ of the graph $H-x$ obtained from H by deleting x are computed. Then, for $i=1,\dots,k$ the piece C_i of x is the BG obtained from H_i by adding x and all edges in H joining x to a node in H_i. If $k=1$ the single piece is H itself.

The BG $exp(G,x,D_t)$ resulting from a concept type expansion can be reduced, i.e., replaced by an equivalent BG having fewer nodes, by using pieces.

Definition 8.23 (Extended concept type expansion). The *extended concept type expansion* of a defined type $t=(c)D_t$ at x in G is the BG $extexp(G,x,D_t)$ obtained from $exp(G,x,D_t)$, where x still denote the node resulting of the merging of x and c, as follows. Let $C_1,\dots,C_k$ be the pieces of x in G and let $D_1,\dots,D_l$ be the pieces of c in D_t. For all $i=1,\dots,k$ and $j=1,\dots,l$, if D_j maps to C_i with c mapped to x, then D_j is deleted from $exp(G,x,D_t)$.

In Fig. 8.21, G is a BG and D_t is the definition of the type t of x in G, all concept nodes are assumed to be generic, and the type t'' of z in G is less than the type t' of c (i.e., the head of D_t) and t. D_t and G are restricted to a single piece with respect to c and x respectively. G does not map to D_t and D_t does not map to G with c being mapped to x. Thus, $extexp(G,x,D_t)$ is equal to $exp(G,x,D_t)$. One can also check that G and D_t are irredundant. Nevertheless, $extexp(G,x,D_t)$ is redundant. Indeed, by folding z onto xc, $extexp(G,x,D_t)$ is transformed into K, which is equivalent to the subBG H of G.

Note that the reduction used in the extended concept type expansion can also be done after any external join. Figure 8.21 can be used to show (consider that D_t is a BG and not necessarily a type definition, i.e., x and c have the same labels) that after such a reduction the obtained BG can be redundant even though the two joined BGs are irredundant.

Let us now consider the type contraction operation, which can be more or less considered as inverse to the expansion. In order to define this contraction operation, we use the fact that x is a cut node of $exp(G,x,D_t)$, i.e., if x is deleted from $exp(G,x,D_t)$, two disjoint (not necessarily connected) graphs are obtained, which correspond to D_t-c and $G-x$.

Let us assume that a graph G contains a subBG D isomorphic to a concept type definition graph D_t. If D is shrunk into a single node, then all neighbors of the nodes in D that are outside D are linked to the new node resulting from the shrinking of D. This can add irrelevant information, as shown in Fig. 8.22.

In contracting a subgraph corresponding to a type definition, one does not want to add irrelevant information, but one also does not want to lose relevant information. For instance, if one knows that Paul is a child of the woman Mary, i.e., if G in Fig. 8.23 is a fact and if we simply contract the definition of *Mother*, one obtains graph H and the fact that Paul is a child of Mary is lost.

To avoid information loss in type contraction, the parts that strictly specialize the corresponding parts in the type definition are kept. More precisely,

Definition 8.24 (Concept type contraction). Let G be a BG and D_t be a concept type definition, with head c. Let π be a homomorphism from D_t to G, such that c

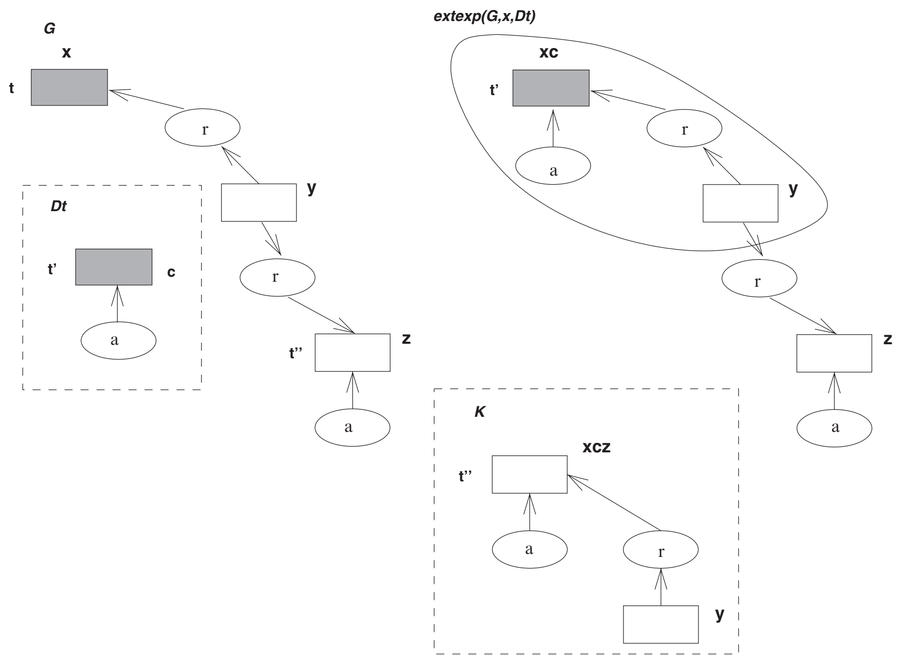

Fig. 8.21 Extended concept type expansion and redundancy

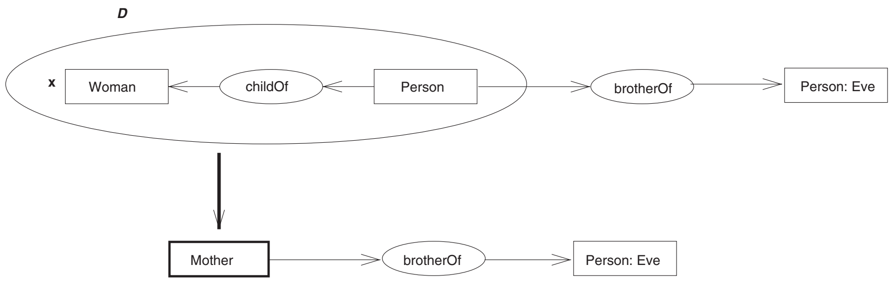

Fig. 8.22 A problematic contraction of a concept type

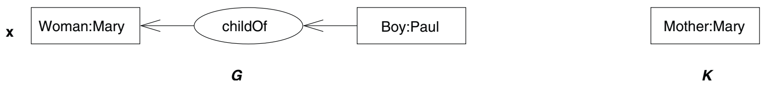

Fig. 8.23 A concept type contraction with information loss

and $\pi(c)$ have the same type. Let $x = \pi(c)$. The BG $contract(G,x,D_t)$ is obtained from G by replacing the type of x with t and, for each piece P of x in G that maps to a piece of c in D_t (without considering the label of x), deleting $P - x$.

An example is given Fig. 8.24, where a *4-Wheel* is a concept type less than the concept type *BigCar*.

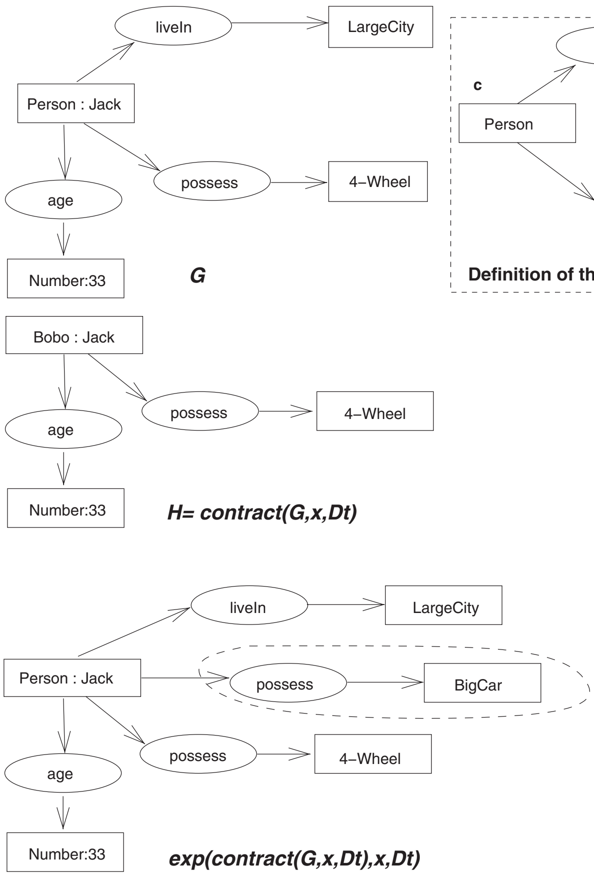

Fig. 8.24 Concept type contraction

Note that $contract(exp(G,x,D_t),x,D_t) = G$. Indeed, the type of x in $exp(G,x,D_t)$ is t and it is unchanged by the contraction. Furthermore, the homomorphism is a BG isomorphism, thus no subBG is added by the contraction. When first computing a contraction then an expansion, i.e., $exp(contract(G,x,D_t),x,D_t)$, the graph obtained is not necessarily equal to G (an example is given in Fig. 8.24 when the type Bobo is expanded in H). Nevertheless, $exp(contract(G,x,D_t),x,D_t)$ is hom-equivalent to G since the deleted pieces in G are redundant (they map to pieces in D_t) as well as the pieces of D_t which are not in G.

Finally, let us point out that the concept type expansion and contraction operations, as defined in this chapter, do not change the BG semantics. More precisely, let G be a BG and G' be obtained from G by a type expansion or contraction. Let f be the formula translating the type definition involved in this operation. Then $\Phi(\mathcal{V}), f \models (\Phi(G) \leftrightarrow \Phi(G'))$.

Like the concept type definition, a relation type definition can be considered as equivalent to two reciprocal BG rules and similar relation type expansions and contractions can be defined.

8.6 Bibliographic Notes

In [Sow84] a maximal join operation is proposed that was implemented in the system presented in [SW86]. The definition of a maximal join between two graphs G and H in [Sow84] relies on the notion of a common generalization of G and H, say K, and "compatible" homomorphisms from K to G and H respectively, say π_1 and π_2. The join is performed according to these homomorphisms by pairwise merging each node $\pi_1(x)$ and $\pi_2(x)$ for each node x of K. However, the definition of compatibility (Definition 3.5.6 in [Sow84]) and the associated result (Theorem 3.5.7 in [Sow84]) work only if the homomorphisms are *injective* on the concept node set of K. Then, there is a bijection between the concept nodes of the joined subgraphs of G and H. Maximal joins have been used especially in natural language processing (e.g., cf. [FLDC86]).

Compatible partitions were introduced in [CM92]. Canonical conceptual graphs were introduced in [Sow84], and the characterization Theorem 8.3 as well as the recognition algorithm of canonical graphs were given in [MC93].

Type definitions as well as type expansion and contraction operations were introduced by Sowa [Sow84]. Two kinds of type expansion are defined: the minimal type expansion, which is the same as ours, and the maximal type expansion, which essentially extends the minimal type expansion with a maximal join. This second operation aims at avoiding the creation of redundant parts; but contrary to our extended type expansion it does not preserve the semantics and might strictly specialize the graph. We kept the overall idea of the type contraction operation proposed by Sowa and precisely define it with graph operations.

Although type definitions were introduced as early as 1984, their processing received little attention. Let us mention the work of Leclère (cf. [Lec95], [Lec97] and [Lec98]) extending the framework of BGs to concept and relation type definitions. In this framework, recursive type definitions (direct or indirect) are forbidden and a primitive type cannot be specialized by a defined type. The specialization relation between BGs is extended by the type expansion and contraction operations, and it is shown that soundness and completeness properties are kept, i.e., given two BGs G and H defined on a vocabulary $\mathcal{V}$, $G \succeq norm(H)$ if and only if $\Phi(\mathcal{V}), \Phi(H) \models \Phi(G)$, where $\Phi(\mathcal{V})$ is the set of formulas associated with the enriched vocabulary, which now includes formulas translating type definitions. A unique expanded form obtained by performing type expansion until stability is associated with each BG. The expanded form is a BG where all types are primitive (i.e., not defined). Thus two expanded BGs can be compared by homomorphism. Let us add that to the best of our knowledge type definitions have never been studied in conceptual graphs beyond the BG model.