Chapter 2
Basic Conceptual Graphs

Overview

This chapter presents the kernel of the knowledge representation language, namely basic conceptual graphs. A *basic conceptual graph (BG)* has no meaning independently from a vocabulary, and both are defined in Sect. 2.1. The fundamental notion for reasoning with BGs is the *subsumption* relation. Subsumption is first defined by a *homomorphism*: Given two BGs G and H, G is said to subsume H if there is such a BG homomorphism from G to H. Section 2.2 defines the BG homomorphism, as well as the particular case of BG isomomorphism. BGs and subsumption provide a basic query-answering mechanism, as shown at the end of this section. The subsumption properties are studied in Sect. 2.3. Subsumption is not an order as there may be nonisomorphic equivalent BGs, i.e., BGs that subsume each other. However, by suppressing redundant parts, any BG can be transformed into an equivalent *irredundant* BG. Irredundant BGs are unique representatives of all equivalent BGs. Finally, the subsumption relation restricted to irredundant BGs is not only an order but also a lattice. Section 2.4 introduces another way of defining the subsumption relation by sets of elementary graph operations. There is a set of *generalization* operations and the inverse set of *specialization* operations. Given two BGs G and H, G subsumes H if and only if G can be obtained from H by a sequence of generalization operations (or equivalently, H can be obtained from G by a sequence of specialization operations). Section 2.5 introduces the issue of equality, which will be a central topic of the next chapter (Simple Conceptual Graphs). A BG is said to be *normal* if it does not possess two nodes representing the same entity. Normal BGs form the kernel of reasoning based on homomorphism.

In Sect. 2.6, the complexity of BG fundamental problems is studied. In particular, it is proven that checking whether there is a homomorphism between two BGs is an NP-complete problem.

2.1 Definition of Basic Conceptual Graphs (BGs)

2.1.1 Vocabulary

Basic conceptual graphs (BGs) are building blocks for expressing different sorts of knowledge: Assertions or facts, queries or goals, rules describing implicit knowledge, rules describing the evolution of some world, constraints and so on. In this chapter they are used for representing facts and queries, but they will be used for representing more complex knowledge in further chapters.

A fact is an assertion that some entities exist and that these entities are related by some relationships. Any entity has a type (e.g., Car, Person, Toy, etc.), and the set of types is ordered by a *subtyping* relation, also called a *specialization* relation or *a-kind-of* relation (e.g., the type Boy is a kind of the type Person). "The type t is a specialization of the type t'," equivalently "t' is a generalization of t," simply means that any entity of type t is also of type t'. It is assumed that there is a most general type, called the *universal* type, denoted $\top$.

Two kinds of entities are considered. An entity may be either a specific entity (e.g., *that* small red car) or an unidentified entity (e.g., *a* boy). A specific entity is called an *individual* entity and an unidentified entity is called a *generic* entity.

Besides asserting the existence of specific or unidentified typed entities, a fact can assert that relationships hold between these entities (e.g., Paul *possesses* a teddy bear, Paul *is-the-son-of* a person and so on). The relations are structured by a specialization order as the types of entities are. For example, the relation *is-the-son-of* is a specialization of the relation *is-the-child-of*, which is itself a specialization of the relation *is-a-relative-of*. The representation of an entity (of the application domain) is traditionally called a *concept* in the conceptual graphs community and we use this expression hereafter. A basic conceptual graph (BG) is composed of two kinds of nodes, *concept* nodes representing entities that occur in the application domain, and *relation* nodes representing relationships that hold between these entities. The ordered set of concept (entity) types is denoted T_C. An individual concept is referenced by an *individual marker* belonging to a set $\mathcal{I}$ of individual markers, and there is a *generic marker* $*$, which denotes an unspecified entity. The same marker $*$ is used for denoting a generic entity regardless of type. The set of relations is denoted T_R. An element of T_R is called a *relation symbol or a relation type*. These three sets compose the vocabulary used for labeling the two kinds of nodes of a BG: A concept node is labeled by a pair composed of a type and either an individual marker or the generic marker; a relation node is labeled by a relation symbol. A vocabulary is precisely defined as follows:

Definition 2.1 (Vocabulary). A *BG vocabulary*, or simply a vocabulary, is a triple $(T_C, T_R, \mathcal{I})$ where:

- T_C and T_R are finite pairwise disjoint sets.
- T_C, the set of *concept types*, is partially ordered by a relation $\leq$ and has a greatest element denoted $\top$.

- T_R, the set of *relation symbols*, is partially ordered by a relation $\leq$, and is partitioned into subsets $T_R^1, \ldots, T_R^k$ of relation symbols of arity $1, \ldots, k$, respectively. The arity of a relation r is denoted $arity(r)$. Any two relations with different arities are not comparable.
- $\mathcal{I}$ is the set of *individual markers*, which is disjoint from T_C and T_R. Furthermore, $*$ denotes the *generic marker*, $\mathcal{M} = \mathcal{I} \cup \{*\}$ denotes the *set of markers* and $\mathcal{M}$ is ordered as follows: $*$ is greater than any element in $\mathcal{I}$ and elements in $\mathcal{I}$ are pairwise incomparable.

In some works, it is assumed that T_C has a specific structure, such as a tree, a lattice or a semi-lattice. In this book, having a greatest element is the only property required for the ordered set T_C.

A set T_C (resp. T_R) of concept (resp. relation) types is also called a *hierarchy* of concept (resp. relation) types.

The three sets (concept, relation or marker set) play different roles and are assumed to be pairwise disjoint. A specific syntax is used for representing elements of a vocabulary: A concept type begins by an upper case letter, a relation symbol by a lower case letter, and an individual is a proper name or begins by #. This allows the user to quickly determine the role of an identifier and to simply differentiate close identifiers playing different roles. For instance, the word *father* could represent a concept type (e.g., Paul is an individual of type *Father* means that Paul is a father), a binary relation symbol *fatherOf* could represent the binary relation relating a father and one of his children (e.g., $fatherOf(Paul, July)$ means that Paul is the father of July), and if "Father" is the name of the chief of a gang then Father could be used as an individual marker.

Example. Figure 2.1 is a subset of the concept type set corresponding to the children photo example (Sect. 1.1.1). Figure 2.2 is a part of the relation type set for the same example. In this case, for each arity i of relation, there is a greatest element denoted by $\top_i$. $\top_i$ can be seen as representing any relation of arity i.

Type of Individuals and Relation Signatures

A vocabulary can be considered as the representation of a very simple ontology. Different kinds of knowledge can be added to a vocabulary. Two simple extensions are often considered, namely *individual typing* and *relation symbol signatures*.

First, individual markers can be typed. The type of an individual marker represents the most specific information about the category of the individual referenced by this marker. A mapping, say τ, from the set of individuals $\mathcal{I}$ to the set of concept types T_C is thus added with, for an individual m, $\tau(m)$ representing the most specific type of m. Consequently, in any BG relative to this vocabulary, a concept node with marker m will have a type greater than or equal to $\tau(m)$. For instance, let us assume that, in some applications, there is an individual marker $R2$ known to represent a $ToyRobot$ (without any other details) i.e., $\tau(R2) = ToyRobot$. Individual concept nodes with marker $R2$ and type Toy, or $Robot$, or $MobileEntity$ may appear. But, if $Android < ToyRobot$, then no concept node labeled $(Android, R2)$

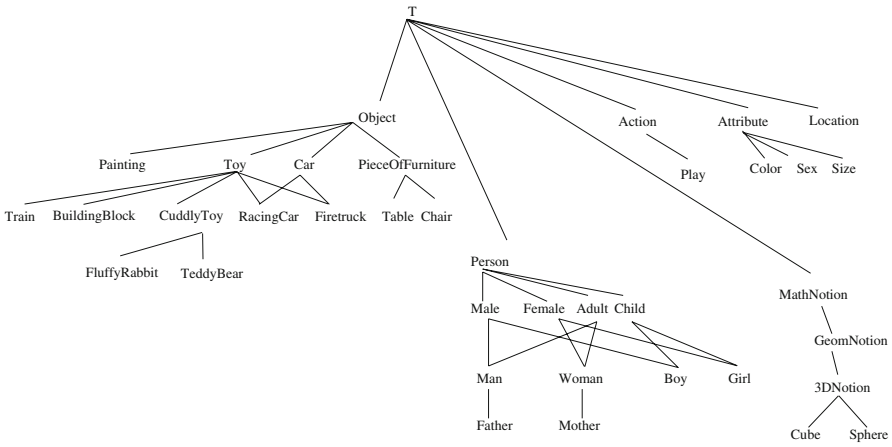

Fig. 2.1 A concept type set

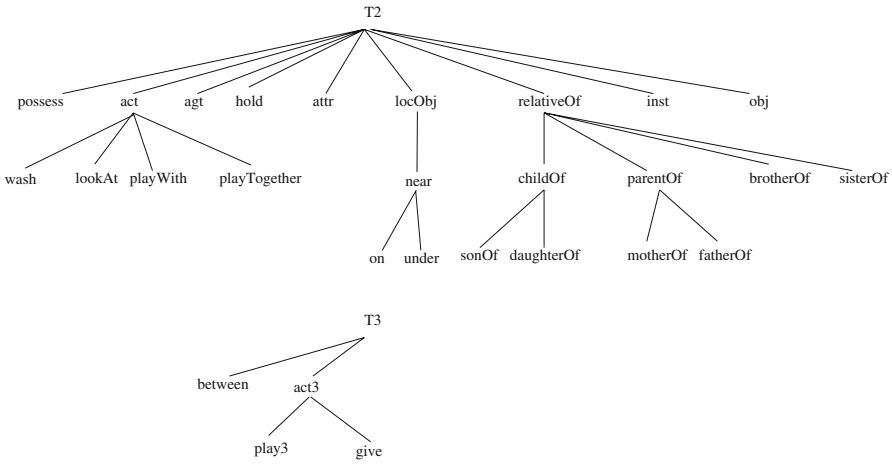

Fig. 2.2 A relation symbol set

can appear because $Android < \tau(R2)$. Typing the individuals may be a drawback whenever knowledge about the individuals may evolve. This is typically the case in a knowledge acquisition context. For instance, $R2$ may first be known as a *Toy* and later as a *ToyRobot*.

The second extension is the introduction of relation symbol signatures. A relation symbol signature specifies not only the arity of the relation symbol but also the maximal concept type of each of its arguments. For instance, the signature of the relation symbol *possess* could be $(Person, Object)$, which means that the first argument of a relation *possess* is of type *Person* and the second argument is of type $Object$. Relation signatures are formally defined by a mapping σ, which to every relation symbol $r \in T_R^j$, $1 \leq j \leq k$ associates $\sigma(r) \in (T_C)^j$; this mapping also has to

fulfill the following condition: $\forall r_1, r_2 \in T_R^j$, $r_1 \leq r_2 \Rightarrow \sigma(r_1) \leq \sigma(r_2)$, i.e., for all $1 \leq i \leq j$, the i-th argument of $\sigma(r_1)$ is less than or equal to the i-th argument of $\sigma(r_2)$. That is, when a relation symbol r_2 is specialized into a relation symbol r_1, its arguments can be specialized but cannot be generalized.

Example. For instance, let us consider the relation symbols $\top_2$, *relativeOf*, *parentOf*, *motherOf*. Signatures for these relations can be: $\top_2(\top, \top)$, *relativeOf(Person, Person)*, *parentOf(Person, Person)*, *motherOf(Mother, Person)*. If the ternary relation *play3* is intended to mean that two persons are playing together with a toy then its signature could be *play3(Person, Person, Toy)*. In the same way, the signature of the ternary relation *give* could be *give(Person, Person, Object)* with the following intuitive meaning: The first person is giving the object to the second person.

In the forthcoming chapters, more complex vocabularies and more complex knowledge, such as rules or constraints, will be studied. Rules and constraints allow the representation, in the conceptual graph model, of "heavyweight" formal ontologies.

2.1.2 Basic Conceptual Graphs

A *basic conceptual graph (BG)* is a bipartite multigraph (cf. Chap. A). "Bipartite" means that a BG has two disjoint sets of nodes and that any edge joins two nodes from different sets (i.e., an edge cannot join two nodes of the same set). "Multigraph" means that a pair of nodes may be linked by several edges. One set of nodes is called the set of concept nodes (representing entities), and the other set is called the set of relation nodes (representing relations between entities). If a concept c is the i-th argument of a relation r then there is an edge between r and c that is labeled i.

Example. Figure 2.3 shows a BG consisting of four concept nodes (the individual *Paul* who is a *Child*, a *Car*, a *Person* and the individual *Small* which is a *Size*) and three relations: one ternary relation, *play*3, whose neighbor list is ([Child: Paul], [Person], [Car]), and two binary relations, *attr* and *possess*, whose neighbor lists are respectively ([Child: Paul], [Car]) and ([Car], [Size: Small]). This graph can be considered as representing the following fact: "There is a car, which is small. This car is possessed by Paul, who is a child. There is a person, and Paul and this person are playing with that car." This way of describing the fact puts the emphasis on the car, but the emphasis could have been put on Paul as the drawing suggests it (e.g., "Paul, who is a child, and a person, are playing with a small car that belongs to Paul") or any subset of entities. A basic conceptual graph has no privileged nodes. Figure 2.4 shows another BG that could represent the assertion that "Paul, who is a child, is washing himself, and he is playing with his mother." Note the parallel edges between the concept [Child:Paul] and the relation (wash), indicating that the agent and the object of the relation (wash) are the same entity, the child Paul. Figure 2.5 shows a BG with more complex cycles, asserting that "a father and his child are

playing together on a mat; the mother of the child is looking at them; she is on the sofa near the mat."

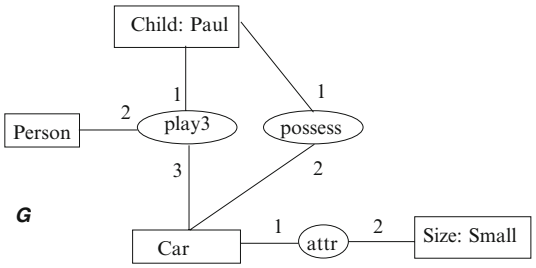

Fig. 2.3 A BG with a ternary relation

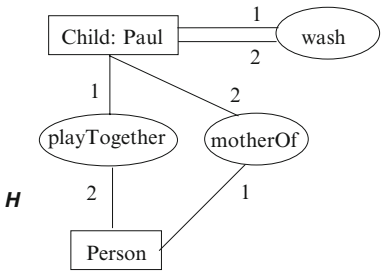

Fig. 2.4 A BG with parallel edges

Every node has a label. A relation node is labeled by a relation symbol and a concept node is labeled by a concept type and by a marker. Thus, a basic conceptual graph is built relative to a vocabulary, and it has to satisfy the constraints enforced by that vocabulary.

Definition 2.2 (Basic Conceptual Graph). A *basic conceptual graph* (BG) defined over a vocabulary $\mathcal{V} = (T_C, T_R, \mathcal{I})$, is a 4-tuple $G = (C, R, E, l)$ satisfying the following conditions:

- (C, R, E) is a finite, undirected and bipartite multigraph called the *underlying graph* of G, denoted $graph(G)$. C is the *concept node* set, R is the *relation node* set (the node set of G is $N = C \cup R$). E is the family of *edges*.
- l is a labeling function of the nodes and edges of $graph(G)$ that satisfies:
  1. A concept node c is labeled by a pair *(type(c), marker(c))*, where *type(c)*$\in T_C$ and *marker(c)*$\in \mathcal{I} \cup \{*\}$,
  2. A relation node r is labeled by $l(r) \in T_R$. $l(r)$ is also called the type of r and is denoted by *type(r)*,

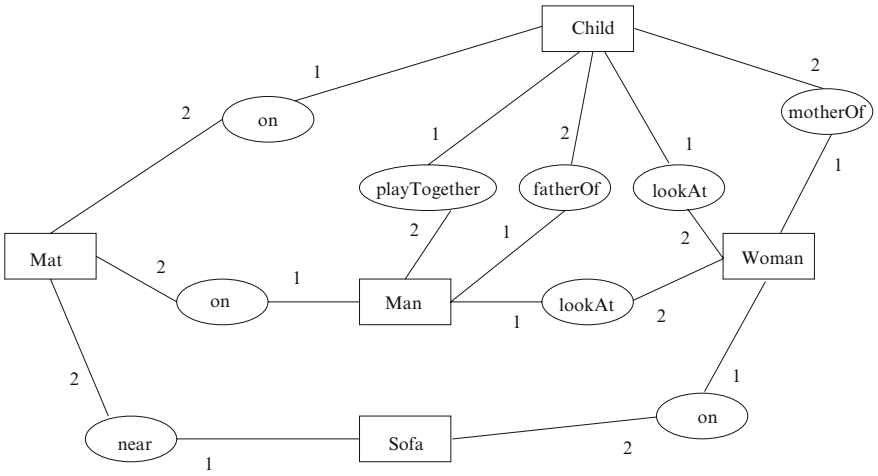

Fig. 2.5 A BG with more complex cycles

3. The degree of a relation node r is equal to the arity of type(r),
4. Edges incident to a relation node r are totally ordered and they are labeled from 1 to $arity(type(r))$.

First note that a BG does not need to be a *connected* graph. It is natural to question what the smallest BGs are. The preceding definition does not prevent a BG from being *empty*, that is to be the tuple $(\emptyset, \emptyset, \emptyset, \emptyset)$. For theoretical purposes, it is sometimes convenient to consider the empty BG. We will denote it by $G_\emptyset$. A BG may be restricted to a single concept node or several concept nodes. We will call them *isolated* concept nodes. But as soon as a BG contains a relation node, it contains at least one concept node, since there are no 0-ary relation symbols. Note also that, as there may be parallel edges, one has to distinguish between the number of edges incident to a relation node (this number is given by the arity of its type) and the number of its neighbors. For instance, the relation (wash) in Fig. 2.4 is incident to two edges but has only one neighbor.

An important kind of BGs consists of BGs having a single relation node.

Definition 2.3 (Star BG). A *star BG* is a BG restricted to a relation node and its neighbors.

With BGs restricted to single concept nodes, star BGs are the elementary building blocks of BGs. This point will be specified in Chap. 8, but let us outline the idea. First, the set of relation symbols of a BG vocabulary can be described by a set of star BGs: A relation symbol r of signature $(t_1, \ldots, t_k)$ is represented by a star BG, the relation node is labeled r and has k neighbors with the i-th being labeled $(t_i, *)$ (cf. Fig. 2.6). Then every non-empty BG definable on this vocabulary can be generated from this set of star graphs by the so-called specialization operations (if the BG has isolated concept nodes, one has to add, to the set of star graphs, the graph restricted to the single concept node $[\top : *]$).

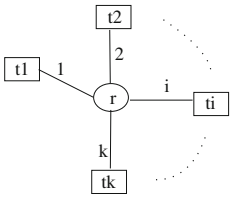

Fig. 2.6 A star BG associated with the relation symbol r with signature $(t_1,\ldots,t_k)$

Notations and Remarks

An edge labeled i between a relation r and a concept c is denoted (r,i,c). The i-th argument of a relation r is also called the i-th neighbor of r and is denoted $r[i]$ i.e., (r,i,c) is in G if and only if $r[i]=c$ in G. The neighbor list $(c_1,\ldots,c_k)$ of a relation r of arity k is the list such that $c_i = r[i]$. It is said that $r(c_1,\ldots,c_k)$ is in G if $(c_1,\ldots,c_k)$ is the neighbor list of the relation r. The same concept node may appear several times in the neighbor list of a relation, so strictly speaking a BG is a multigraph and not a graph.

We will adopt the following classical conventions for BGs. A relation symbol is also called a *relation type* even if a concept type and a relation type have different meanings. A concept type can be considered as a class, i.e., the class of all the entities having this type, but a relation type does not represent a class. Nevertheless, calling "relation type" a relation symbol allows us to simplify notations by considering the type of any node, either concept or relation, in a BG. In a BG drawing, concept nodes are represented by rectangles and relation nodes by ovals. In textual notation, rectangles are replaced by [] and ovals by (). The generic marker is the marker by default, and it is generally omitted. Thus a generic concept of type t, whose label is $(t,*)$, is simply noted [t]. An individual concept with label (t,m) is noted [t:m].

For binary relation nodes, numbers on edges can be replaced by directed edges. A relation node is then incident to exactly one incoming edge and one outgoing edge, linking it to its first and second neighbor, respectively. For example, Fig. 2.7 pictures the same BG as in Fig. 2.3, with arrows on edges incident to binary relations; the arrow from [Car] to (attr) stands for an edge labeled 1, and the arrow from (attr) to [Size : Small] stands for an edge labeled 2.

Relation labels are naturally ordered by the partial order on T_R, and concept labels are ordered as follows:

Definition 2.4 (Ordered set of concept labels). The set of *concept labels*, defined over a given vocabulary, is the set of couples (t,m) such that $t \in T_C$ and $m \in \mathcal{M}$. This set is the cartesian product of the two partially ordered sets T_C and $\mathcal{M}$, thus it is (partially) ordered by $(t,m) \leq (t',m')$ if and only if $t \leq t'$ and $m \leq m'$.

Example. $(Boy,Paul) \leq (Person,Paul) \leq (Person,*)$, but $(Boy,*)$ and $(Person,Paul)$ are not comparable because $Boy < Person$ and $Paul < *$.

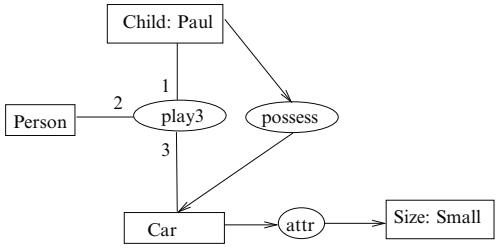

Fig. 2.7 Another drawing of G (cf. Fig. 2.3)

2.1.3 SubBGs and PseudoBGs

We are interested here in subgraphs of a BG that are themselves BGs. We call them *subBGs*. Let us begin with their formal definition before discussing the distinction between a "subgraph of a BG" and a "subBG."

Definition 2.5 (subBG). Let $G = (C, R, E, l)$ be a BG. A *subBG* of G is a BG $G' = (C', R', E', l')$, such that:

- $C' \subseteq C$, $R' \subseteq R$, $E' \subseteq E$,
- l' is the restriction of l to $C' \cup R' \cup E'$.

G' is a *strict* subBG of G if the number of nodes in G' is strictly less than the number of nodes of G.

In graph theory, a subgraph of a graph G is defined as a graph obtained from G by removing nodes (and edges incident to these nodes). A subBG must be a BG, thus it is not possible to remove just any nodes. Since the degree of a relation node has to be equal to the arity of its type, deleting a concept node of G implies deleting all neighboring relation nodes. Similarly, a graph obtained from a BG by removing only edges is not a BG; that is if an edge is removed then its incident relation must be removed too.

Example. G' in Fig. 2.8 is a subgraph of G in Fig. 2.3. G' is not a BG because in G' the relation (play3), which is a ternary relation, has only two neighbors. Thus G' is not a subBG of G.

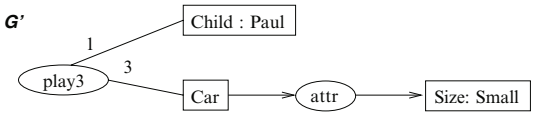

Fig. 2.8 G': a subgraph of G (cf. Fig. 2.4) which is not a subBG

A subBG of G can be obtained from G only by repeatedly deleting a relation or an isolated concept.

Property 2.1. A subgraph G' of a BG G is a subBG if and only if all neighbors in G of any relation r in G' are also neighbors of r in G'.

Definition 2.6 (Boundary node). A *boundary node* of a subBG G' of G is a node of G' that has a neighbor outside G'.

As the arity of a relation type is a constant, a boundary node is necessarily a concept.

If edges or nodes are arbitrarily deleted from a BG, the obtained graph is called a *pseudo-BG*. More generally, a pseudo-BG has concept and relation nodes, and edges connecting concepts to relations, but it might not satisfy other conditions on a BG.

Definition 2.7 (Pseudo-BG). A *pseudo-BG* is obtained from the definition of a BG by removing some conditions from the set of conditions 2.

Example. G' in Fig. 2.8 is a pseudo-BG.

Pseudo-BGs naturally occur during the construction of BGs, using a graph editor for instance. Indeed, the graphs obtained before the whole completion generally are only pseudo-BGs and not BGs. More generally, during knowledge acquisition, any constraint of the model can be a drawback. Pseudo-BGs represent minimal constraints that in our opinion have to be enforced: having two kinds of nodes and preventing edges to connect nodes of the same kind. However, after completion of a knowledge acquisition step, the constraints of the vocabulary (possibly including individual types or relation signatures) are useful validation tools.

2.2 BG Homomorphism

2.2.1 Subsumption and Homomorphism

The *subsumption* relation (denoted $\succeq$) is the fundamental notion for reasoning with BGs. Let G and H be two BGs over the same vocabulary. Intuitively, G subsumes H (noted $G \succeq H$) if the fact—or the information—represented by H entails the fact represented by G, or in other words, if all information contained in G is also contained in H. "G subsumes H" is equivalent to "H is subsumed by G" denoted by $H \preceq G$. Let us keep this intuitive meaning for now; we will come back to the semantic of subsumption later. This subsumption relation can be defined either by a sequence of elementary operations or by the classical homomorphism notion applied to BGs (which essentially is a graph homomorphism, with this point being studied in Chap. 5).

Definition 2.8 (BG homomorphism). Let G and H be two BGs defined over the same vocabulary. A *homomorphism* π from G to H is a mapping from C_G to C_H and from R_G to R_H, which preserves edges and may decrease concept and relation labels, that is:

- $\forall (r,i,c) \in G, (\pi(r), i, \pi(c)) \in H$,
- $\forall e \in C_G \cup R_G, l_H(\pi(e)) \leq l_G(e)$.

If there is a homomorphism (say π) from G to H, we say that G *maps* or *projects* to H (by π).

Example. Figure 2.9 G maps to H and K.

Remark

In the conceptual graph community a BG homomorphism is traditionally called a *projection*. We prefer the term BG homomorphism for two reasons: First, it corresponds to the use of "homomorphism" in mathematics, indeed a BG homomorphism is a mapping between BGs preserving the BG structure; secondly, in the relational database model, there is an operation called "projection," which is rather different from a BG homomorphism (cf. Chap. A).

Definition 2.9 (Subsumption relation). Let G and H be two BGs defined over the same vocabulary. The *subsumption* relation $\succeq$ is defined by: $G \succeq H$ if there is a homomorphism from G to H.

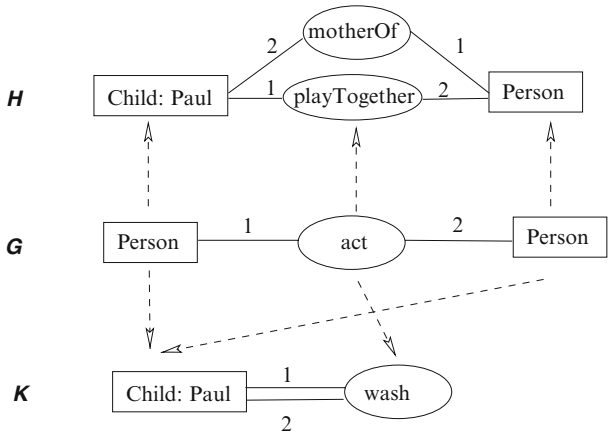

Fig. 2.9 Homomorphisms from G to H and K

Example. Let us consider the BGs G and H in Fig. 2.10. There are two homomorphisms from G to H, namely h_1 and h_2, pictured by dashed and dotted lines, respectively. By h_2, [Child] in G is mapped to [Child:Paul] in H, by h_1 it is mapped to [Child]. The relation (act) is mapped to the appropriate relation (playWith). The other nodes have the same image by both homomorphisms: [Toy] and [Car] are mapped to [FireTruck], the two relations (on) are mapped to the sole relation (on) in H, and each remaining node is mapped to the node with same label in H.

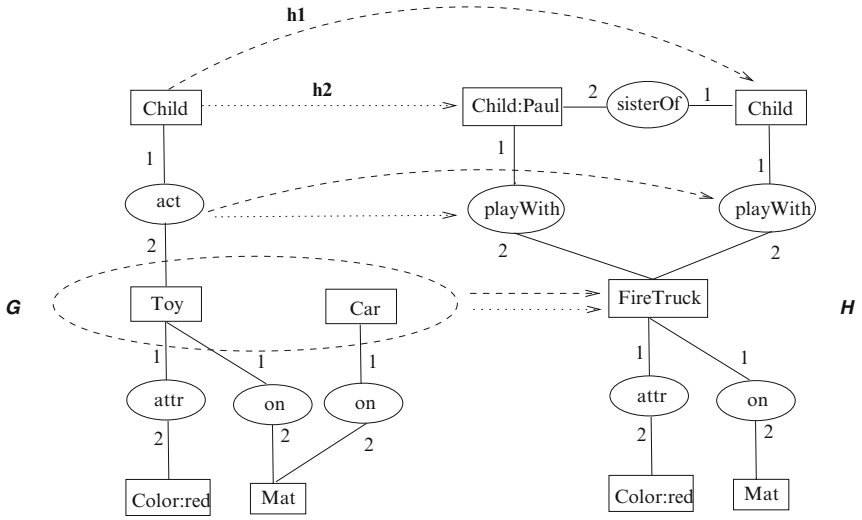

Fig. 2.10 $G \succeq H$

Subsumption emphasizes the fact that a BG is always relative to a given vocabulary and has no meaning independently from it. In other words, the same labeled graph satisfying the properties of Definition 2.2 leads to two different BGs if it is considered relative to two different vocabularies. For instance, let us consider the BG G in Fig. 2.10, which is relative to the vocabulary $\mathcal{V}$ given in Fig. 2.1. If $\mathcal{V}'$ is obtained from $\mathcal{V}$ by deleting the fact that $Firetruck < Car$, then there is no homomorphism from G to H considered as BGs over $\mathcal{V}'$, whereas there is a homomorphism from G to H when they are BGs over $\mathcal{V}$.

Definition 2.10 (Image by homomorphism). Let $G = (C_G, R_G, E_G, l_G)$ and $H = (C_H, R_H, E_H, l_H)$ be two BGs and let π be a homomorphism from G to H. The image of G by π is $\pi(G) = (\pi(C_G), \pi(R_G), E', l')$, with E' being equal to the edges of E_H with one extremity in $\pi(C_G)$ and the other in $\pi(R_G)$, and l' being the restriction of l_H. In other words, $\pi(G)$ is the subgraph of H induced by $\pi(C_G) \cup \pi(R_G)$.

It is straightforward to check:

Property 2.2. If G and H are two BGs and π is a homomorphism from G to H, then $\pi(G)$ is a subBG of H.

Let us now observe what happens for homomorphism when BGs have several individual concepts with the same marker. Let us consider the two BGs in Fig. 2.11. There is clearly a homomorphism from G to H but there is no homomorphism in the other direction, i.e., from H to G, even if the two BGs have the same intuitive meaning that is: *The individual m has properties r and s*. We will discuss this point in the section about normal BGs (cf. Sect. 2.5).

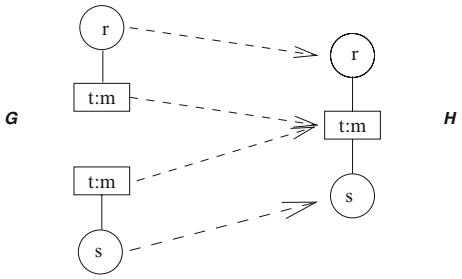

Fig. 2.11 $G \succeq H$ and $H \not\succeq G$

2.2.2 Bijective Homomorphisms and Isomorphisms

The notion of BG isomorphism is naturally defined as follows:

Definition 2.11 (BG isomorphism). An isomorphism from a BG G to a BG H is a bijection π from C_G to C_H and from R_G to R_H, which satisfies:

- $\forall (r,i,c) \in G, (\pi(r),i,\pi(c)) \in H$, and reciprocally, $\forall (r',i,c') \in H$, $(\pi^{-1}(r'),i,\pi^{-1}(c')) \in G$,
- $\forall e \in C_G \cup R_G, l_G(e) = l_H(\pi(e))$.

A bijective BG homomorphism from G to H is an isomorphism from $graph(G)$ to $graph(H)$, but it is a BG isomorphism only if it furthermore preserves labels (i.e., it fullfills condition 2 of the BG isomorphism definition). Indeed, since a relation is mapped to a relation of the same arity, a bijective homomorphism π from G to H always fullfils the property that for all r' and c' in H, for all c and r in G, such that $c' = \pi(c)$ and $r' = \pi(r)$, $(\pi(r),i,\pi(c)) \in H$ implies $(r,i,c) \in G$ (and reciprocally by the homomorphism definition). Thus, the first condition in the BG isomorphism definition is satisfied by a homomorphism as soon as it is bijective.

Examples of bijective BG homomorphisms that are not BG isomorphisms naturally occur when BGs represent data forms to fill in.

A very simple data form schema is represented by DF in Fig. 2.12; F is a partially filled data form corresponding to DF. It is simple to see that the mapping represented by dashed arrows is a bijective homomorphism from DF to F.

Property 2.3. For any BG G, a bijective homomorphism from G to itself is an isomorphism.

Proof. The sets $T_C \times \mathcal{M}$ of concept labels and T_R of relation symbols are partially ordered.

Let us consider two linear extensions of the dual orders of these partially ordered sets, both denoted $\leq_w$, i.e., given two labels l_1 and l_2, $\leq_w$ is a total order and if $l_1 \geq l_2$, then $l_1 \leq_w l_2$.

Assign a weight to each concept and relation, which is the position of its label in $\geq_w$ (starting from 1). It becomes heavier as it becomes more specialized. Then,

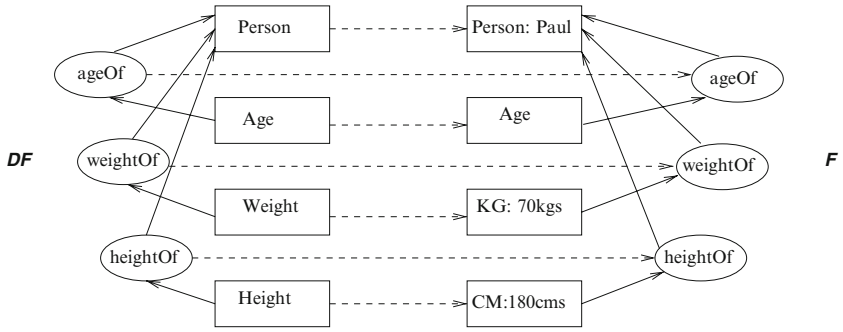

Fig. 2.12 Data form and bijective homomorphism

to a BG, assign a weight (let us denote it w) which is the sum of its relation and concept weights. Note that if there is an injective homomorphism from a BG G_1 to a BG G_2, then $w(G_1) \leq w(G_2)$. Now let π be a bijective homomorphism from G to itself. Assume that π strictly restricts the label of at least one relation or concept of G. Then $w(G) < w(\pi(G))$ which is impossible since $\pi(G) = G$. Thus π is an isomorphism. □

2.2.3 BG Queries and Answers

With BGs and subsumption, we can build a basic query-answering mechanism. Let us consider a KB (knowledge base) B composed of a set of BGs, representing some assertions about a modeled world, e.g., a set of BGs representing facts about the children photo example (Sect. 1.1.1). A *query* made to this base is itself a BG, say Q. Elements answering Q are intuitively defined as the elements in B that entail Q, or equivalently, elements that are specializations of Q, or also, elements that are subsumed by Q.

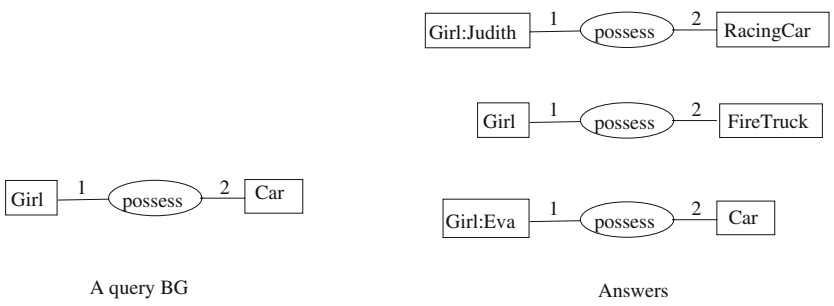

Fig. 2.13 A query and possible answers

More precisely, a query, hence the notion of answer, can be interpreted in different ways:

- A query Q can be seen as a representation of a "yes/no question": *"Is the knowledge represented by Q asserted by the KB?"* For instance, the query in Fig. 2.13 may represent the question, "Is there a girl who owns a car?" or, "Is there a car owned by a girl?" Note that, as a BG does not need to be a connected graph, B can be seen as composing a single BG. The answer is boolean, which is true if and only if Q subsumes B.
- A query Q can be seen as a "pattern" allowing us to extract knowledge from the KB. Generic nodes in the query represent variables to instantiate with individual or generic nodes in the base. The query Q in Fig. 2.13 would become "find all patterns in which a girl owns a car." With this interpretation, each homomorphism from Q to B defines an answer to Q. An answer can be seen as the homomorphism itself, which assigns a node of the base to each node of the query. Or it can be seen as the subgraph of B induced by this homomorphism, i.e., the image of Q (see Fig.2.13 and also Fig. 2.12, where, assuming that the data form DF is used as a query, F can be seen as an answer to this query).

Note that distinct homomorphisms from Q to B may produce the same image graph; thus defining answers as image graphs instead of homomorphisms induces a potential loss of information. In turn, an advantage of considering image graphs is that the set of answers can be seen as a BG. We thus have the property that the results returned by a query are in the same form as the original data. This property would be mandatory if we were interested in processing complex queries, i.e., queries composed of simpler queries; in this context, the answers to a query would be seen as a knowledge base, which could be used to process another query.

A simple extension of this basic mechanism is classically considered. A query is not simply a BG but a BG with distinguished concept nodes defining the part to be considered to define an answer. These nodes are usually distinguished by a question mark (?) added to their marker. Then an answer is given by the part of the homomorphism having for domain the set of marked nodes, or by the associated image graph. With the example in Fig. 2.13, marking the node [Girl] would lead to the query "find all girls who own a car" and, if answers are given as image graphs, would return the graphs [Girl:Judith], [Girl] and [Girl:Eva]. We then have a mechanism equivalent to conjunctive queries in databases, see Chap. 5.

2.3 BG Subsumption Properties

2.3.1 Subsumption Preorder

Properties concerning homomorphisms can be directly proven from the definitions given above. For instance, it is simple to check the following property:

Property 2.4 (Composition of homomorphisms). The composition of two homomorphisms is a homomorphism; thus the subsumption relation $\succeq$ is transitive.

This property is also a direct corollary of two known properties: First, the composition of two graph homomorphisms is a graph homomorphism (the structure is preserved). Secondly, order relations are transitive relations (the composition of two decreases of labels is a decrease of labels).

$\succeq$ is also a *reflexive* relation since every BG maps to itself with the identity homomorphism. But $\succeq$ is not *antisymmetric*: Indeed, there are non-isomorphic BGs that map to each other. For instance, the BGs G and H in Fig. 2.14 are non-isomorphic whereas $G \succeq H$ and $H \succeq G$ (there is a homomorphism from H to G, which maps both concept nodes with label l_1 to the same node in G). Figure 2.15 presents a more complex example: Any BG in the figure subsumes any other.

Definition 2.12 (Hom-equivalence). Two BGs G and H are said to be *hom-equivalent* if $G \succeq H$ and $H \succeq G$, also denoted $G \equiv H$.

If $G \succeq H$ and G and H are not hom-equivalent, we say that G is strictly more general than H and note $G \succ H$ (or $H \prec G$).

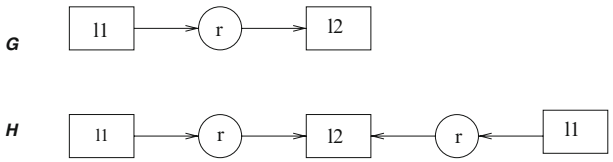

Fig. 2.14 Two hom-equivalent BGs

An injective homomorphism is a special case of a homomorphism enforcing the fact that distinct nodes necessarily have distinct images. Let $G \succeq_i H$ denote the fact that there is an *injective* homomorphism from G to H. In Fig. 2.14 one has $G \succeq_i H$ but not $H \succeq_i G$. It is simple to check that this relation is transitive, reflexive and antisymmetric; thus $\succeq_i$ is an order relation.

Property 2.5. $\succeq$ is a preorder on the BGs defined over $\mathcal{V}$ (and it is not an order). $\succeq_i$ is an order on the BGs defined over $\mathcal{V}$.

Let $G \succeq_i H$, then there is a bijective homomorphism from G to a subBG of H.
If the injectivity of a homomorphism concerns only the concept nodes, the induced binary relation is not an order as shown in Fig. 2.16, where $r \leq s$ (however, in this case, checking "redundancy" becomes simple).

From a knowledge viewpoint, considering $\succeq_i$ instead of $\succeq$ can be pertinent whenever *two different concept nodes always represent two different entities*. From a complexity viewpoint, the decision problems "Is there a homomorphism from G to H," "Is there an injective homomorphism from G to H?" and "is there a bijective homomorphism from G to H?" are all NP-complete (cf. Sect. 2.6). Nevertheless, the two relations $\succeq$ and $\succeq_i$ behave differently from a complexity viewpoint (for instance,

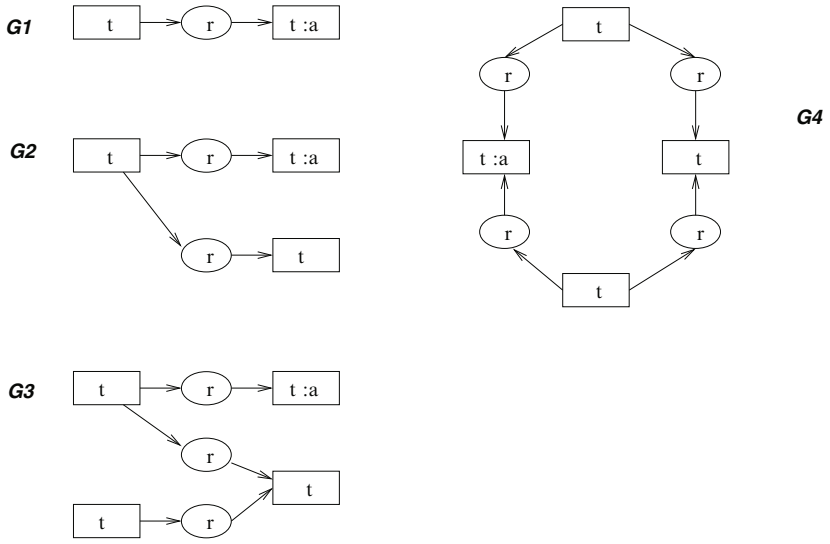

Fig. 2.15 Four hom-equivalent BGs

Fig. 2.16 Injectivity restricted to concept nodes ($r \leq s$)

given two BGs T and H, such that T is acyclic, checking whether $T \succeq H$ is a polynomial problem—see Chap. 7—whereas checking $T \succeq_i H$ remains NP-complete) and, contrary to homomorphism, injective homomorphism is not a complete operation with respect to the formal semantics of BGs studied in Chap. 4.

2.3.2 Irredundant BGs

In this section the equivalence relation associated with the subsumption preorder is studied. We show that any equivalence class has a distinct representative which is the "simplest" BG of the class and that this element is unique (up to isomorphism).

Definition 2.13 (Irredundant and redundant). A BG is called *redundant* if it is hom-equivalent to one of its strict subgraphs. Otherwise it is called *irredundant*.

Let G be a redundant BG. Let us consider a strict subBG H hom-equivalent to G and having a minimum number of nodes. H is irredundant. Thus,

Property 2.6. If a BG G is redundant, then it has an irredundant strict subBG hom-equivalent to it.

Let G be a BG and H a subBG of G. There is a trivial homomorphism from H to G, thus a BG is irredundant if and only if there is no homomorphism from it to one of its strict subBGs. In other words, there is no non-injective (or equivalently there is no non-surjective) homomorphism from G to itself. Equivalently, every homomorphism from G to itself is a bijective homomorphism. We have proven that any such homomorphism is in fact an isomorphism (Property 2.3). Thus we have:

Property 2.7. A BG G is irredundant if and only if every homomorphism from G to itself is an isomorphism (automorphism).

Example. In Fig. 2.14, G is irredundant while H is not; the BGs in Fig. 2.15 are all hom-equivalent, and G_1 is the only irredundant BG.

We now prove that a class of hom-equivalent BGs contains a unique irredundant BG. Thus this irredundant BG can be taken as the representative of the equivalence class. For this we use the following property.

Property 2.8. Let G be any BG and H be an irredundant BG. If G is hom-equivalent to H, then G has a subBG isomorphic to H.

Proof. If G and H are hom-equivalent, there is a homomorphism, say π, from H to G and a homomorphism, say π', from G to H. Now consider the homomorphism $\pi' \circ \pi$ from H to H. $\pi' \circ \pi$ is an isomorphism (by Property 2.7).

Let $G' = \pi(H)$. Let us show that π is an isomorphism from H to G'. π is surjective by definition of G', and π is injective since $\pi' \circ \pi$ is injective. Thus π is a bijective homomorphism from H to G'.

Now, for all concept or relation x in H, $label(x) \geq label(\pi(x)) \geq label(\pi' \circ \pi(x)) = label(x)$ thus $label(x) = label(\pi(x))$. π is thus an isomorphism from H to G', which proves the property. □

Theorem 2.1. *Each hom-equivalence class contains a unique (up to isomorphism) irredundant BG which is the BG having the smallest number of nodes.*

Proof. Property 2.8 implies that two irredundant BGs are hom-equivalent if and only if they are isomorphic. Thus an equivalence class contains at most one irredundant BG. Now, given an equivalence class, let us take a BG, say G, of minimal size (with the size being the number of concepts and relations). G is irredundant, otherwise it would be hom-equivalent to one of its strict subBGs, which contradicts the hypothesis on G. □

Example. In Fig. 2.15, each BG contains an irredundant subgraph isomorphic to G_1, to which it is hom-equivalent.

G may contain several irredundant subBGs hom-equivalent to it, but in this case they are all isomorphic to each other. Any of these subBGs can be taken as the *irredundant form* of G. The following property gives more insight into the relationships between a BG and its irredundant form: A redundant BG can be *folded* to its irredundant form.

Definition 2.14 (Folding). Let G be a BG, let π be a homomorphism from G to itself and let $G' = \pi(G)$. π is called a *folding* from G to G' if each node of G' is invariant by π, i.e. $\forall x \in C_{G'} \cup R_{G'}, \pi(x) = x$.

Property 2.9. Let G be a BG and let G' be one of its hom-equivalent irredundant subBGs. Then there is a folding from G to G'.

Proof. By hypothesis there is a homomorphism, say π, from G to G'. Let π' be the homomorphism obtained from π by restricting its domain to G': π' is an isomorphism (from Corollary 2.7). Then $\pi'' = \pi' \circ \pi$ is also a homomorphism from G to G', which is the identity on G'. π'' is thus a folding. □

The folding notion is a direct adaptation for BGs of a classical notion in graph theory. Note that G' being irredundant is essential to this property: A BG cannot always be folded into any of its hom-equivalent subBGs.

If a BG G has duplicate relations, then it is redundant since the subgraph obtained by deleting a duplicate relation is hom-equivalent to it. More generally, if G has two relations with the same neighbors in the same order, such that one relation has a type greater or equal to the type of the other, then the relation with the more general type is redundant. In the absence of redundant relations, G is redundant if and only if there is a homomorphism from G to itself that maps two concepts to the same concept, say c_1 and c_2 to c_2; in this case there is a homomorphism from G to the subBG obtained by deleting c_1 and its neighboring relations. Finally, let us point out that rendering a BG irredundant is generally not a simple task. Indeed, the associated decision problem, namely "Given a BG G, is G redundant?", is an NP-complete problem, as we shall see in Sect. 2.6.

Considering irredundant BGs only, the subsumption relation becomes a special partial order—it is a lattice.

Property 2.10. Let $\mathcal{G}$ be the set of all irredundant BGs definable over a given vocabulary. Then $(\mathcal{G}, \preceq)$ is a lattice.

Proof. The set of BGs over a given vocabulary admits a greatest element: the empty BG. Should we exclude the empty BG, there is still a greatest element, the BG restricted to one generic concept with universal type. Now let us consider G and H two (irredundant) BGs and their disjoint sum $G+H$. G and H subsume $G+H$. And every BG K which is subsumed both by G and H is also subsumed by $G+H$ (the union of two homomorphisms from G to K and from H to K defines a homomorphism from $G+H$ to K). Thus the irredundant BG hom-equivalent to $G+H$ is the greatest lower bound of G and H. The set of irredundant BGs is thus an inf-semi-lattice, and since it possesses a greatest element, it is a lattice. □

Note that the previous property is true even if the set of concept node labels is not a lattice. The only property required, if the empty BG is not admitted, is that the concept type set has a greatest element. There is no condition on the relation symbol set.

Example. In Fig. 2.17, the vocabulary given on the left is restricted to a concept type set, and it should be noted that this set is not a lattice. The lattice on the right

represents all irredundant BGs constructible on this vocabulary, except for the empty BG.

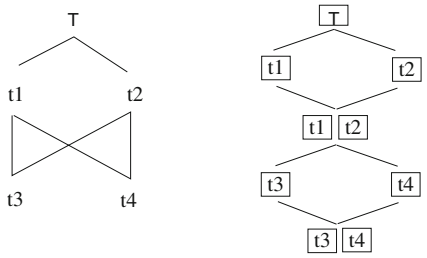

Fig. 2.17 A lattice of irredundant graphs

2.4 Generalization and Specialization Operations

Transforming a BG into its image by a homomorphism is a global operation between BGs that can be decomposed into simpler operations. Sets of elementary operations "equivalent" to homomorphism—in the sense that there is a homomorphism from G to H if and only there is a sequence of elementary operations transforming G into H—are presented in this section. Some of these elementary operations are frequently used in applications (e.g., the join in natural language processing). More complex operations can be defined using these elementary operations (e.g., extended join and maximal join defined in Chap. 8). Finally, these elementary operations are used in order to inductively define BGs (cf. Chap. 8).

There are five elementary generalization operations and five inverse operations, called elementary specialization operations.

2.4.1 Elementary Generalization Operations for BGs

Any generalization operation is a "unary" operation, i.e it has a BG (and some elements of it) as input and has a BG as output.

Definition 2.15 (Generalization operations). The five elementary generalization operations are:

- **Copy.** Create a disjoint copy of a BG G. More precisely, given a BG G, $copy(G)$ is a BG which is disjoint from G and isomorphic to G.
- **Relation duplicate**. Given a BG G and a relation r of G, $relationDuplicate(G,r)$ is a BG obtained from G by adding a new relation node r' having the same type

and the same list of arguments as r, i.e., the same neighbors in the same order. Two such relations of the same type and having exactly the same neighbors in the same order are called *twin relations*.

- **Increase.** Increase the label of a node (concept or relation). More precisely, given a BG G, a node x of G, and a label $L \geq l(x)$ $increase(G,x,L)$ is the BG obtained from G by increasing the label of x up to L, that is its type if x is a relation, its type and/or its marker if x is a concept.
- **Detach.** Split a concept into two concepts. More precisely, let c be a concept node of G and $\{A_1,A_2\}$ be a partition of the edges incident to c, $detach(G,c,A_1, A_2)$ is the BG obtained from G by deleting c, creating two new concept nodes c_1 and c_2, having the same label as c, and by attaching A_1 to c_1, and A_2 to c_2 (A_1 or A_2 may be empty).
- **Substract.** Given a BG G, and a set of connected components $C_1,\ldots,C_k$ of G, $substract(G,C_1,\ldots,C_k)$ is the BG obtained from G by deleting $C_1,\ldots,C_k$ (the result may be the empty graph).

Remark

Copy and relation duplicate are actually equivalence operations, i.e., they do not change the semantics of the graph (cf. Chap. 4).

The *copy* operation is in fact not needed in generalization operations since a copy of a graph can be obtained by a Relation Duplicate sequence followed by a Detach sequence ended by a Substract. It is considered as a generalization operation for symmetry reasons with specialization operations.

The copy operation is also needed because, in some situations, we do not consider BGs up to isomorphism. For instance, in implementation and drawings two copies of the same BG must be considered as two different BGs.

The substract operation is defined as the deletion of a set of connected components, and not the deletion of only one component, in order to be the inverse of the disjoint sum, which is an elementary specialization defined later.

Example. The *relation duplicate* operation is illustrated in Fig. 2.18. The *detach* operation is illustrated in Fig. 2.19.

It is now possible to precisely define what *a BG G is a generalization of a BG H* means:

Definition 2.16 (Generalization). A BG G is a generalization of a BG H if there is a sequence of BGs $G_0 = (H), G_1, ..., G_n = (G)$, and, for all $i = 1,\ldots,n$, G_i is obtained from G_{i-1} by a generalization operation.

Example.

Figure 2.20 presents some of the graphs occurring in a generalization sequence from H to G. H_1 is obtained from H by splitting the nodes [Child:Paul] and [FireTruck] (Detach operation), then deleting the connected component K pictured in the dashed rectangle (Substract operation). H_2 is obtained from H_1 by duplicating the (on) relation (Relation Duplicate operation). H_3 is obtained from H_2 by splitting

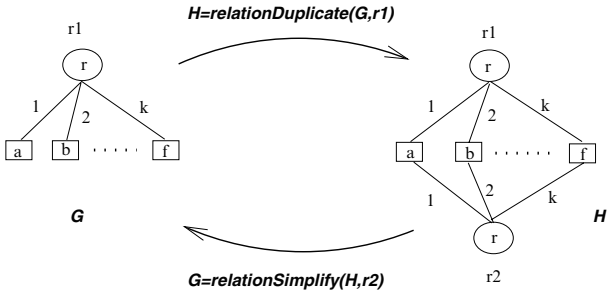

Fig. 2.18 Relation duplicate and its inverse Relation simplify

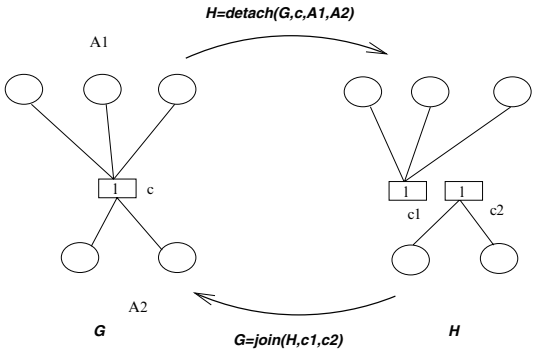

Fig. 2.19 Detach and its inverse Join

the node [FireTruck] (Detach operation). Finally, H_4 isomorphic to G is obtained by increasing some labels (Increase operation). This generalization sequence can be associated with the homomorphism h_1 in Fig. 2.10. The equivalence between a homomorphism and a generalization sequence is stated in next properties (Property 2.12 and Property 2.13).

Property 2.11. A subBG G' of a BG G is a generalization of G.

Proof. Any boundary concept of G' can be split into c_1 and c_2 in the following way: The neighbors of c_1 are the neighbors of c, which belong to G', and the neighbors of c_2 are the neighbors of c, which do not belong to G'. Then, all connected components having a node outside G' are removed, and the remaining BG, which has been obtained from G by generalization, is precisely G'. □

2.4.2 Generalization and Homomorphism

In this section, we prove that G subsumes H if and only if G is a generalization of H. Thus, homomorphism, which is a global notion (it is a mapping between sets),

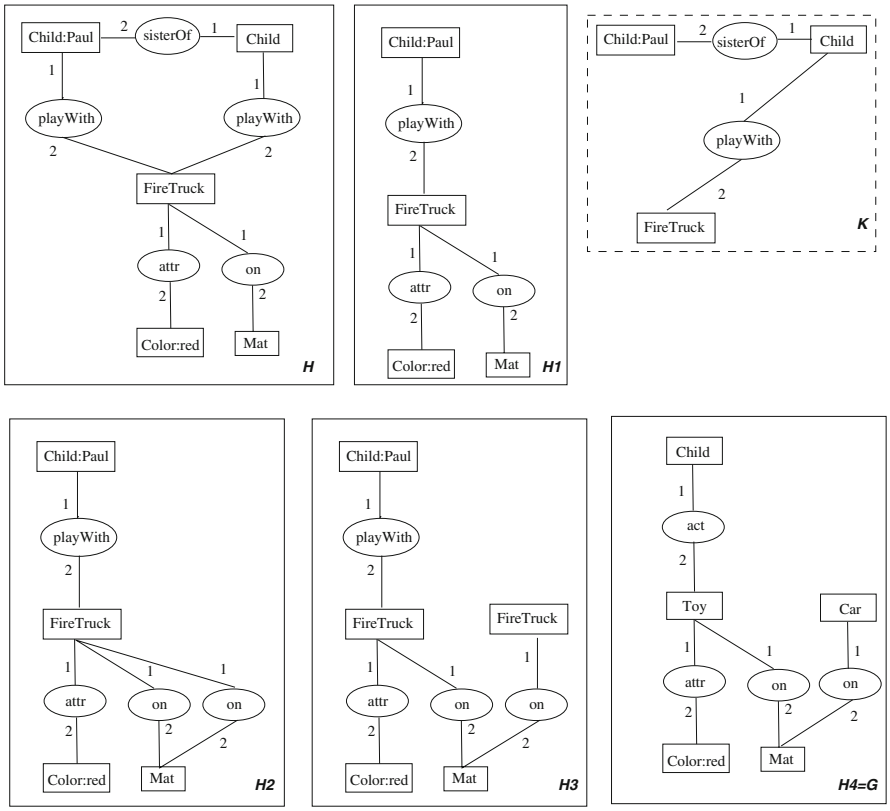

Fig. 2.20 Generalization from H to G

can be replaced by a sequence of simple operations: Copy and substract are simple to understand, and the three others are local. Furthermore, the five operations are simple to draw.

Property 2.12. If a BG G is a generalization of a BG H, then there is a homomorphism from G to H ($G \succeq H$).

Proof. One first proves that if G is obtained from H by an elementary generalization operation then there is a homomorphism from G to H. It is trivial for the copy operation since an isomorphism is a homomorphism.

- *Relation duplicate*. Let r' be a relation duplicate of a relation r. The mapping π defined as follows is a homomorphism:
  $\pi(r') = r$ and $\forall x \neq r$, x node of G, $\pi(x) = x$.

- *Increase*. Let $G = increase(H, x, L)$. The mapping which associates to each node of G its corresponding node in H is a homomorphism (the only difference

between G and H is that $l_G(x) \geq l_H(x)$).

- *Detach*. Let c_1 and c_2 be the concepts resulting from the split of c. The mapping π defined as follows is a homomorphism:
  $\pi(c_1) = \pi(c_2) = c$ and $\forall x \in C_G \cup R_G \setminus \{c_1, c_2\}$, $\pi(x) = x$.

- *Substract*. G is equal to a subBG of H and the identity is a homomorphism.

As the composition of two homomorphisms is a homomorphism, one concludes by recurrence on the length of a generalization sequence. □

Property 2.13. If there is a homomorphism from a BG G to a BG H, then G is a generalization of H.

Proof. Let π be a homomorphism from G to H. We build a sequence of generalization operations from H to G as follows.

1. We first build a generalization sequence from H to $\pi(G)$. As $\pi(G)$ is a subBG of H, property 2.11 can be applied: The sequence is composed of a sequence of Detach on the boundary nodes of $\pi(G)$ followed by a Substract.
2. for every relation r such that $|\pi^{-1}(r)| = 1$, say $\pi^{-1}(r) = \{r'\}$, the label of r is increased to that of r'.
3. for every relation r such that $\pi^{-1}(r) = \{r_1, \ldots, r_k\}$, with $k \geq 2$, r is duplicated $k-1$ times into $s_1(= r), \ldots, s_k$, and for any i, the label of s_i is increased to that of r_i.
4. for every concept c such that $|\pi^{-1}(c)| = 1$, say $\pi^{-1}(c) = \{c'\}$, the label of c is increased to that of c'.
5. for every concept c such that $\pi^{-1}(c) = \{c_1, \ldots, c_k\}$, with $k \geq 2$, c is split by a Detach into k concepts $d_1, \ldots, d_k$, and, for any i, the edges incident to d_i are the images by π of the edges incident to c_i. Furthermore, for any i, the label of d_i is increased to $l_G(c_i)$. We obtain (a BG isomorphic to) G.

□

The two previous properties give the following theorem, which justifies the term *generalization* for the existence of both a homomorphism and a sequence of elementary generalization operations.

Theorem 2.2 (Homomorphism and generalization). *There is a homomorphism from a BG G to a BG H ($G \succeq H$) if and only if G is a generalization of H.*

Example. In Fig. 2.20 some graphs of a generalization sequence from a BG H to a BG G are pictured. This sequence corresponds to the homomorphism h_1 from G to H presented in Fig. 2.10. It follows the proof steps of Property 2.13, except that all labels are increased at the last step instead of during the construction. More specifically, H_1 is equal to $h_1(G)$; its construction thus corresponds to step 1 of the proof. Step 2 is delayed. The construction of H_2 corresponds to step 3, with the increase in labels being delayed. Similarly, the construction of H_3 corresponds to step 4, with the increase in labels being delayed. Finally, the construction of H_4 gathers all label increases.

2.4.3 Elementary Specialization Operations

The elementary specialization operations defined in this section are inverse (up to an abuse of language explained hereafter) operations of the elementary generalization operation defined previously. Besides the copy operation, there are three unary operations and one binary operation.

Definition 2.17 (Elementary specialization operations). The five elementary specialization operations are:

- **Copy** (already defined as a generalization operation).
- **Relation simplify.** Given a BG G, and two twin relations r and r' (relation with the same type and the same list of neighbors), $relationSimplify(G,r')$ is the BG obtained from G by deleting r'.
- **Restrict.** Given a BG G, a node x of G, and a label $l \leq l(x)$ $restrict(G,x,l)$ is the BG obtained from G by decreasing the label of x to l, which is its type if x is a relation, its type and/or its marker if x is a concept.
- **Join.** Given a BG G, and two concepts c_1 and c_2 of G with the same label, $join(G,c_1,c_2)$ is the BG obtained from G by merging c_1 and c_2 in a new node c, i.e. the edges incident to c are all edges incident to c_1 and to c_2 and c_1 and c_2 are deleted.
- **Disjoint sum.** Given two disjoint BGs $G = (C,R,E,l)$ and $H = (C',R',E',l')$, $G+H$ is the union of G and H that is the BG $(C \cup C', R \cup R', E \cup E', l \cup l')$. G or H may be empty.

It is noted in the definition of the *disjoint sum* operation that G or H may be empty. This specific case is needed in order to obtain any BG from the empty BG $G_\emptyset$, with $G_\emptyset$ being more general than any BG. The Relation simplify and Join operations are pictured in Fig. 2.18 and Fig. 2.19, together with their inverse generalization operations, respectively Relation duplicate and Detach.

Example. In reading Fig. 2.20 from G to H, one obtains the sketch of a specialization sequence. This sequence corresponds to the homomorphism h_1 from G to H presented in Fig. 2.10. More specifically, H_3 is obtained from G by four restrict operations. H_2 is obtained from H_3 by a join of the concept nodes with label $(FireTruck,*)$. A relation simplification of a relation with label (on) in H_2 yields $H_1 = h_1(G)$. Finally, a disjoint sum of H_1 and K followed by two joins on the concept nodes with labels, respectively $(Child,Paul)$ and $(FireTruck,*)$, produces H.

It is straightforward to check the following relationships between elementary generalization operations and elementary specialization operations:

- If H is obtained from G by the deletion of a relation r', twin of a relation r, then G is isomorphic to a BG obtained from H by duplicating r, and conversely.
- If H is obtained from G by restricting to b the label a of a node x, then G can be obtained from H by increasing the label of x from b to a, and conversely.

- If H is obtained from G by joining two nodes x and y into a new node xy, then G is isomorphic to a BG obtained from H by detaching xy into two nodes x' and y' having the same neighboring as x and y in G, and conversely.
- If H is equal to the disjoint sum $G+K$, then G can be obtained from H by a substract operation.

Thanks to these relationships, we will make an abuse of language whereby we say that elementary specialization operations are inverse of elementary generalization operations.

The direct definition of a specialization notion inverse to the definition of a generalization (cf. Definition 2.16) for specialization operations is not so simple. This is due to the operation disjoint sum which is a binary operation which has one new input BG. Thus, instead of a *sequence*, we will first consider a *tree*, as illustrated in Fig. 2.21 and defined below.

Definition 2.18 (Specialization tree). A specialization tree of a BG G is an anti-rooted tree T whose nodes are labeled by BGs in such a way that:

- T has an anti-root labeled by G.
- Let x be any node with label H, then
  - x is a leaf (i.e., x has no predecessors), or
  - x has exactly one predecessor labeled K, and H is obtained from K by one of the unary specialization operations, or
  - x has exactly two predecessors, respectively labeled K_1 and K_2, and $H = K_1 + K_2$.

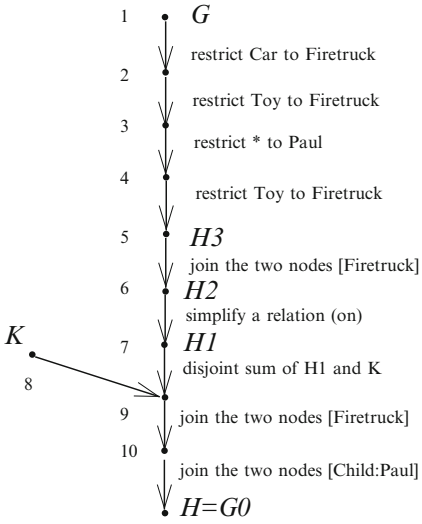

Fig. 2.21 The specialization tree corresponding to Fig. 2.20

The transitive closure of an anti-rooted tree is a partial order, and any total ordering of the nodes containing that partial order is called a linear extension of the partial order.

Definition 2.19 (Specialization). Given a specialization tree of a BG G, the sequence of BGs labeling any linear extension of the tree is called a specialization sequence of G. G is said to be a *specialization* of any BG H appearing in a specialization sequence of G.

Example. In Fig. 2.21, the vertex set of the specialization tree is $\{1,...,11\}$. There are several linear extensions of this tree which differ only by the rank of the vertex 8 labeled by K. In any case, this node has to appear before node 9. For instance the linear extension $(1\ ...7\ 8\ 9\ 10\ 11)$ yields a specialization sequence of form $(G = H_4,\ ...,\ H_3,\ ,H_2,\ H_1,\ K,\ ...,\ H)$.

Property 2.14. If G and H are two BGs, then H is a specialization of G if and only if G is a generalization of H

Proof. Let us consider a specialization sequence from G to H. Then, it is possible to transform this sequence into a generalization sequence from H to G by replacing each elementary specialization operation by an elementary generalization operation. Conversely, any generalization sequence from H to G can be transformed into a specialization sequence from G to H. □

In some situations, especially when one wants to actually build a BG by a sequence of elementary specialization operations, it is interesting to consider a specialization graph and not only a tree. Let us assume, for instance, that there is a set of BGs $\mathcal{B}$ used as building blocks (cf. Chap. 8). In other words, the leaves of the specialization tree can be labeled only by elements of $\mathcal{B}$. Then imagine that, to build a certain BG, one needs twice the "same" subBG G' (e.g., this BG has two isomorphic disjoint subgraphs). Assume that a first subBG G' has been built and G'' isomorphic to G' is needed. If a specialization tree is considered, a second graph has to be built from the building blocks in the same way as G'. In a specialization graph, G'' can be simply obtained by copying G'.

Definition 2.20 (Specialization graph). A specialization graph of a BG H is a directed graph without circuits D whose nodes are labeled by BGs in such a way that:

- D has an anti-root labeled by H.
- Let x be any node with label G, then
  - x is a source (i.e., x has no predecessors), or x has exactly one predecessor labeled K, and G is obtained from K by one of the unary specialization operations, or x has exactly two predecessors, respectively labeled K_1 and K_2, and $G = K_1 + K_2$,
  - x may have several successors, but at most one does not correspond to a copy operation.

Property 2.15. Given any specialization graph D of H, there is a specialization tree T of H with the same set of leaf labels.

Proof. Let us consider a specialization graph D with its anti-root labeled by H and the set of its leaf labels equal to $\{L_1,\ldots,L_p\}$. Let us prove the property by recurrence on the number of nodes having at least two successors. If D has no node with two or more successors, then D is a tree. Otherwise, let us consider a node x of D having at least two successors such that all the predecessors of x have exactly one successor. Let the successor set of x be $\{y_1,\ldots,y_k\}$, $k \geq 2$, with $y_2,\ldots,y_k$ being obtained from x by a copy operation. Let us denote by D_x the subgraph of D induced by all the ascendants of x (x included). The graph D' obtained from D by creating $k-1$ disjoint copies of D_x with anti-roots $x_2,\ldots,x_k$, respectively, and by replacing each arc from x to y_i by the arc joining x_i to y_i, for $i = 2,\ldots,k$, is a specialization graph with its anti-root labeled by H and its leaf label set being equal to $\{L_1,\ldots,L_p\}$. Furthermore, the number of nodes of D' having at least two successors is equal to the number of nodes of D having at least two successors minus one. One concludes by using the recurrence hypothesis. □

2.4.4 Specialization and Homomorphism

From Property 2.14 and Theorem 2.2, one obtains:

Theorem 2.3 (Homomorphism and Spec./Gen.). *Let G and H be two BGs. The three following propositions are equivalent:*

1. *G is a generalization of H*
2. *H is a specialization of G*
3. *there is a homomorphism from G to H, i.e. $G \succeq H$*

Let us consider a homomorphism π from G to H. We have shown how it is possible to build a sequence of elementary generalization operations transforming H into G (cf. proof of property 2.13). Using the duality between generalization and specialization operations, it is possible to transform this sequence into a sequence of elementary specialization operations transforming G into H (Property 2.14). For the sake of completeness, we briefly indicate how a specialization sequence transforming G into H can be directly built from π (i.e., without considering a generalization sequence from H to G). Note that this sequence is not the exact inverse of the generalization sequence given in the proof of Property 2.13 because the order in which label modifications naturally occur are not the same.

1. For every concept or relation node x in G, the label of x is restricted to the label of $\pi(x)$. We obtain G_1. Now, all nodes with the same image by π have the same label.
2. All concept nodes in G_1 with the same image in H are joined. More precisely, for every concept node c' in H such that $\pi^{-1}(c') = \{c_1,\ldots,c_k\}$, $k \geq 2$, all the c_i

are joined. We obtain G_2. Now, all relations with the same image by π are twin relations.
3. A sequence of relation simplifications is performed on G_2 to keep only one relation per set of relations with the same image in H. More precisely, for every relation r' in H such that $\pi^{-1}(r') = \{r_1, \ldots, r_k\}$, $k \geq 2$, $(k-1)$ relation simplications on r_2, ..., r_k (for instance) are performed. The graph obtained, i.e., G_3, is isomorphic to $\pi(G)$.
4. Let us consider the BG K obtained from H by deleting all nodes of $\pi(G)$ that are not boundary nodes. A disjoint sum of G_3 and K yields G_4.
5. Finally, a sequence of join operations are applied on G_4: Each boundary node of G_3 is joined to its corresponding node in K. The BG obtained is (isomorphic to) H.

Example. Let us consider Fig. 2.20, which sketches a generalization sequence from H to G guided by the homomorphism h_1 in Fig. 2.10. Reading this figure in the inverse direction, one obtains the sketch of a specialization sequence from G to H built from h_1 as above. More specifically, H_3, H_2 and H_1 respectively correspond to the BG G_1, G_2 and G_3 built at the above steps 1, 2 and 3. The union of H_1 and K corresponds to G_4. Finally, H is obtained.

2.5 Normal BGs

2.5.1 Definition of Normal BGs

Two concept nodes having the same individual marker represent the same entity. This can be seen as the simplest kind of equality. Introducing the equality between any kind of concept nodes (i.e., not only individual concepts but also generic concepts) is the main objective of the next chapter, and the present section can be considered as an introduction to this chapter.

If a BG has several individual concept nodes with the same marker, we would like to be able to merge these nodes into a single individual concept, while keeping the same intuitive meaning (and the same formal semantics, see Chap. 4). Note first that, even when there is an obvious way of merging the concepts, the obtained BG is generally not hom-equivalent to the original BG. This point is illustrated by Fig. 2.22. Let us consider the BG G, in which the individual marker m appears twice. The nodes to be merged have exactly the same label, thus merging them simply consists of performing a Join operation. Let H be the resulting BG. G and H are obviously semantically equivalent, but H is strictly more specific than G for the subsumption relation (indeed there is a homomorphism from G to H but no homomorphism from H to G). Now, assume for instance, that G represents an assertion and let Q be the query pictured in the figure. There is a homomorphism from Q to H but not to G, thus H answers Q while G does not.

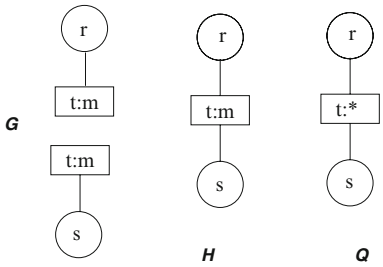

Fig. 2.22 Homomorphism and normality

Thus, BGs having several individual concept nodes with the same marker do not have a *normal* behavior with respect to the subsumption relation, and this naturally leads to the following definition of normal BGs:

Definition 2.21 (Normal BG). A BG is called *normal* if there is at most one individual concept with a given marker, i.e., all individual concepts have distinct markers.

Example. H and Q in Fig. 2.22 are normal while G is not. G in Fig. 2.23 is not normal because the individual *Paul* occurs in two concept nodes, and $norm(G)$ is a normal BG semantically equivalent to G.

A second issue concerns the conditions under which any BG can be transformed into a semantically equivalent normal BG. Let P_c denote the partition of the set of concept nodes of a BG defined as follows: Any generic concept defines a class restricted to itself, and all the individual concepts with the same marker define a class. A BG is normal if and only if its partition P_c is the discrete partition. If a BG is not normal, then one can consider its quotient graph G/P_c defined as follows:

Definition 2.22 (G/P_c). Let $G = (C, R, E, l)$ be a BG. G/P_c is the graph defined as follows:

- its set of concept nodes is in bijection with P_c,
- its set of relation nodes is in bijection with R,
- for any (r, i, c) in G, (r', i, c') is in G/P_c, where r' is the relation node corresponding to r, and c' corresponds to the class containing c.

In order to transform the graph G/P_c into a BG, a label has to be defined for the classes of P_c. The label of a node corresponding to a trivial class of P_c is the label of the unique node of that class. The marker associated with a non trivial class is obviously the individual marker appearing in all nodes of this class. But, if no conditions are enforced on the types of individual concepts composing the class, there is no garantee that a type preserving semantic equivalence can be defined.

We shall now enumerate some simple conditions, but, as we shall see, none is completely satisfactory from a knowledge representation viewpoint.

- **Unicity condition**
  One can impose that all concepts with the same individual marker have the same

type. Then the label of the node of G/P_c corresponding to a non-trivial class is the label of any concept in this class. In this case, G and G/P_c have the same intuitive meaning.

Example. By merging c_1 and c_2 in G in Fig. 2.23, the normal BG $norm(G)$ is obtained.

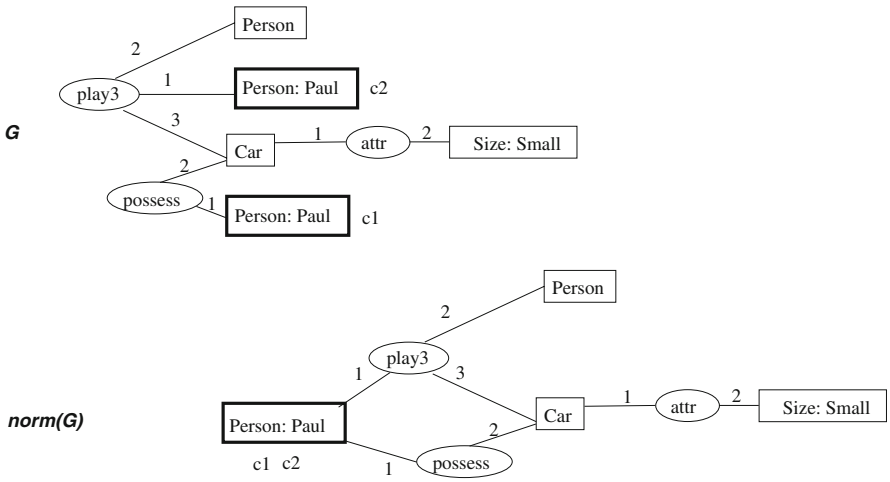

Fig. 2.23 Normalization of a BG

In such cases, G/P_c is a normal BG that is called the *normal* form of G. The notation $norm(G)$ is also used. A canonical homomorphism from G to G/P_c can be defined. This homomorphism associates to any concept node c the node corresponding to the class containing this node, and to any relation its corresponding relation.

But allowing individual concepts with the same marker and different types could be useful. For instance, let us consider the following situation where #123 is a pick-up car. One would like to represent specific characteristics of the pick-up #123 (e.g., number of kilometers in the bush, composition of first-aid kit, state of the tires) by an individual concept of type *Pickup* and marker #123, and to represent administrative data of the car #123 (e.g., names and addresses of the successive owners) by another individual concept of type Car and the same individual marker. In this situation, as *Pickup* is a subtype of *Car*, these two individual concepts could be merged into a single individual concept of type *Pickup* while keeping the same semantics. This leads to the following condition.

- **Minimality condition**
  One can relax the unicity condition of types and replace it by a minimality condition, i.e., for any class A of P_c the set T_A of types of concepts in A has a minimal element $min(T_A)$ (within T_A). In this case, G and G/P_c have the same intuitive meaning (with the type of concept in G/P_c associated with the class A being equal to $min(T_A)$). But this minimality condition is still too strong. Indeed, let

us consider a BG having a class $A = \{c, c'\}$ of P_c with labels $l(c) = (t, m)$ and $l(c') = (t', m)$, and t and t' incomparable. The minimality condition is not respected, and the two individual concepts cannot be merged, even if m actually represents an entity of type t'' less than t and t'. But what happens if $\{t, t'\}$ has several minimal elements (within T_C)?
This leads to the following condition.

- **Greatest lower bound condition**
  This condition consists of requiring that the set of all concept types of any class of P_c has a greatest lower bound (glb) in T_C. Hence, as P_c is relative to a specific BG, generalizing this glb condition to any BG leads us to assume that T_C is an inf-semi-lattice. But in this case, if one wants to keep the BG semantic, one must assume the following property: If m is a t and m is a t', then m is a $glb(t, t')$. This lattice-theoretic interpretation of the concept type set is discussed in the next chapter.
- **Typing of individual markers**
  Let us recall that a typing of the individual markers is a mapping τ from the set of individuals $\mathcal{I}$ to the set of concept types T_C. For an individual m, all occurrences of individual concept nodes with marker m must have a type greater than or equal to $\tau(m)$. Having a typing of individual markers comes down to the unicity condition. Indeed, before any computation with a BG, all types of individual concepts with marker m can be replaced by $\tau(m)$.
  In the Simple Conceptual Graphs model presented in the next chapter, we will see how this problem is tackled and how it is possible to explicitly express that several concepts, either generic or individual, represent the same entity. Thus the simple conceptual graph model can be considered as the basic conceptual model enriched by equality between concepts.

2.5.2 Elementary Operations for Normal BGs

The subsumption relation, defined through the homomorphism notion, is the fundamental reasoning tool in conceptual graphs. Thus, normal BGs, which have a nice behavior relative to homomorphism, are the fundamental class of BGs (and of the simple conceptual graphs studied in the next chapter). Previous generalization and specialization operations correspond to homomorphisms between BGs. But can we take these operations as reference operations on normal BGs? The only flaw is that, as such, they are not internal operations on normal BGs since a non-normal BG can be produced from a normal BG. Moreover, given two normal BGs G and H with $G \succeq H$, there may be no way of deriving one from the other without producing a non-normal BG somewhere in the derivation (cf. Fig. 2.24).

If we consider generalization, the guilty operation is the operation *detach* (when it splits an individual node). Concerning specialization, there are two guilty operations, *restrict* (when it restricts a generic concept to an individual concept, whereas

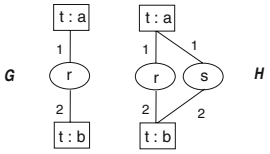

Fig. 2.24 Every specialization sequence from G to H contains a non-normal BG

the individual marker already appears elsewhere in the BG) and *disjoint sum* (when there are individual concepts with the same label in the two BGs).

Let us consider a situation where it is possible to merge individual concepts having the same marker. Then the operations *detach*, *restrict* and *disjoint sum* can be slightly transformed as follows:

- **detach**$_{norm}$. Modify *detach* as follows: If c is an individual concept, replace the marker of c_2 by a generic marker (which involves an increase).
- **restrict**$_{norm}$. Modify *restrict* as follows: Let x be a generic concept restricted to an individual node; if there is already a concept with the same individual marker, merge these concepts (which involves a restrict of the greater type to the smaller, if the types of the concepts are not equal, followed by a join).
- **disjoint-sum**$_{norm}$. Modify *disjoint sum* as follows: A disjoint sum operation is followed by the merging of the individual concepts having the same marker.

Note that the modified generalization and specialization operations are internal operations on normal BGs. It remains to be shown that they define the same relationship between normal BGs as the original ones.

Property 2.16. Let G and H be two *normal* BGs. G is a generalization of H using detach$_{norm}$ instead of detach (notation $G \succeq_{norm} H$) if and only if there is a homomorphism from G to H.

Proof. Similar to the proofs of Property 2.12 and Property 2.13 for BGs. Each generalization operation yielding G_j from G_i defines a homomorphism from G_j to G_i, thus by composition of all homomorphisms associated with a derivation from H to G, one obtains a homomorphism from G to H. Reciprocally, let us consider the proof of Property 2.13: detach$_{norm}$ preserves the property that any subBG of H is a generalization of H (Property 2.11), thus the proof still holds for $\succeq_{norm}$. □

Property 2.17. Let G and H be two *normal* BGs. H is a specialization of G (notation: $H \preceq_{norm} G$) using restrict$_{norm}$ instead of restrict and disjoint-sum$_{norm}$ instead of disjoint sum if and only if $G \succeq_{norm} H$.

Proof. Similar to the proof of Property 2.14.

If G_j is obtained from G_i by a relation duplicate operation, then G_i is obtained from G_j by a relation simplify operation.

If G_j is obtained from G_i by a detach$_{norm}$ and the marker m of the individual node c_2 becomes a generic marker, then G_i can be obtained from G_j by a restrict$_{norm}$ of

c_2 to the marker m.
If G_j is obtained from G_i by an increase operation, then G_i is obtained from G_j by a restrict operation, therefore by a restrict$_{norm}$ operation.
If G_j is obtained from G_i by a substract operation, then G_i is obtained from G_j by a disjoint sum, therefore by a disjoint-sum$_{norm}$ operation.
It is straightforward to check the other direction. □

Corollary 2.1. *Let G and H be two* normal *BGs. $H \preceq_{norm} G$ if and only if $G \succeq_{norm} H$.*

Theorem 2.4 ($\preceq_{norm}$) and homomorphism). *Let G and H be two* normal *BGs. The three following propositions are equivalent:*

1. $H \preceq_{norm} G$
2. $G \succeq_{norm} H$
3. *there is a homomorphism from G to H.*

2.6 Complexity of Basic Problems

Let us consider the following three basic problems:

- BG-HOMOMORPHISM: given two BGs G and H, is there a homomorphism from G (the source) to H (the target)?
- HOM-EQUIVALENCE: given two BGs, are they equivalent?
- REDUNDANCY: given a BG, is it redundant?

We will show that they are all NP-complete. Let us first stress two points. First, the fact that these problems are NP-complete does not mean that they cannot be efficiently solved in practice. Chapter 6 is devoted to algorithmic techniques for finding homomorphisms. Chapter 7 is devoted to particular cases in which BG-HOMOMORPHISM can be solved in polynomial time.

Secondly, the really fundamental problem is BG-HOMOMORPHISM, and that explains why we focus on it throughout this book. Indeed, let us assume that we have an algorithm for solving BG-HOMOMORPHISM; checking whether two BGs are hom-equivalent can be done with two calls to this algorithm, and checking whether a graph G is redundant can be done with a number of calls to this algorithm, which is linear in the size of G. In the trivial case where G has a redundant relation node (i.e., a relation node with type t such that there is another relation node with the same argument list and type $t' \leq t$), it is redundant. Otherwise, G is redundant if and only if it can be mapped to one of its subgraphs $G - \{x\}$ obtained by deleting a concept node x and all relation nodes incident to it. There are at most $|C_G|$ such subgraphs. Thus, an efficient algorithm for BG-HOMOMORPHISM directly yields an efficient algorithm for HOM-EQUIVALENCE and REDUNDANCY.

In practice, we are interested in computing the irredundant form of a graph, and not only in checking whether it is redundant. Assume that we have an algorithm able to exhibit a homomorphism from G to H if any. A simple algorithm for computing

the irredundant form of G is as follows: First, delete redundant relation nodes in G; secondly, check if there is a homomorphism from G to one of its subgraphs $G-\{x\}$; if no, G is its own irredundant form; if yes, let π be a homomorphism from G to $G-\{x\}$: The irredundant form of G is the irredundant form of $\pi(G)$.

Theorem 2.5. BG-HOMOMORPHISM *is NP-complete.*

Proof. BG-HOMOMORPHISM is obviously in NP. To prove the completeness we build a reduction from the well-known 3-SAT problem. The input of 3-SAT is a propositional formula f in 3-conjunctive normal form (3-CNF), i.e., a conjunction of disjunctions (or clauses), each with three literals, and the question is whether there is a truth assignment of the variables in f such that f is true.

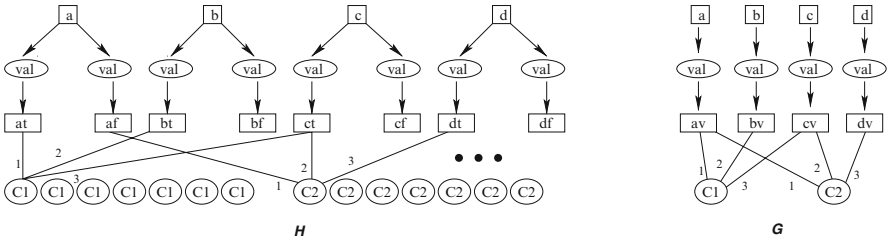

Fig. 2.25 Example of transformation from 3-SAT to BG-HOMOMORPHISM

Let $f = C_1 \wedge \ldots \wedge C_k$ be an instance of 3-SAT. Without loss of generality, then we suppose that a variable appears at most once in a clause. Let us create four concept types for each variable x: `x`, `xf`, `xt` and `xv`. We also create one relation type `C`$_i$ for each clause C_i, and a relation type `val`. Each concept type `xv` is greater than `xt` and `xf`, these are the only possible comparisons between distinct types.

We build the graph $H(f)$ as follows: For every variable x in f, we have three concept nodes `[x]`, `[xt]` and `[xf]` in $H(f)$ and two relation nodes typed `val` linking the first to the latter ones (intuitively, this means that the variable x can be valuated by `true` or `false`). Let us say that the truth value `true` (resp. `false`) is associated with `[xt]` (resp. `[xf]`). Then for every clause $C_i = (l_x \vee l_y \vee l_z)$ in f (where l_x, l_y and l_z are literals over variables x, y and z), we add the 7 relation nodes typed `C`$_i$, having as first argument `[xt]` or `[xf]`, as second argument `[yt]` or `[yf]`, and as third argument `[zt]` or `[zf]`, that correspond to an evaluation of the clause to true (more precisely, if we replace, in the clause C_i, each positive (resp. negative) literal l_j, $1 \leq j \leq 3$, by the truth value (resp. the negation of the truth value) associated with the *jth* neighbor of the relation node, C_i is evaluated to true).

In graph $G(f)$, two concept nodes `[x]` and `[xv]` are created for each variable x and they are linked by a binary relation `(val)`. For each clause $C_i = (l_x \vee l_y \vee l_z)$, there is one relation `(C`$_i$`)` linked to `[xv]`, `[yv]` and `[zv]`. This question means "Is there a valuation of variables such that all clauses evaluate to `true`?"

This transformation from the 3-SAT formula $(a \vee b \vee \neg c) \wedge (\neg a \vee c \vee \neg d)$ is illustrated in Fig. 2.25. In graph H, not all edges issued from the clauses have been

drawn, for readability reasons. It is immediate to check that, for a formula f, there is a valuation of its variables such that each clause is evaluated to true if and only if $G(f)$ can be mapped into $H(f)$. □

Chapter 5 provides other polynomial reductions to BG-HOMOMORPHISM, in particular from GRAPH-HOMOMORPHISM, CONJUNCTIVE-QUERY-CONTAINMENT, CONJUNCTIVE-QUERY-EVALUATION and CSP (Constraint Satisfaction Problem).

Deciding whether two BGs G and H are hom-equivalent or checking whether a graph is redundant could be problems that are simpler than BG-HOMOMORPHISM, but they are not:

Theorem 2.6. HOM-EQUIVALENCE *and* REDUNDANCY *are NP-complete.*

Proof. Let (G,H) be an instance of BG-HOMOMORPHISM. It is easily checked that there is a homomorphism from G to H if and only if H is hom-equivalent to $G+H$ (the disjoint sum of G and H), hence we have an immediate reduction from BG-HOMOMORPHISM to HOM-EQUIVALENCE. Let us now build a reduction from BG-HOMOMORPHISM to REDUNDANCY. Without loss of generality we assume that G and H have no redundant relation nodes. We build G' and H' from G and H, respectively, by adding some "gadgets" such that (1) G' and H' are irredundant, (2) there is no homomorphism from H' to G', and (3) there is a homomorphism from G to H if and only if there is a homomorphism from G' to H'. For instance, let r and s be binary relation types that do not occur in G and H and that are incomparable to all other relation types. We obtain G' from G by adding a relation node of type r between any two distinct concept nodes x and y in G. We obtain H' from H by adding a relation node of type r between all concept nodes x and y in H, where x and y may be the same node, and a relation node of type s between all distinct x and y in H. Check that G' and H' satisfy conditions (1), (2) and (3). Because of (1) and (2), their disjoint sum $G'+H'$ is redundant if and only if there is a homomorphism from G' to H', thus, from (3), if and only if there is a homomorphism from G to H. □

2.7 Bibliographic Notes

In his seminal book [Sow84], Sowa first settled a simple model and progressively added more complex notions. This simple model corresponds to the "basic conceptual graphs" studied in this chapter. In [CM92] a first study of its properties was conducted. This basis has evolved over the years. The definitions and results of this chapter integrate this evolution, but essentially rely on [Sow84] and [CM92].

The "vocabulary" defined in Sect. 2.1.1 is an evolution of the "support" in [CM92], itself based on the semantic network in [Sow84]. Originally, the concept type set was a lattice and relation symbols were not ordered. In addition, a "conformity relation" enforced constraints on labels of individual concept nodes. It is equivalent to the individual typing mapping τ mentioned at the end of Sect. 2.1.1.

In Sowa's book, basic conceptual graphs were connected graphs. Specialization operations, called canonical formation rules, were introduced. It was shown that a mapping π from G to H, called a projection, could be associated with every specialization sequence from G to H. In [CM92] projection was identified as a graph homomorphism and its equivalence with a specialization sequence was stated, by proving the reciprocal property: To every projection a specialization sequence can be associated. Generalization operations dual of specialization operations were also introduced. Equivalence between projection/homomorphism, generalization and specialization was thus proved (Theorem 2.2 and Theorem 2.3). Specialization and generalization rules of the present book basically come from the extension of [CM92] to non-connected graphs in [MC96] (research report [CM95] for an English version). Other sets of rules have been proposed, we will comment on them in the bibliographical notes in the Simple Conceptual Graphs chapter, as these rules consider graphs with equality. The fact that this subsumption is not an order (contrary to the claim in [Sow84]), but only a preorder had been several times noted (e.g., [Jac88]). [CM92] introduced irredundancy and proved that each BG equivalence class contains a unique irredundant graph (Theorem 2.1). The additional property 2.9 is from [CG95]. In the same paper, an alternative proof of Property 2.3 is provided. Normal graphs were independently introduced in [MC96] and [GW95]. The NP-completeness of BG-HOMOMORPHISM, HOM-EQUIVALENCE and REDUNDANCY was proven in [CM92] (with different proofs).

http://www.springer.com/978-1-84800-285-2

Graph-based Knowledge Representation
Computational Foundations of Conceptual Graphs
Chein, M.; Mugnier, M.-L.
2009, XIV, 428 p., Hardcover
ISBN: 978-1-84800-285-2