Chapter 1
Introduction

In Sect. 1.1, we place the book in the "Knowledge Representation and Reasoning" (KR) Artificial Intelligence (AI) domain. We first briefly outline key concepts of the KR domain, and then review KR formalism properties that we consider to be essential. The second section is devoted to an intuitive presentation of Conceptual Graphs that were initially introduced by Sowa in 1976 [Sow76] and developed in [Sow84]. In the third section, we introduce the graph-based KR formalism that is detailed in the book. This KR formalism is based on a graph theoretical vision of conceptual graphs and complies with the main principles delineated in the first section.

1.1 Knowledge Representation and Reasoning

Knowledge Representation and Reasoning has long been recognized as a central issue in Artificial Intelligence. Very generally speaking, the problem is to symbolically encode human knowledge and reasoning in such a way that this encoded knowledge can be processed by a computer via encoded reasoning to obtain intelligent behavior. Human knowledge is taken here in a very broad sense. It can be the knowledge of a single person, of an expert in some domain, shared knowledge of ordinary people (common sense knowledge), social knowledge accumulated by generations, e.g., in a scientific domain, etc. Thus, we will not distinguish the *modeling view* of KR, which involves studies on how to computationally represent knowledge about the world, or the *cognitivist view* of KR, which assesses how to computationally represent cognitive capacities of a human being.

Moreover, we shall carefully avoid specifying the exact meanings of the notions of "human knowledge," "reasoning," "intelligence" or "representation." All of these issues have been discussed by philosophers since the Greek ancient times (at least in the western world), and a discussion of such topics would be far beyond the scope of this book.

1.1.1 Knowledge-Based Systems

KR is the scientific domain concerned with the study of computational models able to explicitly represent knowledge by symbols and to process these symbols in order to produce new ones representing other pieces of knowledge. Systems built upon such computational models are called knowledge-based systems. Their main components are a knowledge base and a reasoning engine.

Example (a photo of children). Assume we want to represent common sense knowledge about a photo depicting children playing in a room containing toys and furniture. The formalism studied in the book can be used to represent elements in this photo (e.g., there is a girl and a boy, a car is on the table), and knowledge about elements in the photo (e.g., Mary is the name of the girl, Mary is a sister of the boy). It is also used to represent general background knowledge (e.g., a building block is a toy, if a person A is a sister of a person B then A and B are relatives).

Knowledge representation and reasoning formalism can also express problems to be solved concerning the facts and general knowledge represented. For instance, one may ask with what kind of toy Mary's brother is playing. Answering such questions requires descriptive knowledge but also reasoning capabilities (e.g., *modus ponens*, which states that if A holds and if B can be deduced from A, then B holds).

Main Components of a Knowledge Base

A knowledge base (KB) gathers symbolic knowledge representation about an application domain. We use the expression "application domain" to denote the part of the world (which can be real or fantasy, a sophisticated model of a system or a model of an expert competence) about which we represent knowledge and reasoning.

A KB generally contains different kinds of knowledge, typically an *ontology*, *facts*, *rules* and *constraints*. From an epistemological viewpoint, an ontology provides an answer to the question "What kinds of things exist in the application domain?" or expressed in a more generic way, "How can we think about the world ?" A computational *ontology* provides a symbolic representation of objects, classes of objects, properties and relationships between objects used to explicitly represent knowledge about an application domain. It is the cornerstone of a knowledge representation since all other pieces of knowledge (e.g., facts, rules or constraints) are represented by computational structures built with ontology terms.

Besides a KB, a knowledge-based system contains a *reasoning engine*. The reasoning engine processes knowledge in a KB in order to answer some question or to solve some goal. A reasoning engine is composed of algorithms processing elements of the KB in order to construct "new" knowledge, i.e., new symbolic constructs that are only implicit in the KB. We should stress that in a knowledge-based system we cannot have knowledge representation without having reasoning mechanisms. A large part of KR research consists of finding a tradeoff between expressivity or generality of knowledge representation formalism and the efficiency of the reasoning mechanisms.

Knowledge Incompleteness

An essential point is that a KB is not assumed to provide a complete picture of the world. The fundamental reasons are that any real thing, e.g., a human face or a pebble, cannot be described by a finite set of symbolic structures, and also that a thing does not exist in isolation but is included in unlimited sets of encompassing contexts. Thus, incompleteness of descriptions is a central feature of knowledge-based systems, and is a main distinction with respect to databases: For some sentences, it cannot be determined whether they are true or false given the knowledge in the base. For instance, a KB representing the photo of children can be queried by an unlimited number of questions (e.g., "Is the house where the photo was taken located in a village?" "How old is the boy?" "Who are the children's parents?" "In what country were the toys built?" and so on *ad libitum*). To be answered, these questions would need an unlimited amount of knowledge.

1.1.2 Requirements for a Knowledge Representation Formalism

In a nutshell, we are interested in KR formalisms that comply, or aim at complying, with the following requirements:

1. to have a *denotational* formal semantic,
2. to be *logically* founded,
3. to allow for a *structured representation* of knowledge,
4. to have good *computational* properties,
5. to allow users to have a maximal *understanding and control* over each step of the KB building process and use.

The graph-based KR formalism presented in this book has the first three properties, parts of the formalism have the fourth property, and we think that, at least for some application domains, it has the last one too. We think that presently there is no universal KR formalism. Indeed, such a formalism should represent natural languages, and the present systems are far from being able to do that. Thus, every existing KR formalism, including the one presented here, can be efficiently used only for specific reasoning on specific knowledge (e.g., a privileged application domain, namely semantic annotation, will be briefly presented). The end of this section is devoted to a brief discussion on the previous five requirements.

1.1.2.1 Denotational Semantics: *What* Rather than *How*

A KR formalism should allow us to represent knowledge in an *explicit* and a *declarative* way: The meaning of the knowledge represented should be definable independently of the programs processing the knowledge base. Namely, it should not be necessary to precisely understand how reasoning procedures are implemented to

build a knowledge representation, and one should be able to update the knowledge base content without modifying any program. Ideally, the result of inferences should depend only on the semantics of the given data and not on their syntax, i.e., semantically equivalent knowledge bases should lead to semantically equivalent results, regardless of their syntactical forms. Thus, having a denotational semantics is an essential KR formalism feature.

A set (or model) semantics is appreciable, particularly whenever the KR formalism has to be used by informatics non-specialists. Indeed, the basic notions of (naive) set theory: element and membership, subset and inclusion, application, relation, etc., are easily understood by many people. In addition, a set semantics should provide the notions of truth and entailment, so that what holds in the modeled world can be determined. This leads us to logic, since logic is the study of entailment, or in other words, reasoning.

1.1.2.2 Logical Foundations

Generally speaking, doing an inference consists of producing a new expression from existing ones. The correctness of an inference mechanism can be defined relative to a logic, and in this book we essentially consider logical entailment, or logical deduction.

What does it mean for a KR formalism to be logically founded? First, the expressions of the formalism are translatable into formulas of a logic. Such a mapping gives a logical semantics to the formalism. Secondly, the reasoning engine contains an inference mechanism, which should have two essential properties with respect to deduction in the target logic: *soundness* and *completeness*. Let $\mathcal{K}$ be a knowledge base expressed in some KR formalism, and let f be a logical semantics of $\mathcal{K}$, i.e., f is a mapping from $\mathcal{K}$ to formulas of some logic. The inference mechanism is *sound* with respect to this semantics if for each expression i inferred from $\mathcal{K}$, $f(i)$ is actually logically deduced from $f(\mathcal{K})$. It is *complete* if, for each expression i such that $f(i)$ is logically deduced from $f(\mathcal{K})$, i is actually inferred from $\mathcal{K}$. In other words, a procedure $\mathcal{P}$ (or algorithm, or system of rules, etc.) is sound with respect to a logical semantics if every inference made by $\mathcal{P}$ corresponds to a logical deduction. It is complete with respect to a logical semantics if every logical deduction in the logical language target of the formalism can be done by $\mathcal{P}$.

Soundness is usually ensured. But not all reasoning algorithms are complete. If an incomplete, nevertheless sound, system answers "yes" to the question "can i be retrieved from $\mathcal{K}$?" then the answer is correct. If it answers "no," then because of incompleteness, it should preferably have answered "I don't know." If the system also computes answers to a query then, if it is sound, all the computed answers are correct answers, and if it is incomplete some answers can be missed.

The incompleteness of an algorithm can be motivated by efficiency concerns when a complete reasoning is too time consuming. It can also be due to the undecidability of the deduction problem. For instance, deduction in First Order Logic (FOL) is undecidable, which means that it is not possible to build an algorithm

deciding in finite time for any pair of formulas (f, g) whether f can be deduced from g. More precisely, FOL deduction is only semi-decidable: One can build an algorithm guaranteed to stop in finite time if the answer is "yes," but which may run indefinitely if the answer is "no." Thus, algorithms that stop in finite time in all cases are necessarily incomplete.

Different logics have been developed for KR purposes. FOL has been adopted as a reference formalism for knowledge representation in the AI community. Its model semantics is described with simple mathematical objects (sets and relations). However, its computational complexity has led to study of fragments of it with good computational properties.

KR formalisms can be compared according to different criteria, such as expressiveness, ease of use, computational efficiency, etc. Logical semantics facilitate expressiveness comparisons between KR formalisms and can avoid doing something that is already known. Indeed, KR deals with knowledge and reasoning, and logic (not only classical logic) is precisely the study of reasoning. The large corpus of results and techniques accumulated by logicians for more than two millennia cannot be ignored.

From a modeling standpoint, other important properties of a KR formalism are its *empirical soundness* and its *empirical completeness* with respect to an application domain. A formalism is empirically sound if any "true" expression in the formalism corresponds to a "true" fact of the application domain. It is empirically complete if any true fact of the application domain can be coded in a true expression of the KR formalism. Naturally, when there is no mathematical model of the application domain, these notions are informal, rather subjective and difficult to evaluate. Having different mathematical semantics of the KR formalism (e.g., a set semantics and a logical semantics) can help, since each semantics can be used to study, with different notions, the correspondence between the KR formalism and the application domain.

1.1.2.3 Knowledge Structuring

A KR formalism should provide a way of structuring knowledge. Knowledge structuring can be motivated by model adequacy (i.e., its "conformity" to the modeling of the application domain) and by efficiency concerns (algorithmic efficiency of reasoning procedures, facility for managing the knowledge base).

One aspect of knowledge structuring is that semantically related pieces of information (e.g., information relative to a specific entity) should be gathered together. This idea was underlying the first KR formalisms, frames and semantic networks, that were far from logics. Frames have been introduced in [Min75] as record-like data structures for representing prototypical situations and objects. The key idea was to group together information relevant to the same situation/object. Semantic networks were originally developed as cognitive models and for processing the semantics of natural language (cf. [Leh92] for a synthesis on semantic networks in AI). A semantic network is a diagram that represents connections (relationships)

between objects (entities) or concepts (classes) in some specific knowledge domain. Basic links are the *ISA* link that relates an object and a concept of which it is an instance, the *AKO* (A-Kind-Of) link, that relates two concepts (with one being a kind of the other), and the *property* link that assigns a property to an object or concept. Inferences are done by following paths in the network. For instance, properties of an object are inherited following the ISA and AKO links. The main criticism concerning semantic networks was their *lack of a formal semantics*: What's in a link? [Woo75] What's in a concept? [Bra77]. The same network could be interpreted in different ways depending on the user's intuitive understanding of its diagrammatical representation.

Description logics (DLs), formerly called terminological logics or concept languages, are rooted in semantic networks, particularly in the KL-ONE system [BS85], and have been a successful attempt to combine well-defined logical semantics with efficient reasoning [BCM$^+$03]. This is one of the most prominent KR formalism families. Let us point out, however, that DLs have lost the graphical aspects of their ancestors. Conceptual graphs represent another family of formalisms issued from semantic networks (at least partially, because they are also rooted in other domains), which we shall consider in more detail in the next section.

Another aspect of knowledge structuring is that different kinds of knowledge should be represented by different KR formalism constructs. Ontology, facts, rules, constraints, etc., are distinct sorts of knowledge that are worth being differently represented. An important distinction is between *ontological* and *factual* knowledge. Ontological knowledge is related to general categories, also called concepts or classes. Factual knowledge makes assertions about a specific situation (e.g., "this specific entity belongs to a certain category, and has a certain relationship with another entity, ..."). This distinction is essential in description logics and conceptual graphs.

Another kind of knowledge is *implicit knowledge* described, for instance, by rules of form "if *this* then *that*" (e.g., "if there is a relation r from x to y, then the same relation r also holds from y to x"). Different kinds of rules can be considered. Some rules can be included in an ontology (e.g., the transitivity of a binary relation), other rules, for instance representing possible transformations, are not ontological knowledge. Constraints, i.e., conditions that are to be fulfilled by some pieces of knowledge in order to be correctly processed by a reasoning engine, frequently appear in modeling and should be differentiated from other kinds of knowledge.

1.1.2.4 Good Computational Properties

We are interested in KR formalisms that can be used for building systems able to solve *real problems* and not only toy problems. It is thus essential to anchor these formalisms in a computational domain having a rich set of efficient algorithms. This is a key point. AI aims at building systems (or agents) for solving complex tasks and, from a complexity theory viewpoint, one can say that simple AI problems are computationally difficult to solve—they are often NP-complete or NP-hard. Thus, if

on one hand a KR formalism must firmly moor to logics in order to avoid reinventing the wheel, on the other hand it must also moor to a rich algorithmic domain so that usable systems can be built.

1.1.2.5 Knowledge Programming and the Semantic Gap

A KR system should allow a user to have a maximal understanding and control over each step of the reasoning process. It should make it easy to enter the different pieces of knowledge (e.g., ontological knowledge as well as factual knowledge) and to understand their meaning and the results given by the system, and also (if asked by the user) how the system computed the results. Any computing system should have these qualities, i.e., should limit the semantic gap between real problems and their formulation in a programming language.

The correspondence between knowledge about an application domain and an expression in the KR formalism representing this knowledge must be as tight as possible. Due to the importance of natural language and schemas in the description of knowledge, a KR formalism should allow the user to easily represent simple phrases in natural language and simple schemas. The ability for describing such a correspondence, i.e., the *natural semantics* of a formal expression, is a good empirical criteria for delimiting the usability of the formalism.

As already said, in order to understand the results given by the system, a precise description of *what* is obtained (a denotational semantics) is mandatory. However, in some situations, especially whenever the knowledge deals with a poorly formalized application domain, it may be useful to understand *how* the results have been obtained, i.e., to have an operational semantics. This point is important because there can be a gap between the program and the knowledge represented in the system, i.e., a formal system, and the knowledge itself. There should be a tight correspondence between what is seen by the user and how objects and operations are implemented. This correspondence should be tight enough to enable faithful modeling of the actual data and problems, and to understand why and how results have been obtained. A way to limit the semantic gap is to use a homogeneous model—the same kinds of object and the same kinds of operation occur at each fundamental level (formal, user interface, implementation). Such a correspondence is sometimes called an "isomorphism," but this designation can be misleading. Indeed, this correspondence is not a mathematical function between two mathematical objects but is rather a correspondence between "reality" and a mathematical object. There is a gap between a reality and the concepts describing it, as well as between a conceptual modeling and its implementation, i.e., a representation of this modeling by computational objects. Moreover, as in this book we use "isomorphism" in its usual mathematical sense, we avoid using it metaphorically.

1.2 Conceptual Graphs

This section provides an intuitive introduction to conceptual graphs. Precise definitions are given in subsequent chapters. The conceptual graph model was introduced by Sowa in 1976 [Sow76], developed in [Sow84], and then enriched and developed by the conceptual graph community (cf. the proceedings of the International Conference of Conceptual Structures). It is the synthesis of many works in AI, but its roots are mainly found in the following areas: natural language processing, semantic networks, databases and logics, especially the existential graphs of Pierce, which form a diagrammatical system of logics.

We use the term "conceptual graphs" (CGs in short) to denote the *family* of formalisms rooted in Sowa's seminal work, rather than a precise formalism, and we use specific terms—e.g., basic conceptual graphs, simple conceptual graphs, positive nested conceptual graphs—for notions which are mathematically defined and studied in this book.

1.2.1 Basic Notions

For an intuitive introduction to CGs, let us consider again the photo of children example. We would like to represent some features of this photo and some knowledge necessary to answer non-trivial questions about the content of the photo.

The basic vocabulary is composed of two partially ordered sets: a partially ordered set of concepts or classes (called concept types in the CG community) and a partially ordered set of relation symbols (also called relation types). The partial order is interpreted as a specialization or *AKO* relation: $t_1 \leq t_2$ means that t_1 is a *specialization* of t_2 (or t_2 is a generalization of t_1, t_2 subsumes t_1, t_1 is a subtype of t_2 , or every entity of type t_1 is of type t_2). There is also a set of names, called individual markers, used for denoting specific entities. This basic vocabulary can be considered as a rudimentary ontology.

A basic conceptual graph (BG) is composed of two kinds of nodes, i.e., *concept* nodes representing entities and *relation* nodes representing relationships between these entities. Nodes are labeled by types, and one can indicate that a concept node refers to a specific entity by adding an individual marker to its label. Otherwise the concept node refers to an unspecified entity.

For instance, the conceptual graph K in Fig. 1.1 asserts that:

- there are entities (represented by rectangles): Mary who is a Girl, a Boy, who is unspecified, and a Car, which is also unspecified.
- there are relations (represented by ovals) between the entities: a relation asserting that Mary is the sister of the Boy, and two relations asserting that Mary and the Boy play with the Car. The numbers on edges are used to totally order the neighbors of each relation node. There is also a unary relation asserting that Mary is smiling.

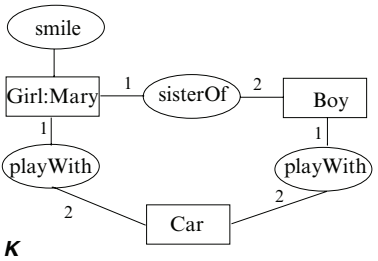


Fig. 1.1 A basic conceptual graph

This BG could be translated by the sentence "Mary (who is a girl) and her brother are playing with a car; Mary is smiling."

Important differences between the conceptual graph model and its semantic networks ancestors are to be pointed out: Firstly, there is a clear distinction between ontological knowledge (e.g., concept or relation types) and other kinds of knowledge (such as factual or implicit knowledge); secondly, relations can be of any arity, whereas the edges of semantic networks represent binary relations only; thirdly, CGs have a logical semantics in FOL.

1.2.2 Subsumption and Homomorphism

The fundamental notion for studying and using BGs is *homomorphism*. In the CG community, a BG homomorphism is traditionally called a *projection*. We prefer the term "homomorphism" for two reasons. First, it corresponds to the classical homomorphism notion in mathematics, and we will see that a BG homomorphism is a mapping between BGs preserving the BG structure. Secondly, there is an operation called "projection" in the relational database model, which is quite different from a BG homomorphism.

A *homomorphism* from a BG G to a BG H is a mapping from the nodes of G to the nodes of H, which preserves the relationships between entities of G, and may specialize the labels of entities and relationships. In graph-theoretic terms, it is a labeled graph homomorphism that we will precisely describe later. For the moment, it is only necessary to know that a *generalization/specialization* relation (or *subsumption*) over BGs can be defined with this notion: G is more general than H (or G subsumes H, or H is more specific than G) if there is a homomorphism from G to H.

Let us consider the graphs G and K in Fig. 1.2. The following mapping from the node set of G to the node set of K (pictured in dashed lines) defines a homomorphism from G to K:

the concept node [Girl] is mapped to the concept node [Girl: Mary] (the unidentified girl is specialized into the girl Mary),

the concept node [Child] is mapped to the concept node [Boy], with Boy being a subtype of Child (the child is specialized into a boy),
the concept node [Toy] is mapped to the concept node [Car], with Car being a subtype of Toy,
the relation node labeled (relativeOf) is mapped to the relation node (sisterOf) (the relation "to be a relative of someone" is specialized into the relation "to be a sister of someone"),
each relation node (actWith) is mapped to a relation node (playWith): The node (actWith) with a first neighbor [Girl] is mapped to the node (playWith) with a first neighbor [Girl:Mary], while the node (actWith) with a first neighbor [Child] is mapped to the node (playWith) with a first neighbor [Boy].

G is a generalization of K (or K is a specialization of G) since each node of G is mapped to a specialized (or identical) node of K, and because the relationships between concept nodes in G are specialized by (or identical to) relationships between the image nodes in K.

This homomorphism maps G to a subgraph of K with the same structure as G. This property is generally not fulfilled by a homomorphism, as illustrated by the graphs H and K in Fig. 1.3: there is a homomorphism from H to K, which maps both concept nodes [Car] in H to the same node [Car] in K. Graph H states that "Mary is playing with a car and a boy is playing with a car" and K specializes it by adding that they play with the same car. Moreover, K contains other relations, which are not involved in the homomorphism from H to K.

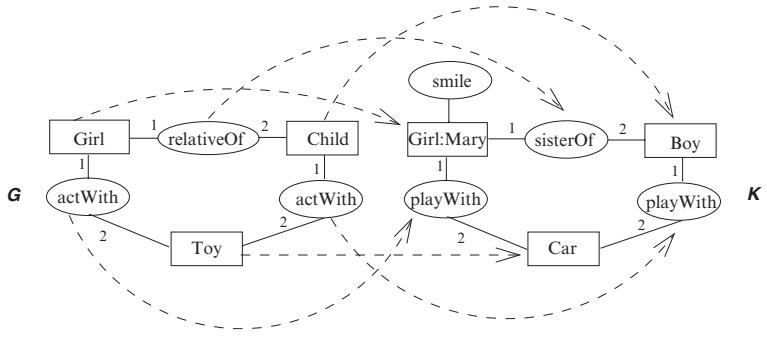


Fig. 1.2 G is a generalization of K

The fundamental problem for BGs is as follows: Given two BGs G and H, is G a generalization of H? i.e., is there a homomorphism from G to H? We call it BG-HOMOMORPHISM. We will see that this problem is NP-complete and possesses interesting polynomial cases. Furthermore, we will see that it is equivalent to other fundamental problems in AI and databases, such as the constraint satisfaction problem, which has been very well studied from an algorithmic viewpoint, or the conjunctive query inclusion problem in database theory.

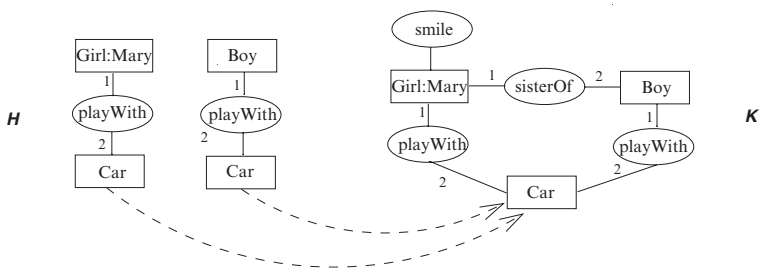


Fig. 1.3 *H* is a generalization of *K*

1.2.3 Formal Semantics

Another essential point is that BGs possess a set semantics and a logical semantics, with the former corresponding to the model theory of the latter.

Let us briefly outline how we provide BGs with a set semantics. First, one has to define a model of a vocabulary. A model of a vocabulary consists of a non-empty set D (the objects of the application domain), called the domain of the model, and the definition δ of the meaning of each element of the vocabulary. δ assigns a part of D (the set of objects of type t) to any concept type t, δ assigns a k-ary relation over D (i.e., a part of D^k composed of the tuples of objects which are related by the relation r) to any relation r of arity k and δ assigns an element of D to any individual marker. As an example let us consider a situation described by the graphs G and K in Fig. 1.2.

- The individual marker Mary is translated by an element in the domain D (i.e., $\delta(Mary)$ is the element in D representing the individual marker Mary).
- The concept types in the vocabulary, Girl, Boy, Car, etc., are translated by subsets of D (e.g., $\delta(Girl)$ is the subset of D representing the concept type Girl).
- The binary relation symbols in the vocabulary, actWith, playWith, sisterOf, relativeOf, etc., are translated by binary relations over D (e.g., $\delta(sisterOf)$ is the binary relation over D representing the binary relation symbol sisterOf), and the unary relation symbol smile is represented by a subset $\delta(smile)$ of D.

Secondly, we define BG models and the meaning of a BG that is satisfied by a model (e.g., the type of the individual marker Mary is Girl, thus $\delta(Mary)$ must be in $\delta(Girl)$; Mary is smiling, thus ($\delta(Mary)$ must be in $\delta(smile)$). Then, an entailment relation between BGs can be defined and, finally, its relationships with homomorphism is stated: Given two BGs G and H, there is a homomorphism from G to H if and only if H entails G.

Let us now outline the logical semantics, classically called Φ. The vocabulary is logically interpreted as follows. A predicate t is assigned to each type t (a unary predicate to a concept type and a k-ary predicate to a k-ary relation type) and a constant m to each individual marker m (for simplicity, here we use the same symbol

for an object in the CG world and for its corresponding object in the FOL language, e.g., a unary predicate t is assigned to a concept type t).

Given two k-ary types t_1 and t_2, $t_1 \leq t_2$ is interpreted by the formula $\forall X\ (t_1(X) \rightarrow t_2(X))$, where X is a tuple of k variables, e.g., $\forall x\ (Girl(x) \rightarrow Child(x))$ or $\forall x \forall y\ (sisterOf(x,y) \rightarrow relativeOf(x,y))$. An existentially closed formula is assigned to a BG, where terms (variables or constants) correspond to concept nodes.

For instance, the formula assigned to G in Fig. 1.1 is:
$\Phi(G) = \exists x \exists y (Girl(Mary) \wedge Boy(x) \wedge Car(y) \wedge smile(Mary) \wedge sisterOf(Mary,x) \wedge playWith(Mary,y) \wedge playWith(x,y))$.

Homomorphism is *sound* and *complete* with respect to logical deduction, i.e., given two BGs G and H, there is a homomorphism from G to H if and only if the formula $\Phi(G)$ can be deduced from the formula $\Phi(H)$ and the logical translation of the type hierarchies. The BG-HOMOMORPHISM problem can thus be identified with a deduction problem. We will show later that basic conceptual graphs strongly correspond to the existential, positive, conjunctive fragment of FOL (which we denote by FOL($\exists$, $\wedge$)) [1].

Basic conceptual graphs constitute the kernel of CGs. They can be used as such to represent facts and queries. They are also basic bricks for more complex constructs, such as nested graphs or rules, corresponding to more expressive CGs.

1.2.4 Full CGs

The most expressive conceptual graphs we shall consider, and call *full conceptual graphs*, were introduced by Sowa in [Sow84] and are inspired from Peirce's existential graphs (cf. [Dau02] for a mathematical study of full CGs and [Rob92] for a presentation of Peirce's existential graphs). The basic idea of the extension from BGs to full CGs is that every FOL formula can be written as a formula by solely using the existential quantifier and the conjunction and negation logical connectors. By adding to BGs *boxes* representing negation and *lines* (called co-reference links) indicating that two nodes represent the same entity, one obtains the same expressiveness as FOL. For instance, the graph of Fig. 1.4 shows that the relation r is transitive on entities of type t, i.e., for all x, y, z of type t, if $r(x,\ y)$ and $r(y,\ z)$ then $r(x,\ z)$. Its logical translation is more precisely:

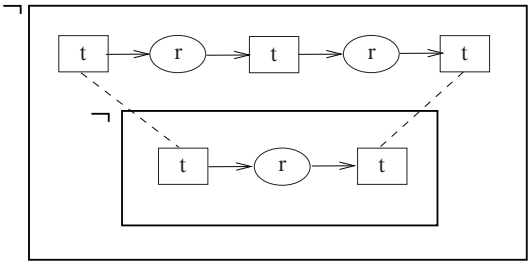
$\Phi(G) = \neg(\exists x\ \exists y\ \exists z(t(x) \wedge t(y) \wedge t(z) \wedge r(x,y) \wedge r(y,z) \wedge \neg(r(x,z))))$,
which is equivalent to: $\forall x \forall y \forall z((t(x) \wedge t(y) \wedge t(z) \wedge r(x,y) \wedge r(y,z)) \rightarrow r(x,z))$

Peirce's existential graphs are provided with a sound and complete set of inference rules that can be adapted to CGs [Sow84] [Wer95a] [Dau02]. However, these rules do not directly lead to automated reasoning because they heavily rely on human intuition[2].

---

[1] We will see that the universally quantified formulas associated with the vocabulary can be dropped without restraining logical expressivity.

[2] For instance, the insertion rule allows one to insert *any* graph (at a place obeying specific conditions), which leads to an infinite number of choices.



For binary relation nodes, directions on edges may replace numbers 1 and 2.

Fig. 1.4 A full CG

In this book, we will only briefly present full CGs, which are unsuitable for our approach, as will become clear in the next section. Instead, we will build limited extensions of BGs (e.g., with atomic negation), in an attempt to keep the essential properties of BGs.

1.3 A Graph-Based Approach to KR

Since 1991 (cf. [CM92]) our aim has been to develop and study a KR formalism respecting, as far as possible, the five requirements presented in Sect. 1.1. Our work belongs to a logical approach to KR (and to the KR scientific community mainstream as presented, for instance, by Brachman and Levesque in [BL04] or Baader in [Baa99]), but it is also *graph-based* as explained hereafter. The formalism is based on graphs and graph-theoretic notions and operations. It is logically founded, but in some way is "autonomous" from (existing) logics. Stated differently, our KR formalism is a pure graph-theoretic formalism, whose core corresponds to a FOL fragment, and most extensions of this core correspond to FOL fragments. Since it is embedded in graph theory, it is easy to define new operations, simple from a graph viewpoint, and having simple intuitive semantics, but that do not necessarily have a formal semantics expressed in a classical logic.

1.3.1 Motivations

Let us first outline our motivations for a graph-based approach to KR. They can be divided into two categories: qualities of graphs for knowledge modeling and qualities of graphs for computations.

From a *modeling viewpoint*, we see two essential properties in the basic conceptual graph model. The *objects*, i.e., basic graphs, are easily understandable by users (typically knowledge engineers or specialists in an application domain), at least if the graphs are reasonably small (note that it is always possible to split up a large conceptual graph into smaller ones while keeping its semantics). This feature partially explains the success of semantic networks and, more generally, the success of graphical models, such as the entity/relationships model, UML, Topic Maps, etc. Many people, and not only computer scientists, are now familiar with kinds of labeled graphs (mainly trees, but not exclusively). This fact is especially important in knowledge acquisition. In our approach to CGs, this quality does not only concern the descriptive facet of the formalism. *Reasoning* mechanisms are also easily understandable, for two reasons. First, homomorphism is a "global" graph-matching notion that can be easily visualized (we will also see that homomorphism is equivalent to a sequence of elementary graph operations which are very simple and easy to visualize). Secondly, the same language is used at interface and computing levels. In particular, no transformation has to be done on the components of the knowledge base before reasoning with it.

Thus, reasoning can be explained on the user's representation itself, and explanations can be given at any level from the user's level to the implementation level. At the implementation level, a graph can be represented by a structure with pointers, which is a graph too! To sum up, using a graph-based KR should reduce the semantic gap mentioned in Sect. 1.1.2.

From a *computational viewpoint*, labeled graph homomorphism firmly moors BGs to combinatorics. The graph homomorphism notion (or its variant, relational structure homomorphism) was recognized in the 90s as a central notion, unifying complexity results and algorithms obtained in several domains (e.g., cf. [Jea98] and [FV93]). On the other hand, considering graphs instead of logical formulas provides another view of knowledge constructs (e.g., some notions like path, cycle, or connected components are natural on graphs) and provides other algorithmic ideas, as we hope is illustrated throughout this book.

1.3.2 Extensions of the Basic Formalism

Full CGs *à la Peirce* are no longer graphs (in the graph theory sense); they are diagrams. Associated inference rules are not graph-based operations either. In our opinion, qualities of the BG model from a knowledge representation and reasoning perspective (as presented above) are at least partially lost: namely, the readability of objects as well as the easy understanding of the inference mechanism, and relationships with combinatorial problems.

Rather than jumping from BGs to full CGs, we prefer, depending on the kind of knowledge we would like to represent, to build extensions of the BG model, while keeping its essential properties. These properties represent our motto and can be summarized as follows:

1. objects are *labeled graphs* (mathematically defined with graph-theoretic notions),
2. reasoning mechanisms are based on graph-theoretic operations, mainly relying on graph homomorphism,
3. efficient reasoning algorithms exist for important specific cases,
4. objects and operations have graphical representations, which make them easily understandable by users (limitation of the semantic gap),
5. the BG model is logically founded, with the inference mechanism being sound and complete with respect to FOL semantics.

Let us briefly give an example of extension: BG rules. A rule represents information of the type: "if information H is found, then information C can be added." H is called the hypothesis of the rule and C its conclusion. This notion of a rule has been widely used in AI to represent implicit knowledge, which can be made explicit by applying the rules, on facts for instance.

Rules could be represented as full CGs, but in so doing they would lose their specificity and could not be processed in a particular manner. A BG rule can be defined as a bicolored BG (a more general rule definition will be given). One color (white in the figures) defines the hypothesis, and the other color defines the conclusion (gray in the figures). For instance, the rule in Fig. 1.5 has the same semantics as the full CG in Fig. 1.4, i.e., it says that the relation r is transitive on type t entities. A more complex rule is pictured in Fig. 1.6. This rule allows us to decompose the ternary relation *give* into simpler relations: If there is a relation *give* with first argument a $Human\ x$, with second argument a $Thing\ y$ and with third argument an $Animate$ entity z, then there is a $Gift$ act, whose x is the *agent*, y is the $object$ and z is the $recipient$.

The notion of a rule application is very simple. A rule R is said to be applicable to a BG G if its hypothesis H can be mapped by a homomorphism to G. Then it can be applied to G: Applying R consists of adding the conclusion of R to G guided by the homomorphism from H to G. Rules are provided with a logical semantics extending that of BGs, such that graph mechanisms, namely forward chaining and backward chaining, are sound and complete. Let us consider a KB $\mathcal{K}$ composed of a set $\mathcal{F}$ of BGs representing facts and a set $\mathcal{R}$ of rules. A BG is derived from $\mathcal{K}$ if it can be obtained by a sequence of rule applications to the facts. The basic problem considered is thus as follows: Given a KB $\mathcal{K} = (\mathcal{F}, \mathcal{R})$ and a BG Q, is a specialization of Q derivable from $\mathcal{K}$? Due to the soundness and completeness of the graph mechanisms defined, it can be seen as a deduction problem, called $\mathcal{FR}$-DEDUCTION. By enriching BGs with rules, we obtain a computability model. This is an important property but, similar to the fact that no computability model can be the basis for building a universally good programming language, this does not mean that this KR formalism is suitable for all KR domains. Moreover, this high expressivity comes with undecidability of reasoning. That is why properties on rule sets ensuring decidability of reasoning are studied. A simple example is that of rules that do not add unspecified concept nodes (such as the rule in Fig. 1.5, contrary to the rule in Fig. 1.6 which adds an unspecified Gift). In this case, $\mathcal{FR}$-DEDUCTION

is NP-complete, thus not more difficult than deduction checking in the BG model, i.e., BG-HOMOMORPHISM.

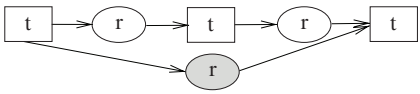


Fig. 1.5 A simple rule

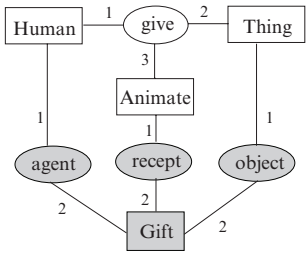


Fig. 1.6 Another rule

Other extensions presented in the book are *type conjunctions*, *nested graphs*, *atomic negation*, and *constraints*, as well as a family of models combining rules and constraints, called the BG-Family.

1.3.3 Several Approaches to CGs

Let us end this section by situating our approach in the CG landscape. Research on CGs can be roughly classified according to three axes: CGs can be seen as a *diagrammatic interface* for other formalisms or as a *diagrammatic calculus of logics*, or as a *graph-based KR formalism*.

Many works are mainly focused on the visual qualities of conceptual graphs. The expressivity and readability of the obtained representations are therefore the main criteria for evaluating a formalism. We shall not forget that a main motivation behind conceptual graphs was the processing of natural language semantics: From this standpoint, the relative easiness in translating a (small) conceptual graph into a natural language sentence it represents is an important criterion. When reasoning mechanisms are associated with representations, they are not part of the CG formalism: They are performed in a procedural way, or rely on another formalism into which the CGs are translated. In this approach, conceptual graphs are seen as a diagrammatical interface for other formalisms (e.g., the Common Logic Standard and

CLIF [CLS07]). They are not a knowledge representation and reasoning formalism *per se*, i.e., in the sense of Sect. 1.1.

Other works can be seen as the continuation of Peirce's work on a diagrammatical system of logics. Conceptual graphs are then diagrams, and reasoning is based on diagrammatic operations. In these works, automated reasoning is not the point and the computational aspect of the formalism is absent.

Finally, the graph-based approach emphasizes the following points (cf. our motto in Sect. 1.3.2), which distinguishes it from the other two approaches:

- CGs are seen as a KR *and reasoning* formalism. Thus, they are provided with their own operations for reasoning.
- Reasoning mechanisms should be *sound and complete* with respect to a formal semantics.
- Reasoning operations should be conducted in an *efficient way*. That is why decidability and complexity studies, as well as the design of efficient algorithms are important issues.
- Reasoning mechanisms are based on graph-theoretic notions, mainly labeled graph homomorphism, as the structure underlying objects is a graph.

We became highly interested in CGs when we proved (cf. [Mug92] and [CM92]) that homomorphism on the BG fragment is complete with respect to the FOL semantics. As Sowa in [Sow84] had proven that homomorphism is sound (with respect to this FOL semantics), the completeness theorem established the graph homomorphism notion as a key reasoning tool. It is agreed that graph homomorphism is a fundamental notion in graph theory and the results we have obtained, as well as results in other fields (e.g., cf. [Jea98] and [FV93]), strengthened our belief that labeled graph homomorphism is a key notion in KR.