In Proceedings of the eigth International Conference on Conceptual Structures (ICCS'2000), pages 156–170. Darmstadt, Germany, August 2000. Springer-Verlag.

Conceptual Graphs, Metamodeling and Notation of Concepts

Olivier Gerbé[1], Guy W. Mineau[2], and Rudolf K. Keller[3]

[1] HEC Montreal.
3000, chemin de la Côte-Sainte-Catherine, Montréal, Québec, Canada H3T 2A7
`Olivier.Gerbe@hec.ca`
[2] Université Laval
Québec, Québec, Canada G1K 7P4
`mineau@ift.ulaval.ca`
[3] Université de Montréal
C.P. 6128 Succursale Centre-Ville, Montréal, Québec, Canada H3C 3J7
`keller@IRO.UMontreal.ca`

Abstract. Knowledge management, in particular corporate knowledge management, is a challenge companies and researchers have to meet. The conceptual graph formalism is a good candidate for the representation of corporate knowledge, and for the development of knowledge management systems. But many of the issues concerning the use of conceptual graphs as a metalanguage have not been worked out in detail. By introducing a function that maps higher level to lower level, this paper clarifies the metalevel semantics, notation and manipulation of concepts in the conceptual graph formalism. In addition, this function allows metamodeling activities to take place using the CG notation.

1 Introduction

Knowledge management, especially corporate knowledge management, is a challenge companies and researchers have to meet. In a previous work, we compared conceptual graphs with other formalisms [3, 5] and we concluded that the conceptual graph formalism was a good candidate for the representation of corporate knowledge, and for the development of knowledge management systems.

Conceptual graphs are a knowledge representation formalism introduced by John Sowa [7] where objects of the universe of discourse are modeled by concepts and conceptual relations that associate concepts. Conceptual graphs have been extensively used and studied by a large scientific community. Using conceptual graph formalism, a corporate memory has been developed at the Research & Development Department of DMR Consulting Group Inc in order to memorize the methods, know-how and expertise of its consultants [4]. This corporate memory, called Method Repository, is a complete authoring environment used to edit, store and display the methods used by the consultants of DMR. The core of the environment a knowledge engineering system based on conceptual graphs.

Four methods are commercially delivered and their documentation in paper and in hypertext format is generated from conceptual graphs. About two hundred business processes have been modeled and from about 80,000 conceptual graphs, we generated more than 100,000 HTML pages in both English and French that can be browsed using commercial Web browsers.

However some fundamental aspects of conceptual graphs remains ambiguous and not formally specified. For example, the notation of concepts is syntactically defined in the new CG standard [1] but its use and its semantics seems ambiguous. Figure 1 illustrates this problem of notation. How could one represent the notion (concept) of the person John using conceptual graphs?

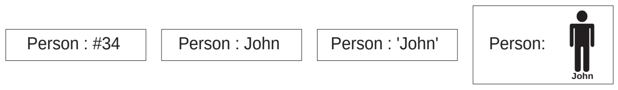

Fig. 1. Which notation?

Another example is the manipulation of embedded graphs. Concepts may have conceptual graphs in the referent field. Figure 2 illustrates this problem of manipulation. How could a CG system access the conceptual graph that is in the referent field of this graph?

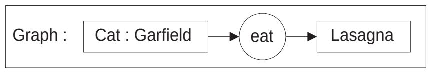

Fig. 2. How to access the embedded graph?

We propose in this paper a metamodeling approach which will semantically define the notation and manipulation of concepts in the CG language. Little work has been done on metamodeling and conceptual graphs. John Esch in [2] introduces two relationships: Kind that links a concept to its type and Subt that links two concept types that are in a subtype relationship. He defines, using the relationship Subt, a type hierarchy for each higher order level and he links, using the relationship Kind, types from different levels. But Esch does not deal with conceptual relations and relation types. In [8] Michel Wermelinguer defines more formally higher order types and proposes a translation to first order logic. He defines one hierarchy for all the concept types and one hierarchy for all the relation types and organizes them in regard of their nature and their order. But none proposes a complete metamodel of the conceptual graph language itself.

These two approaches are compatible and our work extends the notions introduced by these authors. We formally define the conceptual graph language using conceptual graphs and we propose in this paper a notation for referents based on the metamodel. In addition, we unify the two operators τ and ρ [6] in one general operator ω that maps a higher order level to a lower order level and so allows the manipulation of concepts at the lower levels.

This paper[1] is organized as follows. Section 2 presents an overview of the metamodel of the conceptual graph formalism and Section 3 details five basic components of this metamodel: element, concept, referent, individual concept and generic concept. Based on this metamodel, Section 4 formalizes the notation of referents in concepts and Section 5 introduces the function ω that links a concept to the entity it represents and illustrates the use of this function in metamodeling, laying the foundation for a CG theory of metamodeling. Section 6 concludes and presents future work.

2 CG Metamodel Overview

This section gives an overview of the conceptual graph metamodel. The metamodel defines the basic components needed to represent knowledge. Figure 3 presents the concept type hierarchy of the CG language metamodel.

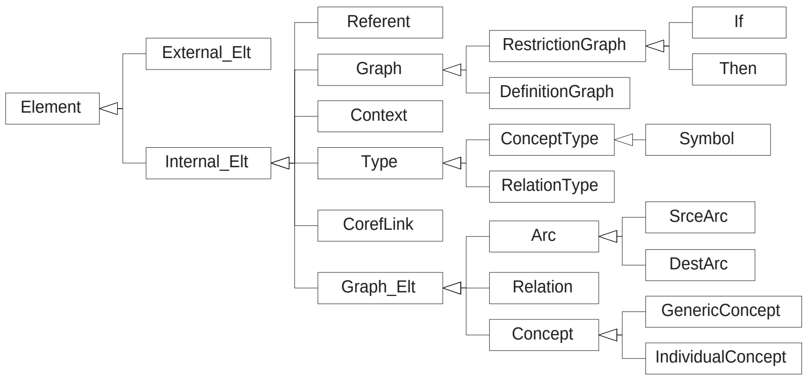

Fig. 3. Metamodel Concept Type Hierarchy.

At the highest level, we have external elements (ExternalElt) that are part of the real world to be represented and internal elements (InternalElt) that are building blocks of the CG language.

External elements represent entities of the Universe of Discourse that are outside of the system but can be referenced by internal elements.

Internal elements are categorized under six types: Referent, Graph, Context, Type, CorefLink, and GraphElement. Referents are the proxies that stand for entities of the Universe of Discourse; Graphs are the sentences of the CG language; Contexts help to cluster knowledge; Types are categories to classify entities; Coreference Links associate elements that represent the same entities; and Graph Elements are concepts, relations and arcs.

[1] This work is part of a research project supported by HEC Montreal.

Among concepts we distinguish individual concepts (IndividualConcept) that represent identified entities and generic concepts (GenericConcept) that represent unidentified entities.

Graph has two specialized subtypes: DefinitionGraph and RestrictionGraph. Definition graphs are used to define concept types and relation types. Restriction graphs are graphs that must always be false. They are used to state constraints on types. If and Then are special cases of restriction graphs where the conditions they represent must already be present or be acquired simultaneously.

3 CG Metamodel Components

This section details the core components of the conceptual graph formalism that are relevant to the notation and manipulation of concepts. We introduce a formal definition using conceptual graphs of the five basic concept types related to concept: Element, Concept, Referent, Individual Concept and Generic Concept. But before, in order to understand specification graphs in formal definitions, we illustrate on an example how concept types are specified.

3.1 Concept Type Specification

Concept Types are defined by three kinds of graphs : definition graph, restriction graph, and rule graph as illustrated in Figure 4. This figure shows the graph that specifies the concept type Driver. Definition, restriction and rule graphs give the necessary and sufficient conditions to recognize instances of Driver.

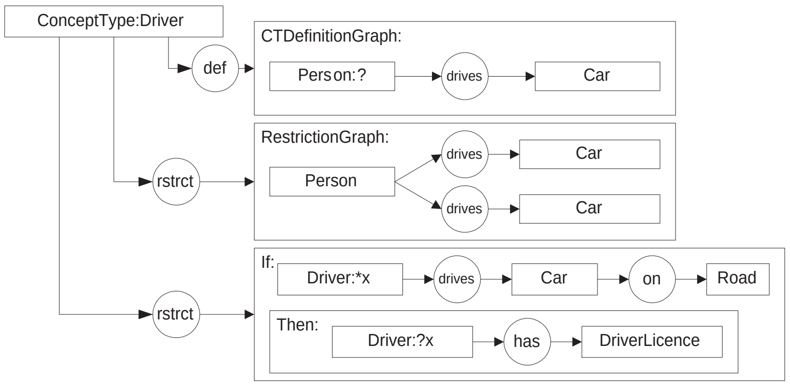

Fig. 4. Type definition of Driver.

Definition Graph. The definition graph shows the conceptual relations that a concept must have. In the example a person that drives a car is a driver under the conditions stated by restriction graphs and rule graphs.

Formally, in [7], a concept type is defined by a lambda expression, the example in Figure 4 states the following equation :

`Driver = [Person : λ]→(drives)→[Car].`

The symbol λ shows that the concept Person is the formal parameter. In the graphic form and in the linear form the symbol λ is replaced by a question mark.

Restriction Graph. Restriction graphs specify supplementary and necessary conditions; these graphs are forbidden graphs or subgraphs. They show particular topologies that must not exist. In our example, the restriction graph states that a driver cannot drive two cars[2].

Rule Graph. Rule graphs specify other supplementary and necessary conditions that are expressed more easily with rules[3]. In our example, the rule graph states that if a driver drives a car on a road then he has a driver license.

3.2 Element

The conceptual graph language is made of elements that are combined to represent knowledge. Figure 5 shows the specification graph of type Element. An element is symbolized by one or more symbols but one symbol may not be linked to two different elements. Symbols are unique identifiers in the system.

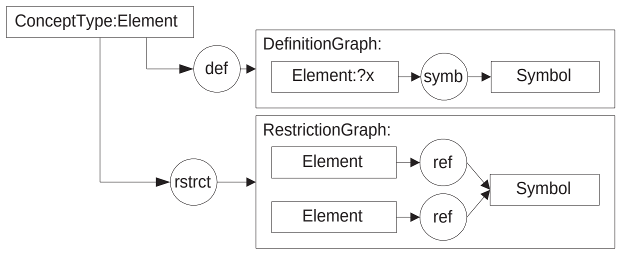

Fig. 5. Specification graph of type Element.

3.3 Concept

The notion of concept is the fundamental notion of the conceptual graph theory. A concept is the representation of an object, an idea or any notion one can perceive and express.

[2] Two different boxes represent two different concepts.
[3] Any rule graph can be rewritten as a restriction graph.

Definition 1. *A concept is the representation of an object of the Universe of Discourse. It is the assembly of two parts: a referent that identifies the represented object and the type that classifies it.*

Figure 6 shows the specification graph of type Concept. The definition graph states that a concept is an internal element that has a type, a referent, and is element of a graph. Restriction graphs clarify building rules, a concept has one and only one type and one and only one referent.

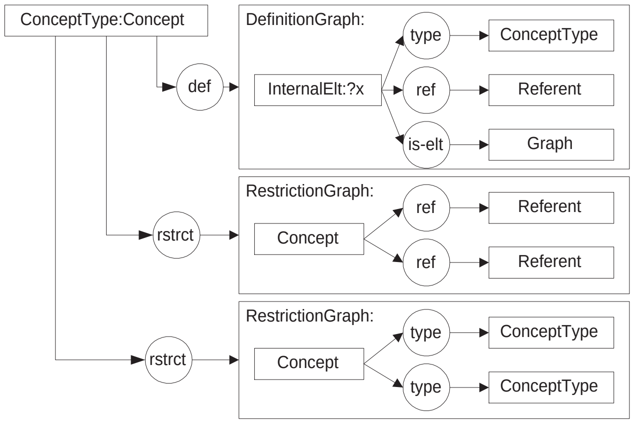

Fig. 6. Specification graphs of type Concept.

One can notice that one type may be associated to several different referents and one referent may be associated to different types and make up different concepts. This later mechanism provides the representation of point of views, one object may be perceived in different ways.

Examples of concepts are :

```
[Person], [TaxiDriver], [Concept]
[Person : #Tom], [Concept : #624], [ConceptType : Person]
```

Examples of concepts whose concept type is Concept are:

```
[Concept : #624], [Concept : [ConceptType : Person] ],
[Concept : [Referent : #624] ]
```

3.4 Referent

The referent is the part of a concept that represents and identifies the object of the universe of discourse for which the concept is an interpretation.

Definition 2. *A referent is a proxy for an object of the universe of discourse in the knowledge base. A referent is made up of a quantifier and a designator that refers to the object.*

Figure 7 shows the specification graph of type Referent. A referent is a part of a concept and represents an element (external or internal). Restriction graphs state that one referent represents one and only one element and two referents cannot represent the same element. There exists a special referent written `#blank` that 'represents' an element which exists but is not identified (3.6).

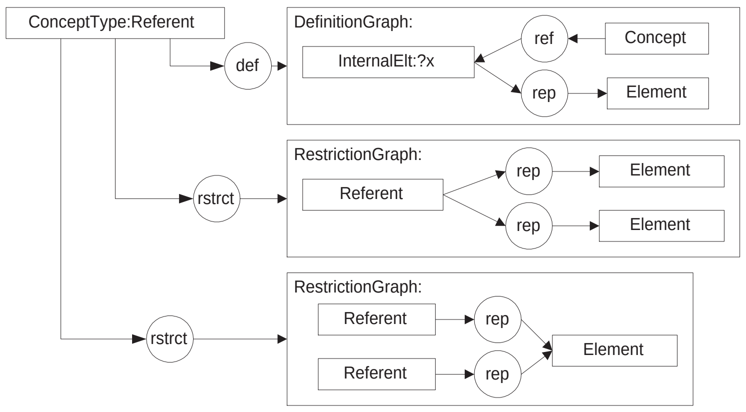

Fig. 7. Specification graph of type Referent.

Examples of referents are :

`#12, #blank, Tom, Person, ∀*x.`

3.5 Individual Concept

Definition 3. *An individual concept is a concept whose represented entity is known. The concept is an interpretation of an object on the universe of discourse that is identified and represented in the knowledge base by a referent other than* `#blank`.

Figure 8 presents the specification graph of type IndividualConcept. An individual concept has a referent that represents an element. The referent of an individual concept is different from the blank referent.

Examples of individual concepts are :

```
[Person : Tom], [Referent : #624], [ConceptType : Person]
```

3.6 Generic Concept

There exist two kinds of concepts depending whether the entity represented by the concept is known or unknown.

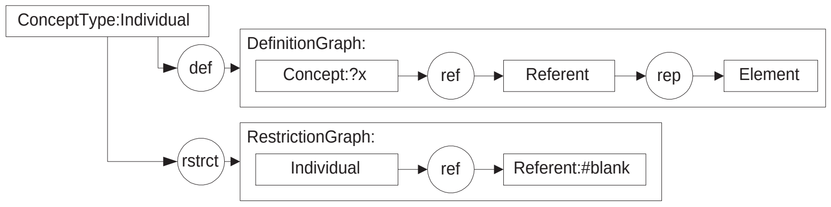

Fig. 8. Specification graphs of type IndividualConcept.

Definition 4. *A generic concept is a concept that represents an unknown entity. The concept is an interpretation of an object of the universe of discourse that exists but is not identified.*

Figure 9 presents the specification graph of type GenericConcept. A generic concept has the special referent `#blank` that represents an unidentified element. In practice this referent is omitted.

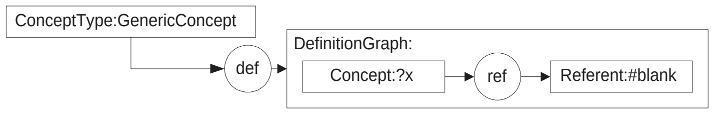

Fig. 9. Specification graphs of type GenericConcept.

Examples of generic concepts are :

```
[Person], [TaxiDriver:#blank], [Concept]
```

4 Notation

This section formalizes the notation of concepts in the conceptual graph formalism. Individual concepts are the basic components of the CG language. Individual concepts are concepts that are abstractions of well identified entities. The referent slot of an individual concept is symbolized by a unique literal preceded by the symbol `#`. The referent represents the identified entity in the system. The entity of the universe of discourse itself may be symbolized. Figure 10 presents the underlying metamodel. A concept has a referent. This referent is symbolized by a symbol and represents an element that may also be symbolized.

An alternative for the notation of the concept is to replace its symbol by any symbol of the represented element. This mechanism may be formalized by a metalevel rule as illustrated in Figure 11. If an element represented by the referent of a concept is itself symbolized by symbols then the referent may be symbolized by the same symbols.

We illustrate this rule on three different examples: notation of concept, notation of concept type and notation of graph.

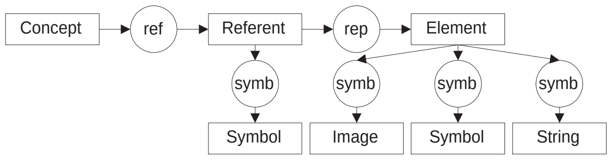

Fig. 10. The metamodel of symbolization.

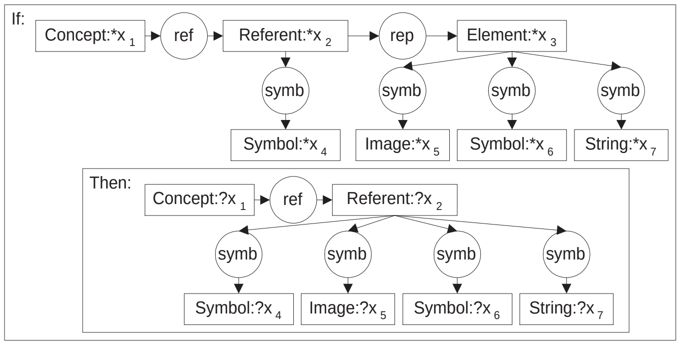

Fig. 11. The notation rule.

4.1 Notation of Concept

A lot of different notation has been used in the literature to denote individual concepts. We give here the example of the representation of the person John. According to the metamodel of Figure 10, the metalevel conceptual graph describing the concept that stands for the person John is presented in Figure 12. The concept `[Person:#34]` has a referent that is symbolized by `#34`. This referent represents the element John (an external element), which can be symbolized by different symbols: a string, a symbol[4] and an image.

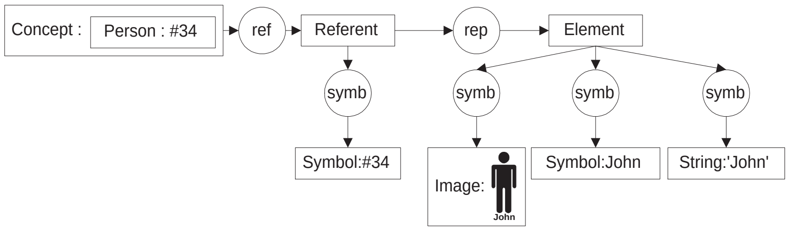

Fig. 12. The meta conceptual graph for notation of John.

[4] A symbol is different from a string. It is a whole and cannot be edited

By applying the notation rule defined above the referent `#34` may be replaced by any of the symbols that stands for the person John. Figure 13 shows the possible notation of the same concept.

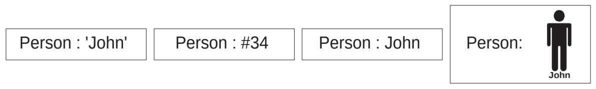

Fig. 13. Different and equivalent notations of John.

4.2 Notation of Concept Type

In the literature concept types, when used as concepts, are denoted with the type label in the referent field. Figure 14 presents the meta level conceptual graph describing the concept that stands for the concept type Person. The concept `[ConceptType:#15]` has a referent symbolized by `#15` The referent represents the type Person that is symbolized by the symbol `Person`.

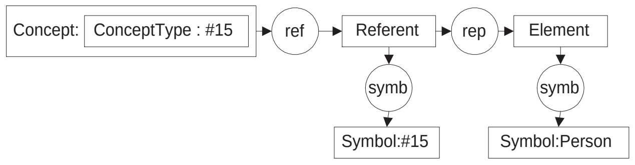

Fig. 14. The meta conceptual graph for notation of the type Person.

By applying the notation rule the referent `#15` may be replaced by the symbol that stands for the type. Figure 15 shows the two possible notations of the concept type.

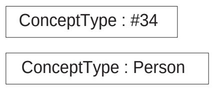

Fig. 15. Different equivalent notations of the type Person.

4.3 Notation of Graph

Concepts that represent conceptual graphs are denoted with the type `Graph` in the type field and the graph itself in the referent field. Figure 16 presents the meta level conceptual graph describing the concept that stands for the graph `[Cat: Garfield]`→`(eat)`→`[Lasagna]`. The concept `[Graph:#72]` has a referent symbolized by `#72`. The referent represents the graph that is symbolized by a symbol `GarfieldMeal`, a linear form and a first order logic form of the graph.

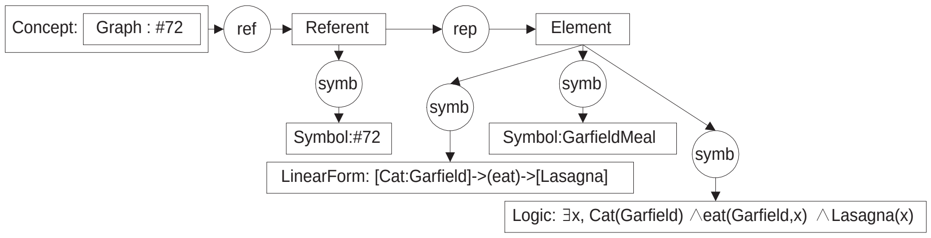
Concept:
Graph : #72
ref
Referent
rep
Element
symb
symb
symb
symb
Symbol:#72
Symbol:GarfieldMeal
LinearForm: [Cat:Garfield]->(eat)->[Lasagna]
Logic: ∃x, Cat(Garfield) ∧eat(Garfield,x) ∧Lasagna(x)

Fig. 16. The meta conceptual graph for notation of the Garfield meal.

By applying the notation rule the referent `#72` may be replaced by any of the symbols that stands for the graph. Figure 17 shows possible notations of the same graph.

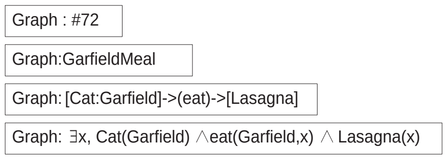
Graph : #72
Graph:GarfieldMeal
Graph: [Cat:Garfield]->(eat)->[Lasagna]
Graph: ∃x, Cat(Garfield) ∧eat(Garfield,x) ∧ Lasagna(x)

Fig. 17. Different equivalent notations of the graph Garfield Meal.

Using this notation mechanism, we can define named elements. We will write `[Graph : GarfieldMeal [Cat: Garfield]`→`(eat)`→`[Lasagna]]` to state that `GarfieldMeal` and `[Cat: Garfield]`→`(eat)`→`[Lasagna]` are two symbols of the graph. Any reference to the graph may be subsequently done by the concept `[Graph : GarfieldMeal]`. For example, this mechanism may be apply with situations; it allows to name situation and avoid to repeat every time the graph that describes the situation when we make a reference to this situation.

5 Meta Level to Data Level and Data Level to Entities

This section introduces the function that links a concept to the entity or entities it represents.

In [6] Sowa describes a mapping between the meta level and the data level. To translate a meta level statement into a data level statement, Sowa introduces two functions τ and ρ. The function τ translates a referent name into a type label. The example below illustrates the use of the function τ. The meta level statement: t_1 is a subtype of t_2 is transformed into a data level statement: every t_1 is a t_2 .

IF : [Type:*t_1]->(subt)->[Type:*t_2]
THEN : [τt_1 : ∀*x] [τt_2 :?x]

The function ρ has the same behavior as τ on relation types and relations; it translates the name of a relation into a relation type label. The example above presents the translation rule from the meta level to the data level in Entity-Relationship diagram.

IF : [Type:*t_1]->(arg1)->[Relation:*r]<-(arg2)<-[Type:*t_2]
THEN : [τt_1]->(ρr)->[τt_2]

5.1 Function ω

More generally, we need a function to access the entity represented by a concept in order to use it. For example, we would like to access the image represented by the concept `[Drawing : BeautifulLandscape]` or we would like to be able to manipulate the graph represented by the concept `[Graph:[Cat]->(on)->[Mat]]`.

Let us define ω as such a function.

Definition 5. *The function ω is defined over $\mathcal{C} \rightarrow \mathcal{E}$ where $\mathcal{C}$ is the set of concepts that represent entities of the system and $\mathcal{E}$ is the set of all referenced elements (internal and external elements).*

Applied on a concept the function ω returns the entity represented by the concept. Obviously, the function is defined on the set of concepts that represent entities of the system.

ω([Drawing : BeautifulLandscape]) =
ω([Graph : [Cat]->(on)->[Mat]]) = [Cat]->(on)->[Mat]

5.2 ω versus τ and ρ

Using the function ω, the above example using τ may be rewritten as follows:

IF : [Type:*t_1]->(subt)->[Type:*t_2]
THEN : [ω([Type:*t_1]):∀*x] [ω([Type:*t_2]):?x]

where ω([Type:*t_1]) returns the entity represented by the concept [Type:*t_1] that is the type represented by t_1 and ω([Type:*t_2]) returns the entity represented by the concept [Type:*t_2] that is the type represented by t_2.

To show the equivalence with τ and ρ if we replace ω([Type:*t_1]) by ωt_1. The rule becomes:

IF : [Type:*t_1]->(subt)->[Type:*t_2]

```
THEN : [ωt₁:∀*x] [ωt₂:?x]
```

In a same way, the translation rule for an E-R diagram becomes:

```
IF : [Type:*t₁]->(arg1)->[Relation:*r]<-(arg2)<-[Type:*t₂]
THEN : [ωt₁]->(ωr)->[ωt₂]
```

5.3 Function ω and metamodeling

The function ω maps a higher level to a lower level. To show the power and the use of the function ω, we give below four examples using the function. Using ω we give the definition of the transitivity, symmetry and anti-symmetry properties of a relation. Such definition uses generic relation types at the higher and lower level in specification graphs. So before giving the associated definitions, we present the notation of co-referenced types that are mapped from a higher level to a lower level.

Concepts that represent co-referenced concepts are denoted with the same symbol in the referent field preceded by an asterisk and by a question mark. Figure 18 presents the meta level conceptual graph describing two co-referenced concepts that stand for an unidentified Concept Type[5].

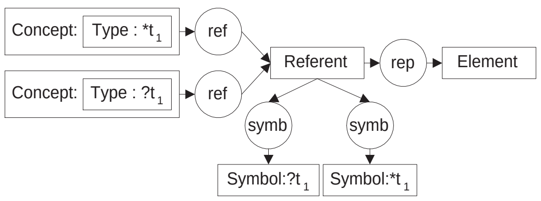

Fig. 18. The meta conceptual graph for notation of coreferenced concepts.

Two concepts that are linked by a coreference link are abstractions of the same element. Therefore they have the same referent that represents the unidentified element. Applied on these concepts the function ω returns the unidentified element that is represented by the referent. For simplicity reasons, ω(`[Type:?t₁]`) will be denoted `?t₁` and ω(`[Type:*t₁]`) will be denoted `*t₁`.

Transitivity. A relationship $\mathcal{R}$ is transitive if and only if :

$$x\mathcal{R}y \wedge y\mathcal{R}z \Rightarrow x\mathcal{R}z \tag{1}$$

Figure 19 presents the graph that defines a transitive relationship at the meta level.

[5] In a coreference link the defining concept is denoted with an asterisk and the bound concept is denoted with a question mark.

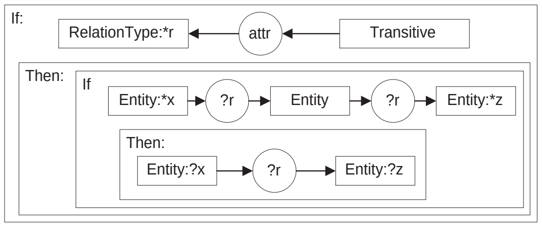

Fig. 19. Definition rule of a transitive relationship.

Symmetry. A relationship $\mathcal{R}$ is symmetrical if and only if :

$$x\mathcal{R}y \Rightarrow y\mathcal{R}x \tag{2}$$

Figure 20 presents the graph that defines a symmetrical relationship.

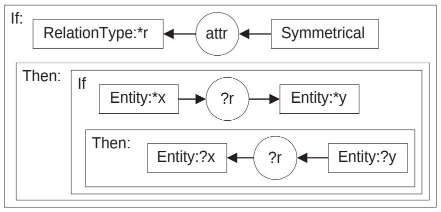

Fig. 20. Definition rule of a symmetrical relationship.

Anti-Symmetry. A relationship $\mathcal{R}$ is anti-symmetrical if and only if :

$$x\mathcal{R}y \wedge y\mathcal{R}x \Rightarrow x = y \tag{3}$$

which is equivalent to

$$\neg(x\mathcal{R}y \wedge y\mathcal{R}x \wedge x \neq y) \tag{4}$$

Figure 21 presents the graph that defines an anti-symmetrical relationship[6].

6 Conclusion

In this paper, we introduced an approach to clarify the semantics, notation, and manipulation of concepts in CG language. This approach uses metamodeling constructs based on CG language. This demonstrates that conceptual graphs may be used in metamodeling activities. The function ω that maps a higher level to a lower level allows the manipulation of concepts from different levels.

[6] Note: two different boxes represent two different concepts

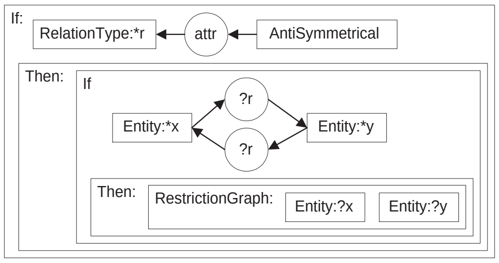

Fig. 21. Definition rule of an anti-symmetrical relationship.

With a simple problem, the representation of concepts, we showed that through the formal definition of a meta level and of mapping functions from one level to the next, we can represent in a uniform way higher level and lower level and navigate between them. There is an obvious need for a complete theory of metamodeling in the CG formalism. We are currently developing such a theory.

References

[1] NCITS.T2 Committee. *Conceptual Graph Standard - draft proposed American National Standard*, 1999.

[2] J. Esch. Contexts, Canons and Coreferent Types. In W. Tepfenhart, J. Dick, and J. Sowa, editors, *Proceedings of the Second International Conference on Conceptual Structures, ICCS'94*, pages 185–195. Springer-Verlag, 1994.

[3] O. Gerbé. Conceptual Graphs for Corporate Knowledge Repositories. In *Proceedings of 5th International Conference on Conceptual Structures*, pages 474–488, 1997.

[4] O. Gerbé and al. *Macroscope Architecture: Architecture of DMR Repository*. DMR Consulting Group Inc., 1994.

[5] O. Gerbé, R. Keller, and G. Mineau. Conceptual Graphs for Representing Business Processes in Corporate Memories. In *Proceedings of the sixth international Conference on Conceptual Structures (ICCS'98)*. Springer-Verlag, 1998.

[6] J. Sowa. Relating Diagrams to Logic. In John F. Sowa (Eds.) Guy W. Mineau, Bernard Moulin, editor, *Conceptual Graphs for Knowledge Representation, ICCS '93*, volume 699, pages 1–35. Springer-Verlag, 1993.

[7] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.

[8] M. Wermelinger. Conceptual Graphs and First-Order Logic. In G. Ellis, R. Levinson, W. Rich, and J. Sowa, editors, *Proceedings of the Third International Conference on Conceptual Structures, ICCS'95*, pages 323–337. Springler-Verlag, 1995.