Parsing Natural Language Sentences to Conceptual Graphs

Iulian Goriac and Cornelius Croitoru

"Al. I. Cuza" University of Iaşi, România
{iulian.goriac,croitoru}@info.uaic.ro

Abstract. Two contributions to the theory of simple conceptual graphs are proposed in this paper. The first is a reformulation of the definition of CG, allowing each concept node to have more than just one elementary concept type, without altering the existing theory. The second one involves reasoning with CGs. A new class of objects is proposed, the procedures, capable of operating any kind of elementary transformation upon the graph structure. Procedures are designed to be used without any modification to the current software tools and are proven to be a sort of generalization to the simple graph rules. These refinements are introduced in order to make the CG formalism more suitable for the representation and high level processing of the natural language.

1 Introduction

A very interesting problem, with many practical applications, is that of Word Sense Disambiguation (WSD). Recent research, summarized in [12], show that graph based semantic representations of texts issue some of the best results when dealing with the unsupervised WDS problem. However, semantic representations can have various other applications, like building question answering systems or even performing argumentation as described in [6].

The intention of the theory presented here is to contribute to the development of a graph based knowledge representation system for the semantics of a natural language discourse. The purpose is not to offer a definite solution to this problem but to harness the representational capabilities of the conceptual graphs and, where appropriate, to enrich the expressiveness of such structures. Attention is paid not only to the static (declarative) aspect of the representations but also to the way the explicitly stored facts are used to obtain implicit affirmations via reasoning.

The main idea is to automatically create a conceptual graph knowledge base (CGKB) using a simple text as input, by employing widely used NLP tools (tokenizer, POS-tagger, lemmatizer, parser). After building a conceptual graph for every sentence in the discourse, different lexical resources (like WordNet, VerbNet, FrameNet etc.) can be used to perform WSD and/or to identify the semantic roles of every word or group of words in a sentence (the actor, the way the action performed, etc.) A further step would be to join the individual

graphs into a larger one by identifying the semantic co-references (e.g. by means of an anaphora resolution process) thus obtaining a unitary graph representation of a discourse. User questions can later be transformed into query graphs and answered by relying on the CG projection mechanism. An application, serving as a proof of concept for the described system was built to support this research and it can be found at *http://www.purl.org/net/wonderland*. This paper, however, will only presented the fundamental theoretical notions used to develop such an application.

The paper is divided into two parts concerning representational aspects of simple conceptual graphs and reasoning aspects. Each chapter is split into a graph theory section (presenting an extension of the existing theory) and a computational linguistic related section (usually underlining some practical aspects). **Chapter 2** redefines simple conceptual graphs allowing each concept node to be attached more than one concept type and, as a consequence of that, in the second section, presents a simple, intuitive process by means of which all the information provided by the NLP tools is represented in a simple CG. **Chapter 3** briefly describes conceptual graph knowledge bases and some of their components. Special attention is paid to the simple graph rules that allow the deduction of facts not explicitly stored in the KB. Some limitations of these rules are presented when facing NLP related tasks and a new kind of inference tool, the procedures, is proposed. Procedures can be used to operate a large number of transformations upon a conceptual graph. Finally procedures are proven to be a generalization of the simple graph rules. The last chapter summarizes the key aspects of the paper and suggests some possible further developments.

2 Simple graphs

The theory of conceptual graphs was initially developed by John Sowa as *"a knowledge representation language based on linguistics, psychology, and philosophy"*[2]. A number of researchers further extended and refined the initial theory by integrating it with the 'standard' graph theory and first order logic (see [2][4][1]). This paper explores some aspects of the simple conceptual graphs theory in order to increase its expressiveness and to underline some algorithmic advantages that will be further used in building and operating upon semantic representations of natural language sentences.

2.1 Extending the definition of simple conceptual graphs

So far, the simple conceptual graphs were defined with a unique concept type for every concept node in the graph. Without violating this constraint, and thus preserving the theoretical acquisitions obtained so far, we show how more than one basic/elementary concept type can be attributed to any single concept node. The main distinction is between *concept type* and *elementary concept type*. The *support* of the conceptual graph includes a hierarchy of *elementary concept types* and to every concept node a non-empty set of *elementary concept types* will be

attached as *concept type*. In order to accommodate this transition, based on the original *subsumption relation* between *elementary concept types*, a definition of the *subsumtion relation* between the new *concept types* is defined.

Considering the above observation we propose the following formal definitions for the *support* and the *simple conceptual graphs*:

Definition 1. *Let $f : A \to B$ be a function:*

- $Dom(f) = \{x \mid (x \in A) \wedge (f(x) \in B)\}$
- $Cod(f) = \{y \mid (y \in B) \wedge (\exists x \in A)(f(x) = y)\}$

Definition 2. *$\hat{\mathcal{P}}(M)$ is the power set of set M without the empty set: $\hat{\mathcal{P}}(M) = \mathcal{P}(M) \setminus \emptyset$.*

If M is a set of elementary concept types $\hat{\mathcal{P}}(M)$ will be the set of concept types.

Definition 3. *The **support** of a conceptual graph is a 4-tuple $S = (T_C, T_R, \mathcal{I}, *)$ where:*

- *T_C is a finite partially ordered set $(T_C, \leq)$ of **elementary concept types** with **top** ($\top_C$) the unique greatest element, also called the **universal (elementary) type**.*
  *$\leq$ is the **subsumtion** or **is-a-kind-of** relation defined over the elements of T_C where $(\forall x, y \in T_C)$ $x \leq y$ means that x is a **subtype** of y or that y **subsums** x; x may substitute y in any context.*
  T_C supports 'multiple inheritance', meaning that x can be declared a subtype for all the elements $y \in ST$, $ST \in \hat{\mathcal{P}}(T_C)$, if $(\forall y)\neg(y \leq x)$.
- *$\mathbf{T_C} = \hat{\mathcal{P}}(T_C)$ is the set of **concept types**, obviously finite, for which the **subsumtion** relation is defined by:*
  $(\forall \mathbf{x}, \mathbf{y} \in \mathbf{T_C})$ $(\mathbf{x} \leq \mathbf{y}) \Leftrightarrow (\forall y \in \mathbf{y})(\exists x \in \mathbf{x})(x \leq y)$.
  *Thus $(\mathbf{T_C}, \leq)$ is a partially ordered set with the unique **universal type** $\top_{\mathbf{T_C}} = \{\top_C\}$.*
- *T_R is a finite set of **relation types**. Each element in T_R has an **arity** k (a non-zero positive integer). The arity is used to partition T_R into k classes $T_1, T_2, \ldots, T_k$. Each class T_i $(i = \overline{1,k})$ is a partially ordered set $(T_i, \leq)$, only allowing 'single inheritance', with $\top_i$ the unique greatest element.*
  *Every relation type $rt \in T_i$ has a **signature** $\sigma \in \underbrace{\mathbf{T_C} \times \ldots \times \mathbf{T_C}}_{i \text{ times}}$ used to restrict the types of **arguments** any relation type may have. Specifically, the concept type of argument $j = \overline{1,i}$ of a relation of type rt must be a subtype of the concept type of the argument j of the same relation type.*
  *The **subsumtion** relation between relation types is defined by taking into account the subsumption relation between signatures:*
  $(\sigma_1 \leq \sigma_2) \Leftrightarrow (|\sigma_1| = |\sigma_2|) \wedge (\forall j, 1 \leq j \leq |\sigma_1|)(\sigma_1^j \leq \sigma_2^j)$, where σ^j represents the element on position j in σ.

– $\mathcal{I}$ *is a set of* ***individual markers*** *used to refer concept nodes.*

– $*$ *is special marker used to indicate* ***generic concept nodes*** *and is called* ***generic marker****.*

– *The* ***subsumtion*** *relation between markers is defined by:*
  - $* \leq *$
  - $\forall m \in \mathcal{I}$, $(m \leq *) \wedge \neg(* \leq m)$
  - $\forall m, n \in \mathcal{I}$, $(m \leq n) \Leftrightarrow (m = n)$

– T_C, T_R, $\mathcal{I}$, $\{*\}$ *are all disjoint 2 - 2.*

Notice that if we enforce every concept type to contain only one elementary concept type, the concept type hierarchy is identical to the one used by in the previous definitions.

Definition 4. *An* ***ordered bipartite graph*** *is a 3-tuple* $G = (V_C, V_R; E_G; l)$ *where:*

– $(V_C, V_R; E_G)$ *is a bipartite graph,* $\{V_C, V_R\}$ *is the explicit bipartition of the vertices and* E_G *is the set of edges.*

– l *is an bijective* ***labelling function*** *mapping the edges adjacent to every* $r \in V_R$ *to* $\{1, 2, \ldots, i\}$ *where* i *is the degree of* r*.*

– $N_G^i(r)$ *is the* i^{th} *neighbour of* r *in* G *indicated by* l*. Notice that because* G *is bipartite* $(\forall i, r)$ $N_G^i(r) \in V_C$*.*

Definition 5. *A* ***simple conceptual graph (SCG)*** *is a 3-tuple* $SG = (S, G, \lambda)$ *where:*

– $S = (T_C, T_R, \mathcal{I}, *)$ *is the support of the conceptual graph.*

– $G = (V_C, V_R; E_G; l)$ *is an ordered bipartite graph where:*
  - $V_C \neq \emptyset$ *is a set of elements called* ***concepts****.*
  - V_R *is a set of elements called* ***relations****.*

– λ *is a function that labels all the vertices of* G*:*
  - *each concept node* $c \in V_C$ *is attributed:*
    - *a concept type* $\mathbf{ct} \in \mathbf{T_C}$ *and only one,*
    - *a marker from* $\mathcal{I} \cup \{*\}$ *and only one:*
      - *if the marker is* $*$ *then the concept is called* ***generic****,*
      - *if the marker is from* $\mathcal{I}$ *the concept is called* ***individual****.*
  - *a relation type* $rt \in T_R$ *is associated to every relation node* $r \in V_R$ *(the assignment must satisfy the signature of* rt*).*

Definition 6. λ *can be used to define a* ***subsumtion*** *relation between the vertices of SCGs defined over the same support:*

– *if* r_1 *and* r_2 *are relation nodes* $(r_1 \leq r_2) \Leftrightarrow (\lambda(r_1) \leq \lambda(r_2))$*.*

– *if* c_1 *and* c_2 *are concept nodes* $(c_1 \leq c_2) \Leftrightarrow (\lambda(c_1) \leq \lambda(c_2))$
a.k.a. $(typeof(c_1) \leq typeof(c_2)) \wedge (markerof(c_1) \leq markerof(c_2))$.

The following definition introduces the main mechanism used for reasoning with CG.

Definition 7. *Considering two conceptual graphs* $SX = (S, X, \lambda_X)$ *and* $SY = (S, Y, \lambda_Y)$ *defined over the same support* S*, a* ***projection*** *from* SY *to* SX *is an application* π *that maps all the vertices of* Y *to vertices of* X *satisfying the following constraints:*

– $\pi(V_C(Y)) \subseteq V_C(X)$ *and* $\pi(V_R(Y)) \subseteq V_R(X)$
– $\forall c \in V_C(Y)$ *and* $\forall r \in V_R(Y)$; $(N^i_Y(r) = c) \Rightarrow (N^i_X(\pi(r)) = \pi(c))$
– $\forall v \in V_C(Y) \cup V_R(Y)$, $\lambda_X(\pi(v)) \leq \lambda_Y(v)$.

Definition 8. *If at least one projection can be found from* SG *to* SH *it is said that* SG ***subsumes*** SH*:* $SH \leq SG$ *or* $SG \geq SH$.

Summarizing, a projection is a type aware conceptual graphs morphism and, of course, there can be more than one distinct projection from one graph to another. The projection finding algorithm is NP-complete.

2.2 Obtaining SCG representations from parsing natural language sentences

Our purpose is to build CG representations for the natural language sentences. We chose to perform a strict Davidsonian reification[8] based on two essential tools used in NLP: the part of speech tagger (POS-tagger) to assign the concept types and the dependency parser to identify the relations. To illustrate the process of building an SCG representation we will use the Stanford Parser[10] and the sentence:

```
The cat is on the mat.
```

We will create one SCG for every sentence and all graphs will be sharing the same support. The first step is to split the sentence into lexical units (tokens) with a *tokenizer*.

```
The/1 cat/2 is/3 on/4 the/5 mat/6 ./7
```

Next, the POS-tagger will perform the morpho-syntactical analysis by attaching tags to every token indicating different grammatical attributes: the part of speech (noun, verb, pronoun, adjective, adverb ...), grammatical number (singular, plural), person ($1^{st}, 2^{nd}, 3^{rd}$), degree (comparative, superlative), etc. The following example uses the Penn tagset[11]:

```
The/DT cat/NN is/VBZ on/IN the/DT mat/NN ./.
```

Further, the relations between the tokens are extracted by using a dependency parser. The kind of information extracted is: the subject of the sentence, the object, the determiners and so on. A complete dependency type hierarchy reference can be found in [5].

```
det(cat-2, The-1)
nsubj(is-3, cat-2)
prep(is-3, on-4)
det(mat-6, the-5)
pobj(on-4, mat-6)
```

Given the information above, building a support is quite straightforward: create a flat concept type hierarchy using the Penn tags, create a relation type hierarchy using Stanford typed dependencies and use the normalized form of every token (*lemma*) as markers. This approach, however, has a severe limitation: it can't take advantage of the individual pieces of information contained by the POS-tag. For instance, `VBZ` indicates a 3^{rd} person singular indicative present verb. If we would consider to test, by using the projection mechanism, only the grammatical number of the concept node we would have to create a large set of query graphs in order to cover all the possibilities (grammatical number is also an attribute for the noun or pronoun and they have different unrelated tags/concept types). To reduce this effort while maintaining all the information attached to every token/concept node the solution is to split the tag information in basic units/elementary concept types and attach all the individual pieces to the same concept node. **This is the main reason for introducing the definition in the previous section.**

Looking for an appropriate set of elementary concept types to use for normalizing the information offered by tags we decided to use the names and the hierarchy proposed by the ISOCat[7] reference implementation of the ISO12620 standard.

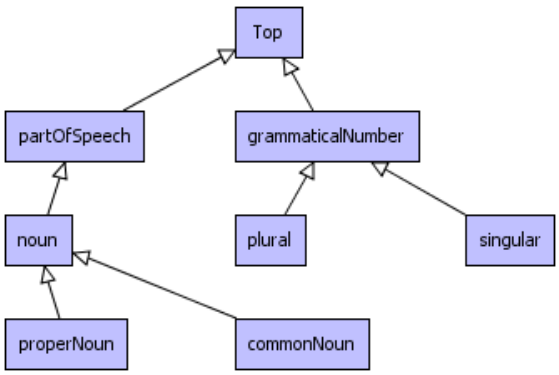

Fig. 1. A small subset of the ISOCat derived elementary concept type hierarchy

Combining the ISOCat derived concept type hierarchy and Stanford typed dependencies derived relation type hierarchy a representation of a natural language sentence as an SCG is presented[1] in **fig. 2**.

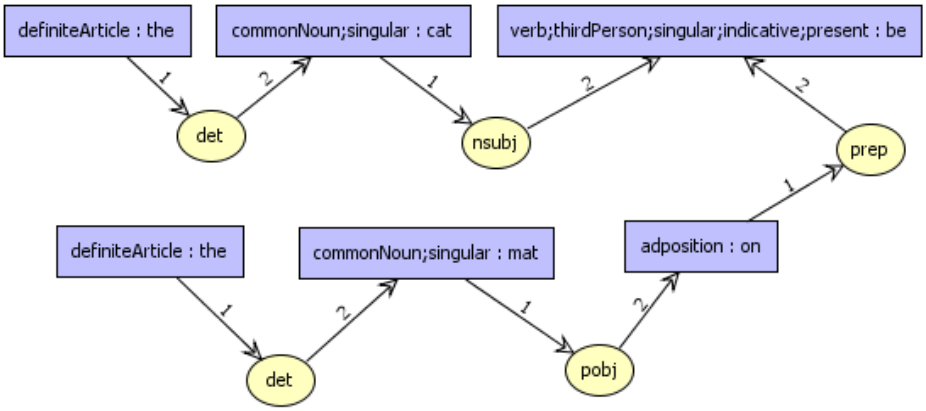

Fig. 2. SCG representation for the sentence *"The cat is on the mat"*

3 Rules and procedures

In [1] the basic elements of a conceptual graph (CG) knowledge base (KB) are summarized. The idea behind such a system is to find the truth value of a fact without necessarily storing it explicitly. There are many aspects the design of a KB should consider and the CGKBs make no exception. The one that we will focus here is the very problem we mentioned before: how can the truth value of a fact be inferred by using the KB stored elements and specific deduction operations. This process is called *reasoning*.

A CGKB contains certain classes of objects. One of them is the class of **facts**, grouping SCGs storing the knowledge in a declarative form. Good examples of such fact graphs are the ones obtained by processing natural language sentences as previously described.

Essential for the CGKBs' reasoning process is the projection mechanism defined in the previous section. Basically a fact graph K can be deduced from a CGKB if there is at least a projection from K to a fact graph in the KB.

Another class of CGKB objects is the class of **rules**. They express knowledge of form "if A is present than B can be added"[1]. Formally rules are defined as coloured CGs.

[1] The images were created with CoGui *http://www.lirmm.fr/cogui/*

Definition 9. *A **coloured simple graph** is an SCG with one of the two colours* $\{0, 1\}$ *attached to every vertex. By* $K_{(i)}$ *is indicated the sub-graph with all its vertices coloured* i*. Using this notation,* $K_{(0)}$ *must be an SCG.*

Definition 10. *A **simple graph rule** is a coloured SCG.* $K_{(0)}$ *is its **hypothesis** and* $K_{(1)}$ *its **conclusion**.*

The rules play a significant role in deduction via the process of *rule application*: "a graph transformation based upon projection"[1].

Definition 11. *If* G *is an SCG and* R *is a rule, both defined over the same support,* R ***can be applied*** *to* G *if there is a projection from the hypothesis of* R *to* G*. Further, the **application** process consists in adding (a copy of) the elements of the conclusion of* R *to* G *(1-coloured vertices and all incident edges) replacing however the hypothesis vertices with their counterparts returned by the projection. Thus a new SCG is obtained. By applying the rule once the resulted graph is an **immediate derivation**. A **derivation** is any of the graphs obtained by indefinitely chaining the application of a rule.*

Notice that the derivation of a graph by using a number of rules can be subsumed by a larger group of fact graphs, allowing the rules to increase the expressiveness of the KB. The cost is that the deduction problem is now undecidable.

3.1 Extending the functionality of the rules

One of the limitation of simple graph rules is that every application increases the complexity of the initial graph. They can only add concept nodes or relation nodes to a SCG, but neither can they remove any of those nor change their types. The latter transformations can reduce the complexity of the fact graphs in a CGKB allowing the deduction to be, if not simpler in terms of algorithmic complexity, at least more efficient in execution time.

Therefore, we propose a new class of objects to be included in a CGKB: the **procedures**. They express knowledge of form "if A is present than B transformation can be operated".

In order to formally define the procedures, the following extension needs to be made to the definition of the support.

Definition 12. *A **procedural support** or a **#support** is a 3-tuple* $\#S = (S, \#T_C, \#*)$ *where:*

- S *is a support (as in **def. 3**).*
- $\#T_C = \{\#-, \#0, \#+\}$ *is a set of elementary concept types supported as arguments by any relation type.*
- $\#*$ *is an individual marker.*
- *each* $\#$ *symbol is different from the ones defined in* S *and have a special semantic.*

Definition 13. *A **procedure** is a 4-tuple $P = (S, A, C, \rho)$ where:*

- *S is a support (as in **def. 3**).*
- *A is an SCG defined over S and called the **antecedent** of P.*
- *C is an SCG defined over $\#S$ and called the **consequent** of P.*
- *$\rho : V_C(C) \to V_C(A)$ bijectively maps a (possibly empty) subset of $V_C(C)$ to $V_C(A)$.*
- *only the concept nodes $c \in Dom(\rho)$ can be marked with $\#*$ and their concept type will necessarily include one and only one element from $\#T_C$.*

Even though a procedural derivation based deduction mechanism can be envisioned, procedures were not intended for such use. They are mainly designed to guard the KB against 'trivial complexity', that is to simplify the fact graphs before they are stored in the KB. The operational semantic of procedure application is defined in the following definition (14).

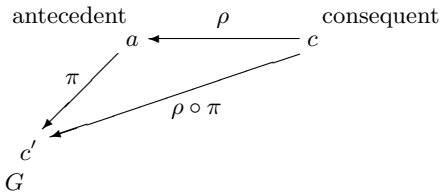

Fig. 3. Mapping the consequent of a procedure to a simple graph via a projection π

Definition 14. *If G is an SCG and P is a procedure, both defined over the same support S, P **can be applied** to G if there is a projection from the antecedent of P to G. If such a projection is found, say π, the **application** of the procedure is performed by executing the following steps:*

1. *DELETE from G all vertices not in $Cod(\rho \circ \pi) \cap Cod(\pi)$.*
2. *$\forall (c, c') \in \{(c, c') \mid (c \in Dom(\rho \circ \pi)) \wedge (c' = (\rho \circ \pi)(c))\}$ the concept node c' is UPDATEd:*
   - *UPDATE the concept type of c'*
     - *if $\#+ \in typeof(c)$: $typeof(c') \leftarrow typeof(c') \cup typeof(c) \setminus \{\#+\}$*
     - *if $\#0 \in typeof(c)$: $typeof(c') \leftarrow typeof(c)$*
     - *if $\#- \in typeof(c)$: $typeof(c') \leftarrow typeof(c') \setminus typeof(c)$*
   - *UPDATE the marker of c'*
     - *if $markerof(c) = *$: $markerof(c') \leftarrow markerof(c')$*
     - *if $markerof(c) \in \mathcal{I}$: $markerof(c') \leftarrow markerof(c)$*
     - *if $markerof(c) = \#*$: $markerof(c') \leftarrow *$*

3. *for each concept node* $c \notin Dom(\rho)$ *a copy of* c*, say* c' *is INSERTed into* G *and* $(\rho \circ \pi)(c) = c'$ *is defined.*
4. *for each relation node* r *in the consequent of* P *a copy of* r*,* r'*, is INSERTed into* G *and* $N^i_G(r') = (\rho \circ \pi)(N^i_G(r)),\ i = \overline{1, arityof(r)}$.

Theorem 1. *For every simple graph rule there is a procedure with the same operational semantic (that will operate the same transformation).*

Proof. For a rule R that can be applied to a graph G the following procedure P is created:

1. the antecedent of P is a copy of $R_{(0)}$.
2. the consequent of P is a copy of R where every concept node from $R_{(0)}$ is assigned the concept type #0 and marker $*$.
3. ρ is created by mapping every copy of a concept node from $R_{(0)}$ in the antecedent of P to the copy of the same concept node in the consequent of P.

Obviously the procedure P can be applied whenever the rule R can be applied. R will add the elements from $R_{(1)}$ to G. P will not delete any concept node from G since all concept nodes from antecedent are mapped by ρ and the mapping will not alter neither their types not their markers. The copy of $R_{(1)}$ will be added to G and connected in the same way as it would be done for the corresponding rule.

□

From **Theorem 1** it can be drawn that procedures may be accounted as generalizations of the simple graph rules. They are an extension to the current formalism because they define ways to operate all kinds of transformations upon an SCG (adding and removing relation and concept nodes, altering the types of the concept nodes and their markers) while maintaining the entire previous functionality. A concrete example of how the procedures operate will be presented below.

3.2 Using procedures to operate upon the SCG representations

The procedures were defined in such a way that the current tools used to handle conceptual graphs can take advantage of them without any modification[2].

The procedure in **fig. 4** applied two times successively to the graph in **fig. 2** yields the simpler derivation in **fig. 5**. The definite article concept nodes and the adjacent relation nodes were eliminated and replaced by a new elementary concept type 'definite' attached to the corresponding noun representing concept nodes.

[2] some client side code should be added thow

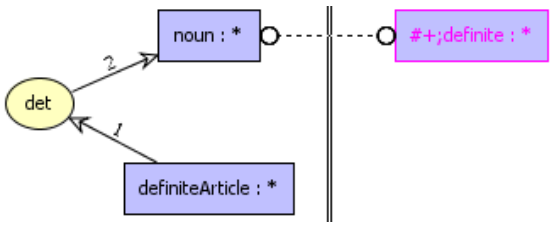

Fig. 4. A procedure edited with CoGui. On the left side of the vertical line is the antecedent and on the right side the consequent.

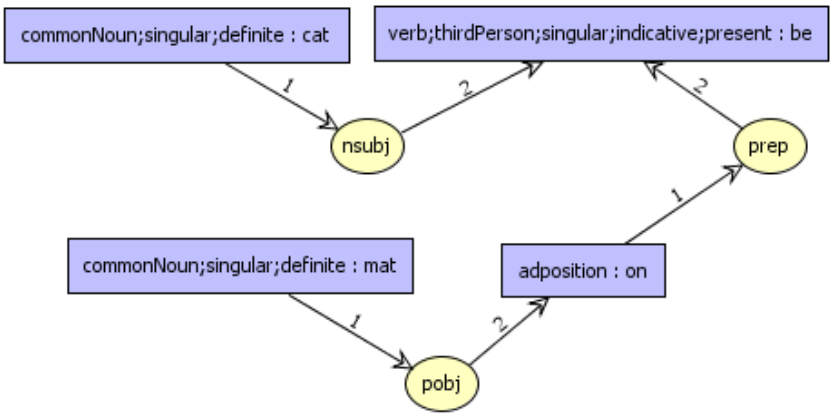

Fig. 5. A derivation of the SCG in **fig. 2** by using the procedure in **fig. 4**

4 Conclusions

This is mainly a theoretical endeavour. What we did was to redefine the simple conceptual graphs in order to make them more suitable for NLP tasks. Using the representational capabilities of the newly defined SCGs, we presented a simple and intuitive way to organize all the information offered by essential NLP processing tools (tokenizer, POS-tagger, lemmatizer, dependency parser) into an SCG and thus allowing this information to be operated upon with the wide variety of tools developed by graph theory. Because the representations are not particularly useful *per se*, we also provided a class of specific tools, the procedures, especially designed to operate upon the SCG representations. All these extensions are built on top of the already existing CG theory and their implementation and use require no modification to the already existing software tools related to CG.

Concretely, the first contribution pertains the definition of SCG − particularly the concept type hierarchy of the support. Based on a certain hierarchy of elementary concept types (as used in the previous definition) a new hierar-

chy of concept types is developed, every concept type is defined as a non-empty subset of the the initial elementary concept types set. The partial order relation between concept types is defined according to the ordering of the elementary concept types − therefore by assigning a concept type to a concept node we can actually attach any number of elementary concept types to that node without requiring a new elaboration of the previous theory. This trick is very helpful when it is intended to assign different grammatical attributes (part of speech, gender, number, person, etc.) to the same concept node and to use the projection mechanism to individually differentiate/select between them.

Procedures can be used to operate upon these representations. Procedures are designed to be a generalization of the simple graph rules and can operate a large number of transformations (insertions, deletions and alterations of nodes' types and markers). One of their most important purposes is to simplify the structure of the fact graphs.

Some of our future options for developing this theory lie in field of automatically building procedures capable to perform WSD and thematic role identification by using examples from VerbNet, PropBank and/or FrameNet. A prototype application, using VerbNet, already exists, however the current stage of development does not allow any speculation about its performance. Another option would be to develop a completely natural language based CG editor (by using, for example, Cogitant[3] and LinkGrammar[9]).

References

1. Jean-François Baget and Marie-Laure Mugnier. Extensions of simple conceptual graphs: the complexity of rules and constraints. *J. Artif. Intell. Res. (JAIR)*, 16:425–465, 2002.
2. Michel Chein and Marie-Laure Mugnier. Conceptual graphs: fundamental notions. *Revue d'Intelligence Artificielle*, 6:365–406, 1992.
3. Cogitant. *http://cogitant.sourceforge.net/*.
4. Madalina Croitoru. *Conceptual Graphs at Work: Efficient Reasoning and Applications*. PhD thesis, 2006.
5. Marie-Catherine de Marneffe and Christopher D. Manning. Stanford typed dependencies manual. 2010.
6. Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.
7. ISOCat. *http://www.isocat.org/index.html*.
8. Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition) (Prentice Hall Series in Artificial Intelligence)*. Prentice Hall, 2008.
9. LinkGrammar. *http://www.abisource.com/projects/link-grammar/*.
10. Stanford Parser. *http://nlp.stanford.edu/software/lex-parser.shtml*.
11. Beatrice Santorini. Part-of-speech tagging guidelines for the penn treebank project. 1990.
12. George Tsatsaronis, Iraklis Varlamis, and Kjetil Nørvåg. An experimental study on unsupervised graph-based word sense disambiguation. In *CICLing*, pages 184–198, 2010.