View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Online Research Database In Technology

Proceedings of the International Multiconference on Computer Science and Information Technology pp. 295–298
ISBN 978-83-60810-22-4
ISSN 1896-7094

Ontological Semantics in Modified Categorial Grammar

Bartłomiej Antoni Szymczak
DTU Informatics, Technical University of Denmark
International Language Studies and Computational Linguistics
Copenhagen Business School
Email: bas@imm.dtu.dk

***Abstract*—Categorial Grammar is a well established tool for describing natural language semantics [1]. In the current paper we discuss some of its drawbacks and how it could be extended to overcome them. We use the extended version for deriving ontological semantics from text. A proof-of-concept implementation is also presented.**

I. Introduction

IN THIS paper we extend the usual Categorial Grammar framework in order to achieve more flexibility. We also present how it can be used with an ontological component, which imposes well-formedness restrictions on sentences.

The objective is to integrate formal ontologies with semantic domains. Such an ontology-oriented semantics may be useful e.g. in content-based text search.

Throughout the paper we use classical Church's type theory, $\mathcal{C}$, as presented in [2]. We use the convention that functional types associate to the left, i.e. type $\gamma\beta\alpha$ is the same as $(\gamma\beta)\alpha$, which is also sometimes denoted as $\alpha \rightarrow (\beta \rightarrow \gamma)$.

The work presented here takes place within the SIABO project [3].

II. Ontological semantics

We wish to construct ontological semantics for a fragment of English, following the approach in [4] and [5].

We can represent concepts appearing in a skeleton ontology as constants of type α: $Child_\alpha, Tall_\alpha, Running_\alpha, \ldots$. Those concepts are to be thought of as sets, e.g. $Child_\alpha$ is a set of all imaginable children. $Tall_\alpha$ is the set of all tall objects, thus properties such as "tall" are introduced in the ontology on the par with classes. $Running_\alpha$ is a set of all imaginable actions of running, etc. Introducing nominalized verb forms as concepts in the ontology is in line with the adoption of the Davidsonian view. We also disregard meaning of determiners in the present fragment, as the resulting semantics is intended to be used for content–based text search. We don't want to be specific about α, but in the current example it could be replaced by $o\iota$, which is traditionally used for representing sets. We will represent intersection of concepts using constant $\cap_{\alpha\alpha\alpha}$, e.g. $child \cap tall$ will be represented as $\cap_{\alpha\alpha\alpha} Child_\alpha Tall_\alpha$.

A simple skeleton ontology can be represented using a set of factual clauses of the form:

$$Sub_{o\alpha\alpha} Child_\alpha Person_\alpha$$
$$Sub_{o\alpha\alpha} Person_\alpha Physical_\alpha$$
$$Sub_{o\alpha\alpha} Person_\alpha Animate_\alpha$$
$$Sub_{o\alpha\alpha} Running_\alpha Action_\alpha$$

$Sub_{o\alpha\alpha}$ is a direct descendant relation, representing the lines that are drawn in the Hasse diagrams. $Isa_{o\alpha\alpha}$ can be specified as the (reflexive) transitive closure of $Sub_{o\alpha\alpha}$ in the following way:

$$\forall c_\alpha [Isa_{o\alpha\alpha} c_\alpha c_\alpha]$$
$$\forall c_\alpha \forall a_\alpha \forall p_\alpha [Isa_{o\alpha\alpha} c_\alpha a_\alpha \supset Sub_{o\alpha\alpha} c_\alpha p_\alpha \wedge Isa_{o\alpha\alpha} p_\alpha a_\alpha]$$

We use constants of type ρ to represent roles (binary relations), e.g. Agt_ρ, Loc_ρ. We can use Peirce product [6] to create compound concepts. We use constant $:_{\alpha\alpha\rho}$ for that purpose, e.g. $agt : child$ will be represented as $:_{\alpha\alpha\rho} Agt_\rho Child_\alpha$. Such a concept formation is well-known from Description Logics, where it would be represented as $\forall agt.child$.

For a sample sentence "The tall kid runs", we would like to derive the following ontologico–algebraic meaning:

$$running \cap agt : (child \cap tall)$$

This can be represented in type theory as a wff_α:

$$\cap_{\alpha\alpha\alpha} Running_\alpha [:_{\alpha\alpha\rho} Agt_\rho [\cap_{\alpha\alpha\alpha} Child_\alpha Tall_\alpha]]$$

We shall extend the specification of subsumption relation to accomodate for the intersection:

$$\forall x_\alpha \forall y_\alpha \forall z_\alpha [[Isa_{o\alpha\alpha} [\cap_{\alpha\alpha\alpha} x_\alpha y_\alpha] z_\alpha] \supset [Isa_{o\alpha\alpha} x_\alpha z_\alpha]]$$
$$\forall x_\alpha \forall y_\alpha \forall z_\alpha [[Isa_{o\alpha\alpha} [\cap_{\alpha\alpha\alpha} x_\alpha y_\alpha] z_\alpha] \supset [Isa_{o\alpha\alpha} y_\alpha z_\alpha]]$$

III. Modified Categorial Grammar

We introduce type ω. Constants of this type represent words in English, e.g. W_kid_ω, W_tall_ω, W_runs_ω. Notice that we use different names for constants denoting words and those representing concepts, e.g. W_tall_ω and $Tall_\alpha$. In this way they are not confused.

Let us define a new type, say, κ, which we will use for lexical entries. We also define three constants, which act as type constructors:

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on July 30,2010 at 09:55:46 UTC from IEEE Xplore. Restrictions apply.

$$B_{\kappa(o\kappa\kappa)}$$
$$F_{\kappa(o\kappa\kappa)}$$
$$E_{\kappa\alpha}$$

Lexical entries are represented using the predicate constant $Lex_{o\kappa\omega}$. The lexicon consists of the set of factual clauses of the form:

$$Lex_{o\kappa\omega}W\_kid_{\omega}[E_{\kappa\alpha}Child_{\alpha}]$$

The type constructor $E_{\kappa\alpha}$ is used in the lexical entry in case the meaning of a word is some "fixed" ontological concept. The other constructors, $F_{\kappa(o\kappa\kappa)}$ and $B_{\kappa(o\kappa\kappa)}$ are reminiscent of the Categorial Grammar's backward and forward slashes, respectively. A lexical entry for word "tall" might look as follows:

$$Lex_{o\kappa\omega}W\_tall_{\omega}[F_{\kappa(o\kappa\kappa)}P\_tall_{o\kappa\kappa}]$$

In the above, $P\_tall_{o\kappa\kappa}$ is a predicate, which might be defined in the following way:

$$\forall c_{\alpha}[[P\_tall_{o\kappa\kappa}[E_{\kappa\alpha}c_{\alpha}][E_{\kappa\alpha}[\cap_{\alpha\alpha\alpha}tall_{\alpha}c_{\alpha}]]] \supset [Isa_{o\alpha\alpha}c_{\alpha}Physical_{\alpha}]]$$

The novelty in our approach is the use of predicates in meanings of words. In standard Categorial Grammar, lambda terms are used for that purpose. They are combined using β–reduction, with the only provision of categorial agreement. Consider the sentence "vitamin smiles." In traditional CG, "vitamin" has category np and meaning $\mathbf{vitamin}$. The word "smiles" has category $np\backslash s$ and meaning $\lambda x.\mathbf{smile}(x)$. Since the categories fit together, the meanings get combined using β–reduction, and the sentence gets the meaning $\mathbf{smile}(\mathbf{vitamin})$. While the sentence is correct syntactically, it's incorrect from an ontological point of view. Unfortunately, usual CG does not allow us to introduce any ontological restrictions on the semantics. Our approach can reject this sentence, as we can use the following lexical entry for "smiles":

$$Lex_{o\kappa\omega}W\_smiles_{\omega}[B_{\kappa(o\kappa\kappa)}P\_smiles_{o\kappa\kappa}]$$
$$\forall c_{\alpha}[[P\_smiles_{o\kappa\kappa}[E_{\kappa\alpha}c_{\alpha}][E_{\kappa\alpha}[\cap_{\alpha\alpha\alpha}Smiling_{\alpha}[:_{\alpha\alpha\rho}\ Agt_{\rho}c_{\alpha}]]]] \supset [Isa_{o\alpha\alpha}c_{\alpha}Animate_{\alpha}]]$$

The inability of β–reduction to fail has been already recognized as a problem by G. Ben–Avi and N. Francez in [7]. They have introduced a new formalism, which includes "β–reduction for ontologically–well typed λ–terms", among 12 definitions that constitute "the Ontological Lambek Calculus". Our proposal, however, has a few further advantages:

- It's very formal – it's formalized fully within $\mathcal{C}$.
- It's very simple – it consists of a few formulas only.
- It's very flexible – it allows adding arbitrary restrictions, e.g. one might like to use restrictions based on parthood relation rather than subsumption.
- It can be implemented in a straight-forward way, as presented in Section V

For those reasons, rather than using β-reduction, we propose using a general proof system, which is a machinery having inherently the notion of failure at disposal. Now, desired semantic restriction can be handled by non-provability of a certain statement, e.g.:

$$K \nvdash \exists x_{\alpha}[P\_smiles_{o\kappa\kappa}[E_{\kappa\alpha}Vitamin_{\alpha}]x_{\alpha}]$$

In the above K denotes a set of formulas consisting of lexical assertions and rules introduced throughout this paper. So a vitamin cannot smile, but a child certainly can:

$$K \vdash \exists x_{\alpha}[P\_smiles_{o\kappa\kappa}[E_{\kappa\alpha}Child_{\alpha}]x_{\alpha}]$$

Not only are we interested in the provability of such a goal, but we would also like to know what is the resulting semantics. In this case it's: $E_{\kappa\alpha}[\cap_{\alpha\alpha\alpha}Smiling_{\alpha}[:_{\alpha\alpha\rho}\ Agt_{\rho}Child_{\alpha}]]$

In other words:

$$K \vdash [P\_smiles_{o\kappa\kappa}[E_{\kappa\alpha}Child_{\alpha}] [E_{\kappa\alpha}[\cap_{\alpha\alpha\alpha}Smiling_{\alpha} [:_{\alpha\alpha\rho}\ Agt_{\rho}Child_{\alpha}]]]]$$

IV. Elimination rules

Recall that the forward sequent rule of the natural–deduction Lambek Calculus takes the form:

$$\frac{a \Rightarrow \Phi_1 : A/B \qquad b \Rightarrow \Phi_2 : B}{a, b \Rightarrow \Phi_1(\Phi_2) : A}$$

and the backward rule:

$$\frac{a \Rightarrow \Phi_1 : B \qquad b \Rightarrow \Phi_2 : B\backslash A}{a, b \Rightarrow \Phi_2(\Phi_1) : A}$$

In the above a, b is the concatenation of a and b. Notice that the only provision for the elimination to take place is the agreement of syntactic categories. This is because $\Phi_1(\Phi_2)$ is always a well-formed λ-term, which can be β-reduced.

For the ease of reading, let us present the forward elimination rule of the generalized grammar in a similar, though quite informal way:

$$\frac{a_{\theta} \Rightarrow F_{\kappa(o\kappa\kappa)}p_{o\kappa\kappa} \qquad b_{\theta} \Rightarrow j_{\kappa} \qquad K \vdash p_{o\kappa\kappa}j_{\kappa}k_{\kappa}}{a_{\theta}, b_{\theta} \Rightarrow k_{\kappa}}$$

Let us formalize our generalized elimination rules in $\mathcal{C}$. The derivation relation ($\Rightarrow$) can be represented using a constant $R_{o\kappa\theta}$. The sequences of meanings will be represented using a well–known logic representation for lists. For the empty list we use the N_{θ} constant, and the list constructor is represented by $L_{\theta\theta\kappa}$.

The list concatenation can be represented using the list appending well–known from logic programming, though here defined for non–empty lists only:

$$\forall x_{\kappa}\forall y_{\kappa}\forall t_{\theta}[A_{o\theta\theta\theta}[L_{\theta\theta\kappa}x_{\kappa}N_{\theta}] [L_{\theta\theta\kappa}y_{\kappa}t_{\theta}] [L_{\theta\theta\kappa}x_{\kappa}[L_{\theta\theta\kappa}y_{\kappa}t_{\theta}]]]$$
$$\forall h_{\kappa}\forall l_{\theta}\forall m_{\theta}\forall t_{\theta}[[A_{o\theta\theta\theta}[L_{\theta\theta\kappa}h_{\kappa}l_{\theta}]m_{\theta}[L_{\theta\theta\kappa}h_{\kappa}t_{\theta}]] \supset [A_{o\theta\theta\theta}l_{\theta}m_{\theta}t_{\theta}]]$$

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on July 30,2010 at 09:55:46 UTC from IEEE Xplore. Restrictions apply.

The above use of $A_{o\theta\theta\theta}$ asserts that the concatenation of a and b yields g, or g is split into a and b. Now the forward elimination rule can be formalized in $\mathcal{C}$:

$$\begin{aligned}\forall a_\theta \forall b_\theta \forall g_\theta \forall j_\kappa \forall k_\kappa \forall p_{o\kappa\kappa}[R_{o\kappa\theta} g_\theta k_\kappa] \\ \supset [A_{o\theta\theta\theta} a_\theta b_\theta g_\theta] \\ \wedge [R_{o\kappa\theta} a_\theta [F_{\kappa(o\kappa\kappa)} p_{o\kappa\kappa}]] \\ \wedge [R_{o\kappa\theta} b_\theta j_\kappa] \\ \wedge [p_{o\kappa\kappa} j_\kappa k_\kappa]\end{aligned}$$

Similarly, the backward elimination rule:

$$\begin{aligned}\forall a_\theta \forall b_\theta \forall g_\theta \forall j_\kappa \forall k_\kappa \forall p_{o\kappa\kappa}[R_{o\kappa\theta} g_\theta k_\kappa] \\ \supset [A_{o\theta\theta\theta} a_\theta b_\theta g_\theta] \\ \wedge [R_{o\kappa\theta} a_\theta j_\kappa] \\ \wedge [R_{o\kappa\theta} b_\theta [B_{\kappa(o\kappa\kappa)} p_{o\kappa\kappa}]] \\ \wedge [p_{o\kappa\kappa} j_\kappa k_\kappa]\end{aligned}$$

We also need the following grammatical axiom, which expresses that if the list of meanings contains only one element, that element is the resulting meaning:

$$\forall k_\kappa [R_{o\kappa\theta} [L_{\theta\theta\kappa} k_\kappa N_\theta] k_\kappa]$$

The English text can also be represented formally in $\mathcal{C}$. We introduce a new type ζ for that purpose. We will represent the text as a list of words. An empty list of words will be represented by a constant $Tnil_\xi$ and a list constructor by a constant $T_{\xi\xi\omega}$.

V. Implementation outline

Using $\mathcal{C}$ instead of some kind of untyped logic as the underlying formalism has important advantages regarding the implementation. It forces us to think of the type of every formula, subformula and symbol. Thanks to that, the resulting formalization of the grammar is well-suited for implementation in a strongly-typed programming language. Using such a language (e.g. Mercury instead of Prolog) helps avoid very hard-to-find bugs and allows the compiler to generate much faster code.

We are interested in translating the formal specification given so far to a Prolog–like logic programming language, so that we can execute specific queries. For that purpose, we have formalized the grammar in $\mathcal{C}$ using only definite clauses. Furthermore, we perform the following steps:

- We write all the constants in lower–case
- We write all variables in upper–case
- We use `i` for intersection constant and `p` for the Peirce product constant
- We drop the subscripts indicating types in all formulas
- We replace '$\supset$' and '$\wedge$' with '`:-`' and '`,`', respectively.
- We add a period at the end of each formula.
- We replace curried notation of argument application with a non–curried one.
- We simply remove the universal quantification over all variables, as it's implicit.

Following all the given steps results in the program presented below. The result is a perfectly valid program in the Mercury logic programming language. Mercury is however strongly typed, so we need to add a few auxiliary definitions in order to compile it.

The type α can be modeled as follows:

```
:-type alpha ---> child; tall; physical;
  action; person; running;
  smiling; animate;
  ...
  i(alpha,alpha);
  p(role,alpha).
```

We need to add the following type, mode and determinism specification for the subsumption predicate:

```
:-pred sub(alpha::in,alpha::out) is nondet.
```

This tells Mercury that predicate `sub` takes an entity of type α as input, and computes an entity of type α as output. Furthermore, we specify that `sub` is a nondeterministic predicate, meaning that it can compute multiple outputs for one input. We need to provide similar specifications for all predicates in our program.

The factual subsumption database, translated directly from our previous definition in $\mathcal{C}$:

```
sub(child,person).
sub(person,physical).
sub(person,animate).
sub(running,action).
sub(smiling,action).
...
```

The type ρ of roles can be defined in Mercury as:

```
:-type role --->
    tmp ; loc ; prp ; wrt
  ; chr ; cum ; bmo ; cby
  ; cau ; cmp ; pof ; agt
  ; pnt ; src ; rst ; dst
  ; via ; ...
```

Clauses defining the `isa` relation:

```
isa(C,C).
isa(C,A):-sub(C,P),isa(P,A).
isa(i(X,_Y),Z):-isa(X,Z).
isa(i(_X,Y),Z):-isa(Y,Z).
```

The predicates included in lexical entries take the following form in Mercury:

```
p_tall(e(C),e(i(tall,C))):-
  isa(C,physical).

p_runs(e(C),e(i(running,p(agt,C)))):-
  isa(C,animate).

p_smiles(e(C),e(i(smiling,p(agt,C)))):-
  isa(C,animate).

p_the(e(I),e(I)).

...
```

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on July 30,2010 at 09:55:46 UTC from IEEE Xplore. Restrictions apply.

The lexicon, translated to Mercury:

```
lex(w_kid,e(child)).
lex(w_tall,f(p_tall)).
lex(w_runs,b(p_runs)).
lex(w_smiles,b(p_smiles)).
lex(w_the,f(p_the)).
lex(w_a,f(p_the)).
...
```

The definition of non–empty list appending:

```
a(l(X,n),l(Y,T),l(X,l(Y,T))).
a(l(H,L),M,l(H,T)):-a(L,M,T).
```

The rules formalized in $\mathcal{C}$ take the following form in Mercury:

```
r(l(X,n),X).

r(G,E1):-
  a(G1,G2,G),
  r(G1,f(P)),
  r(G2,E2),
  P(E2,E1).

r(G,E1):-
  a(G1,G2,G),
  r(G1,E2),
  r(G2,b(P)),
  P(E2,E1).
```

The rules are the only place in the program, where we use higher–order predicates.

The following predicate assigns a meaning to English text:

```
m(T,C):-
        map_lex(T,L),
        r(L,e(C)).
```

Let us add an auxiliary predicate `work` for testing purposes:

```
work(C):-
  m(t(w_the,
      t(w_tall,
        t(w_kid,
          t(w_smiles,
            tnil)))),C).
```

The `map_lex` predicate is used for lexicon look–up of a list of words.

```
map_lex(tnil,n).
map_lex(t(W,T),l(I,MT)):-
  lex(W,I),map_lex(T,MT).
```

Notice that the program does not require higher order unification, except for the simplest case where a variable is bound to a predefined predicate and such a predicate is called. Most logic programming languages provide a facility for such behaviour. We could change the syntax slightly and turn our code into a valid Prolog program by utilizing the `call` library predicate.

Notice that the program is a direct implementation of the theory, hence quite inefficient. Instead of the presented top–down approach, we could derive the semantics in the bottom–up manner in order to avoid unneeded search.

For the sample query:

```
? work(C).
```

We get the following (exactly one) result:

```
C = i(smiling, p(agt, i(tall, child)))
```

VI. Conclusion

We have presented an extension to Categorial Grammar, which allows constructing arbitrary semantics and enforcing arbitrary semantic restrictions in a very flexible manner. The functional composition and β-reduction are replaced with proof rule application. We have utilized it for constructing ontologico-algebraic meaning using ontological restrictions, dropping at the same time syntactic categories and syntactic restrictions. Finally, a proof-of-concept implementation has been given.

Acknowledgment

I am very grateful to my supervisor, prof. Jørgen Fischer Nilsson, for his most kind help.

References

[1] B. Carpenter, *Type-logical semantics*. Cambridge, MA, USA: MIT Press, 1998.

[2] P. Andrews, "Classical type theory," 2001, peter Andrews. Classical type theory. In Alan Robinson and Andrei Voronkov, editors, Handbook of Automated Reasoning,volume 2, chapter 15, pages 965-1007. North-Holland, 2001. 43.

[3] SIABO.dk, "Siabo project website," `http://www.siabo.dk`, 2008.

[4] P. A. Jensen and J. F. Nilsson, "Ontology-based semantics for prepositions," in *Syntax and Semantics of Prepositions*, ser. Text, Speech and Language Technology, Vol. 29. Springer, 2006.

[5] B. A. Szymczak, "Formal ontologies for semantic text processing," Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2007, supervised by Prof. Jørgen Fischer Nilsson, IMM, DTU. `http://www2.imm.dtu.dk/pubdb/p.php?5399`.

[6] C. Brink, K. Britz, and R. A. Schmidt, "Peirce algebras," *Formal Aspects of Computing*, vol. 6, no. 3, pp. 339–358, April 1994, also available as Research Report MPI-I-92-229, Max-Planck-Institut für Informatik, Saarbrücken, Germany (July 1992), and as Research Report RR 140, Department of Mathematics, University of Cape Town, Cape Town, South Africa (August 1992). An extended abstract appears in Nivat, M., Rattray, C., Rus, T. and Scollo, G. (eds), *Algebraic Methodology and Software Technology (AMAST'93): Proceedings of the 3rd International Conference on Algebraic Methodology and Software Technology*. *Workshops in Computing* Series, Springer-Verlag, London, 165-168 (1994).

[7] G. Ben-Avi and N. Francez, "Categorial grammar with ontology-refined types," in *Proceedings of CG04*, 2004.

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on July 30,2010 at 09:55:46 UTC from IEEE Xplore. Restrictions apply.