Final project report

Juraj Dzifcak

December 10, 2008

1 Introduction and Motivation

Recently a new way to define stable models semantics have been proposed by [1], which does not use grounding and applies directly to the syntax of first order formulas. This semantics is different from classical stable models semantics [2] and is more powerful. In another work [3], we investigated and presented a way to use stable model(or answer set) semantics and traditional logic programming to represent the semantics of natural language sentences, and in particular, how to properly and efficiently handle defaults and exceptions. However, this approach had several limitations, most importantly it could not deal with existential quantification. Hence by expanding our work and using this new semantics, we should be able to capture more precisely the semantics of natural language.

2 Motivating examples

The main goal of the project is to use the stable models semantics [1] as a representation language for natural language semantics, and in particular focus on cases where the traditional stable model approach struggled to achieve a satisfactory result. In particular, one of the main problems with traditional stable model semantics are existential quantifiers.

For example for a sentence There is an electricity powered car. the approach based traditional stable model semantics had to rely on introducing new constants and could not represent theThere is part of the sentence properly.

The first part of the project involved a creation and exploration of sentences for which the new ASP formalism might be useful. Let us present some of these:

- There is an electricity-powered car.
- John takes a plane.
- John visited three cities.
- A traveler often flies.
- There is non-dividing cell.

- Neurons do not divide.
- Cells that are ill might reproduce.
- Normally every cell divides.
- CellA is a cell.
- Every cell is normally healthy.
- Most cells are healthy.
- Leukocytes do not divide.

First of all, careful reader will not that not all sentences require existential quantification. This is intended, as we want to also show some reasoning with the representations(especially with the part about cells), and hence want a variety in the representation formulas. Let us analyze some of this sentences. The first sentence is a statement about the existence of an object, in this case a car powered by electricity. The second statement says that there is a object, which is a planer and John took it to fly somewhere. These first sentences are to illustrate our approach and show how the new formalism is useful.

As an example, let us look at the representation of the sentence 'Cells that are ill might reproduce.'. The representation of this sentence should be something like:

$\exists x reproduce(c, x) : -cell(c), divide(x), \exists x ill(c, x).$

In other words, if there is some illness of a cell 'c', it might perform some kind of division, or reproduce itself using some form of division. Please note that cells can perform several types of division, for example mitosis or meiosis. Please note that in this case we do represent intransitive verbs as predicate with two arguments, since from the representation perspective, we are interested in more information, such as what kind of division is it, etc. Let us now discuss this representation in more details. First of all, this representation is not 'safe' [2]. Hence we cannot directly translate it into first order logic. The reason is that the second existential quantifier gives a different 'x', making this rule unsafe.

The next set of sentences is a short basic information about some cells in human organism. We will use this discourse to show how we can use our approach for reasoning. Also, this is not a complete list our grammar can generate, the purpose of these is to provide the basics for illustrating our approach and representations.

In addition, please note that one can easily extend the above list and the grammar as needed.

As discussed before, the approach is to use a CCG grammar and λ-calculus to obtain the final representations. Hence we need to create a CCG grammar and corresponding categories which should cover the above sentences.

Let us now present some basic background for our work.

3 Preliminaries

The following is based on [3].

3.1 λ-calculus

λ-calculus was invented by Church to investigate functions, function applications and recursion [4]. Assume a fixed and infinite set of identifiers $\mathcal{V}$.

A λ-calculus expression (or λ-expression for short), is either a *variable* v in $\sqsubseteq$; or an *abstraction* $(\lambda v.e)$ where v is a variable and e is a λ-expression; or an *application* $e_1 e_2$ where e_1 and e_2 are two λ-expressions. In addition, if e is a λ-expression, then if (e) is a λ-expression. We will also refer to λ-expressions as λ-calculus formulas. For example,

$$\lambda x.plane(x) \qquad (\lambda x.x) \qquad \lambda u \qquad y$$

are λ-expressions. Please note that predicates and functions can be easily expressed by λ-expressions. For brevity, we will often use the conventional representation of predicates and functions instead of their λ-expression.

Variables in a λ-expression can be bound or free. In the above expressions, only y is free. Others are bound. Several operations can be done on λ-expressions. A α-conversion allows bounded variables to changed their name.

Given a λ-expression e and variables x_1 and x_2, $\alpha(e, x_1, x_2) = e[x1 := x2]$, where $e[x1 := x2]$ denotes the substitution of x_1 by x_2 in e.

The substitution, or β-reduction, is the only axiom of λ-calculus and in general replaces a free variable with a λ-expression. Given a free variable x and λ-expressions e_1 and e_2, $(\lambda x.e_1)@e_2 = e1[x := e_2]$.

A β-reduction could be viewed as a function application, and will be denoted by the symbol @. For example,

$$\lambda x.plane(x) \ @ \ boeing767$$

results in

$$plane(boeing767)$$

λ-calculus has been used as a way to formally and systematically translate English sentences to first order logic formulas [5].

3.1.1 Examples

We illustrate the use of λ-calculus in natural language by repeating from [6] how the sentence "*John takes a plane*" can be systematically translated to the the logical representation

$$\exists y.[plane(y) \wedge takes(john, y)]$$

The λ-expression for each constituent of the sentence are as follows:

- "John": $\lambda x.(x@john)$.

- "a": $\lambda w.\lambda z.\exists y.(w@y \wedge z@y)$
- "plane": $\lambda x.plane(x)$
- "takes": $\lambda w.\lambda u.(w@\lambda x.takes(u,x))$

We can combine the above λ-expressions to create the formula for the sentence.

- "a plane":
  $\lambda w.\lambda z.\exists y.(w@y \wedge z@y)@\lambda x.plane(x) =$
  $\lambda z.\exists y.(\lambda x.plane(x)@y \wedge z@y) =$
  $\lambda z.\exists y.(plane(y) \wedge z@y)$
- "takes a plane":
  $\lambda w.\lambda u.(w@\lambda x.takes(u,x))@$
  $\lambda z.\exists y.(plane(y) \wedge z@y) =$
  $\lambda u.(\lambda z.\exists y.(plane(y) \wedge z@y)@\lambda x.takes(u,x))$
  $\lambda u.(\exists y.(plane(y) \wedge \lambda x.takes(u,x)@y))$
  $\lambda u.(\exists y.(plane(y) \wedge takes(u,y)))$
- "John takes a plane":
  $\lambda x.(x@john)@\lambda u.[\exists y.(plane(y) \wedge takes(u,y))]$
  $\lambda u.(\exists y.[plane(y) \wedge takes(u,y)])@john$
  $\exists y.[plane(y) \wedge takes(john,y)]$

3.2 Combinatorial Categorial Grammar

Although various kinds of grammars have been proposed and used in defining syntax of natural language, combinatorial categorial grammars (CCGs) are considered the most appropriate from the semantic point of view. In building the λ-expressions above, CCG parser output would be able to correctly dictate which λ-expression should be applied to which.

A *combinatorial categorical grammar* (CCG) can be characterized by

- a set of *basic categories*,
- a set of *derived categories*, each constructed from the basic categories, and
- some syntactical(combinatorial) rules describing the concatenation and determining the category of the result of the concatenation.

Moreover, every lexical element is assigned to a category. There are various combinatorial rules used in CCGs for natural language, such as (forward/backward/forward-crossing/backward-crossing) function application, (forward/backward/forward-crossing/backward-crossing) substitution and others (See e.g [7]).

The following is an example of a very simple categorical grammar, called CCG_1:

- CCG_1 has two basic categories: N and S.

- The derived categories of CCG_1 are:
  - A basic category is a derived category;
  - If A and B are categories then the expressions $(A\backslash B)$ and (A/B) are categories;

  Thus, $(N\backslash S)$, $(S\backslash N)$, $(N\backslash(N\backslash S))$, (N/S), $(N/S)\backslash N$ are derived categories of CCG_1.
- The syntactic rule for CCG_1:
  - If α is an expression of category B and β is an expression of category $(A\backslash B)$ then the concatenation $\alpha\beta$ is of category A.
  - If α is an expression of category B and β is an expression of category (A/B) then the concatenation $\beta\alpha$ is of category A.
- CCG_1 contains the following objects: *Tim* whose category is NP and *swims* whose category is $(S\backslash NP)$.

Intuitively, the category of 'swims' is $S\backslash NP$ means that if an NP (a noun phrase) is concatenated to the left of 'swims' then we obtain a string of category S, i.e., a sentence. Indeed, 'Tim' being an NP, when we concatenate it to the left of 'swims' we obtain 'Tim swims', which is a sentence.

3.3 Programs with explicit quantifiers

The formalism of logic programs with quantifiers is based on [1]. The syntax is similar to the 'classical' ASP programs(see e.g. [3]), except it allows explicit quantifiers. Formally, an extended rule is of the form

$$\mathbf{F} \leftarrow \mathbf{G}$$

where G and H are formulas with no function constants of positive arity such that every occurrence of an implication in G and H is in a negative formula. An extended program is a set of extended rules. Intuitively, one can assume F and G to be a first order logic formula without 'positive' implications. Also, if G is empty, we might omit the $\leftarrow$ symbol.

The semantics of an extended program P is defined as $SM[F]$, where F is a conjunction of the universal closure of implications corresponding to the rules.

4 The grammar

Let us now discuss the first part of the grammar. For the purpose of this presentation, we will only use two syntactic rules, forward and backward application. Hence the syntactic rules for determining the category of $\alpha\beta$ are as follows:

- (**FA**:) if α is of category B and β is of category A/B then $\beta\alpha$ is of category A, and

- (**BA**:) if α is of category B and β is of category $A\backslash B$ then $\alpha\beta$ is of category A.

The actual syntactical categories and their corresponding CCG categories can be found in the table. The second part, generation of corresponding λ-expression is a part of the next work.

The CCG for our language consists of the several basic categories:

- N – stands for 'noun'
- NP – stands for 'noun phrase'
- S – stands for 'sentence'
- PP – stands for 'propositional phrase'
- $NP[obj]$ – stands for 'noun phrase', representing an object

We will use of specific CCG for this example. Observe also that a word can be assigned to multiple categories. In addition, in general the grammar is not limited to these particular rules, and additional rules may be added as needed.

Let us now present a snippet of the grammar that captures most of our sentences. It is given in the following table (we group words of similar categories together and separate them using ',' to make the table more readable):

With the syntactic part of the grammar, we can parse the sentences, however we cannot obtain their semantics yet. For this, we need the semantic representations of words in the form of λ-calculus formulas, extended to programs with explicit quantifiers. So let us now present some parts of this grammar:

With the above structures, we are now able to parse and represent the semantic of our sentences.

5 Examples

Let us now present the sentences and their corresponding translations. Recall the list of presented sentences:

- There is an electricity powered car.
- John takes a plane.
- John visited three cities.
- A traveler often flies.
- There is non-dividing cell.
- Neurons do not divide.
- Cells that are ill might reproduce.

- Normally every cell divides.
- CellA is a cell.
- Every cell is normally healthy.
- Most cells are healthy.
- Leukocytes do not divide.

The representation of these are as follows:

- There is an electricity powered car. $\exists x car(x) \wedge electricity\_powered(x)$
- John takes a plane. $\exists y.[plane(y) \wedge takes(john, y)]$
- John visited three cities. $\exists x.[city(x) \wedge visited(john, x)]$
- A traveler often flies. $fly(x) : -traveler(x)$.
- There is a non-dividing cell. $\exists x cell(x) \wedge non\_dividing(x)$
- Neurons do not divide. $-divide(x) : -neuron(x)$
- Cells that are ill might reproduce. $\exists x reproduce(c, x) : -cell(c), divide(x), \exists x ill(c, x)$.
- Normally every cell divides. $divide(x) : -cell(x), not - divide(x)$
- CellA is a cell. $cell(cellA)$
- Every cell is normally healthy. $healthy(x) : -cell(x), not - healthy(x)$
- Most cells are healthy. $healthy(x) : -cell(x), \neg\exists y ill(x, y)$
- Leukocytes do not divide. $-divide(x) : -leukocyte(x)$.

Let us now show in detail how to obtain the representation of the first sentence. First of all, the syntactic and semantic categories of the words are:

- 'There' : NP, $\lambda x.x$
- 'is' : $(S/NP)\backslash NP$, $\lambda x.x$
- 'an' : NP/N, $\lambda v.\exists x v@x$
- 'electricity-powered' : $N/N[obj]$, $\lambda y \lambda x.electricity\_powered(x) \wedge y@x$
- 'car' : $N[obj]$, $\lambda x.car(x)$

When parsing this sentence, we combine 'car' with 'electricity-powered', obtaining the category N and the semantics

$(\lambda y \lambda x.electricity\_powered(x) \wedge y@x)@\lambda x.car(x) =$

$\lambda x.electricity\_powered(x) \wedge \lambda x.car(x)@x =$

$\lambda x.electricity\_powered(x) \wedge car(x)$

Then we combine 'an' with 'electricity-powered car', resulting in category NP and semantics

$(\lambda v.\exists x v@x)@\lambda x.electricity\_powered(x) \wedge car(x) =$

$\exists x(\lambda x.electricity\_powered(x) \wedge car(x))@x =$

$\exists x electricity\_powered(x) \wedge car(x).$

Then, 'is' is combined with 'there', obtaining S/NP and the semantics $\lambda x.x$. Finally, we combine 'There is' with 'an electricity-powered car' to obtain S with the semantics

$\lambda x.x@\exists x electricity\_powered(x) \wedge car(x) =$

$\exists x electricity\_powered(x) \wedge car(x)$

Hence we obtain our desired representation. The rest of the sentences can be parsed in a similar way.

6 Implementation

Let us now discuss the implementation aspects of the system. It is capable of parsing a sentence and based on the semantical part of lexical entries(in λ-calculus) and construct the resulting representations of a sentence.

The lexicon provides all the categories and semantics expressions for a word. For example, an entry in the lexicon for the word 'most' can look like follows:

$most : (S/(S|NP))/NP :$

$\#u\#v.(v@X \leftarrow u@X, not \neg v@X)$

The # symbol represents the λ symbol in the λ-calculus expressions. The system currently support several combinatorial rules. Most notable forward and backward application and their generalized version, substitution rules and type raising. By default, the system only uses the application rules unless specified otherwise. The input is composed of the lexicon and the discourse we want to compute the semantic representation for and optionally if we want to use additional combinatorial rules. The sentences are then parsed and their semantical representations are computed.

The lexicon has several special symbols and shortcuts. Let us look at another entry, for a verb 'fly'.

$fly : S|NP, S|N : \#x.fly(x), \#x.fly(X) <--x@X$

Note that the symbol ',' is used as a delimiter to separate multiple syntactic or semantic entries, while ':' is used as a separator between the word, it's syntactic categories and it's semantic categories. These are special symbols that cannot be used in the actual expression, unless the parser is changed. All other symbols can be used freely, although care should be taken when using

JAVA string special symbols (such as the ones for new line '
n'etc.).

This system includes a relatively simple parser with several possible heuristics to speed it up if needed. Note that for most cases, these should not be necessary.

The first heuristics tries to assign categories in a such a way, that for any complex category, the basic categories(or less complex ones) are present in the sentence. This ensures we only try the categories that have a chance to form a category S, sentence. In addition, we assume that the categories in the lexicon are order according to the probability with which they appear in the words and thus the first ones are preferred over the latter ones. Please note that even if this is not the case, this does not yield any decrease in performance. In addition, a parameter can be specified to randomize the order of categories. The second heuristics is used for rule selection. The idea is that we try to combine the less complex categories first, if possible. This is because these combinations seem to more likely lead to a failure if there is one. Also, a randomization parameter in the input can be used to make the order of combinations random.

This system can easily parse the sentences listed in the example sections and produce their semantic representation. However, depending on the structure of the sentences, one might, or might not be able to evaluate them further.

7 Concluding remarks

In this project we presented an approach to represent the semantics of natural logic using a strong non monotonic formalisms. We succeeded in providing a strong framework to represent the semantics of natural language, allowing for both non-monotonicity and existential quantification, while still remaining within reasonable computational boundaries. The next step is to enhance and polish this work further, add more complex grammar and sentences to further support the need for such representation.

8 References

[1] P. Ferraris, J. Lee and V. Lifschitz, A new perspective on stable models, in Proceedings of IJCAI-07, 2007. [2] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming, in Logic Programming: Proceedings of the Fifth International Conference and Symposium, 1988, pp. 1070-1080. [3] C. Baral, J. Dzifcak and Tran Cao Son, Using Answer Set Programming and Lambda Calculus to Characterize Natural Language Sentences with Normatives and Exceptions, in Proceedings of AAAI-08, 2008. [4] Church, A, An unsolvable problem of elementary number theory, *Am. Jou. of Mathematics* 58:345–363, 1936. [5] Blackburn, P., and Bos, J.Representation and Inference for Natural Language: A First Course in Computational Semantics, 2005. [6] Balduccini,

M., Baral, C., and Lierler, Y., Knowledge representation and Question Answering, In Handbook of Knowledge Representation, 2008. [7] Steedman, M., The syntactic process, MIT Press, 2001

Word	Categories	Reference
Non-transitive verb v	$S\backslash NP$,	S1
	$S\backslash N$	S2
	$S\backslash NP[obj]$,	S3
Examples :		
fly	$S\backslash NP$,	
	$S\backslash N$,	
flies	$S\backslash NP[obj]$	
swim	$S\backslash NP$,	
	$S\backslash N$,	
swims	$S\backslash NP[obj]$	
Transitive verb v	$S\backslash NP/NP$,	T1
Examples :		
like	$S\backslash NP/NP$,	
Adverb adv	$(S/(S\backslash NP))/NP$	M
Examples :		
most	$(S/(S\backslash NP))/NP$	
Auxiliary verb axv	$(S/(S\backslash NP))\backslash NP$	D1
Examples :		
do	$(S/(S\backslash NP))\backslash NP$	
is	$(S/NP)\backslash NP$,	I1
	$(S/N)\backslash NP[obj]$	I2
are	$(S/NP)\backslash NP$	A1
are not	$(S/NP)\backslash NP$	A2
Noun (plural) n	N, NP	
Examples :		
cells	N, NP	B1, B2
neurons	N, NP	P1, P2
proteins	N, NP	BA1, BA2
Noun (singular) n	N, $NP[obj]$	
Examples :		
neuron	N, $NP[obj]$	AP1, AP2
CellA, *CellB*	N, $NP[obj]$	T
Adjective adj	NP/N	F1
	$NP/N[obj]$	F2
Examples :		
fictional	NP/N	
	$NP/N[obj]$	
Conjunction c	$(S\backslash S)/S$	C1
	$(NP\backslash NP)/NP$	C2
Examples :		
and	$(S\backslash S)/S$	
	$(NP\backslash NP)/NP$	
Pronoun p	$(NP\backslash N)/(S\backslash NP)$	P1
Examples :		
that	$(NP\backslash N)/(S\backslash NP)$	
a	$(NP/N[obj])$	AR1
	$NP/(NP/N[obj])$	AR2
the	(NP/N)	AR3
	$NP/(NP/N)$	AR4

Table 1: A section of the used CCG grammar

Word	Cat.	λ-ASP-expression
$non-transverb\ v$	S1	$\lambda x.v(x)$
	S2	$\lambda x.v(X) \leftarrow x@X$
	S3	$\lambda x.v(x)$
	S4	$\lambda x.\lambda y.v(x,y)$
$divide$	–	$\lambda x.divide(x)$
	–	$\lambda x.divide(x)$
	–	$\lambda x.divide(X) \leftarrow x@X$
$swim$	–	$\lambda x.swim(x)$
	–	$\lambda x.swim(x)$
	–	$\lambda x.swim(X) \leftarrow x@X$
$most$	M	$\lambda u\lambda v.$ $(v@X \leftarrow u@X,\ not\ \neg v@X)$
do	D1	$\lambda u\lambda v.v@X \leftarrow u@X$
$do\ not$	D1	$\lambda u\lambda v.\neg v@X \leftarrow u@X$
is	I1,I2	$\lambda v.\lambda u.u@v$
are	A1	$\lambda u\lambda v.v@X \leftarrow u@X$
$are\ not$	A2	$\lambda u\lambda v.\neg v@X \leftarrow u@X$
$noun\ n(sing)$	–	$\lambda x@n$
$noun\ n(plu)$	–	$\lambda x.n(x)$
$cells$	–	$\lambda x.cell(x)$
$neurons$	–	$\lambda x.neuron(x)$
$proteins$	–	$\lambda x.bat(x)$
$CellB/CellA$	–	$\lambda x.x@CellB/CellA$
$adjective\ adj$	F1,F2	$\lambda v\lambda u.adj(u) \wedge v@u$
$fictional$	–	$\lambda v\lambda u.fictional(u) \wedge v@u$
and	C1,C2	$\lambda v\lambda u.u \mid v$
or	C1,C2	$\lambda v\lambda u.u\#v$
$that$	P1	$\lambda v\lambda u.u \wedge v$
a	AR1, AR2	$\lambda v.\exists x v@x$
the	AR3, AR4	$\lambda x.x$
$three$	AR3, AR4	$\lambda v.\exists x v@x$

Table 2: λ-ASP-expressions