ISSN 2007-9737

Automatic Theorem Proving for Natural Logic: A Case Study on Textual Entailment

Jesús Lavalle[1,2], Manuel Montes[1], Héctor Jiménez[3], Luis Villaseñor[1], Beatriz Beltrán[2]

[1] Insituto Nacional de Astrofísica, Óptica y Electrónica, Coordinación de Ciencias Computacionales, Santa María Tonanzintla, Mexico

[2] Benemérita Universidad Autónoma de Puebla, Facultad de Ciencias de la Computación, Puebla, Mexico

[3] Universidad Autónoma Metropolitana, Unidad Cuajimalpa, División de Ciencias de la Comunicación y Diseño, Departamento de Tecnologías de la Información, Ciudad de México, Mexico

jlavalle@ccc.inaoep.mx, mmontesg@ccc.inaoep.mx, hjimenez@correo.cua.uam.mx, villasen@ccc.inaoep.mx, bbeltran@cs.buap.mx

Abstract. Recognizing Textual Entailment (RTE) is a Natural Language Processing task. It is very important in tasks as Semantic Search and Text Summarization. There are many approaches to RTE, for example, methods based on machine learning, linear programming, probabilistic calculus, optimization, and logic. Unfortunately, no one of them can explain why the entailment is carried on. We can make reasonings, with Natural Logic, from the syntactic part of a natural language expression, and very little semantic information. This paper presents an Automatic Theorem Prover for Natural Logic that allows to know precisely the relationships needed in order to reach the entailment in a class of natural language expressions.

Keywords. Textual entailment, automatic theorem proving, natural logic.

1 Introduction

The main objective of Automatic Theorem Proving is that given an expression of some logical system, a computer program can decide if that expression follows from a set of axioms and inference rules.

There are many procedures to reach this goal, for example, Resolution, Semantic Tableaux, Hilber Systems, Natural Deduction, Davis-Putnam, and Sequent Calculus.

This work is in the line of Sequent Calculus, in the sense that after applying an inference rule the size of the original expression decreases, which is called subformula property.

Intuitively, it means that the truth of an expression depends only on its constituent elements. Of course, the type of axioms and inference rules change from one system to another.

For example, in *AB* grammars, the words of an expression in English can be considered as axioms, if after applying modus ponens to them we get an expression of type t, it means that the expression in English is a sentence

A collateral objective in Automatic Theorem Proving is to say why an expression does not follow from the axioms. This is called, the explanatory power of the Automatic Theorem Prover. We are going to use this kind of tools to develop an automatic theorem prover for Natural Logic with emphasis on textual entailment.

Recognizing Textual Entailment is essential for other Natural Language Processing tasks such as: Semantic Search, Question Answering, Text Summarization, and Information Extraction. Different methods have been used to solve the RTE problem [7], those methods are based on machine learning, linear programming, probabilistic calculus, optimization, and logic.

The logical methods used in RTE, although they use an inference mechanism, decide on the entailment through machine learning algorithms or some kind of optimization. In such a situation, it is not possible to know what relationships, among the subexpressions of the text and the hypothesis, are avoiding the entailment.

Natural Logic was developed to reason in natural language without having to use some kind of logical form [22, 18]. Natural Logic only uses lexical, syntactic, and basic semantic information of a language. Natural Logic can be viewed as the joint of some kind of Categorial Grammar, with modus ponens as the unique inference rule, and reasoning with polarity.

In this paper, we are going to present an Automatic Theorem Prover for Natural Logic. Its main features are: it can make entailments on more than one subexpression, and it finds precisely the subexpressions that do not permit the entailment.

We explain briefly in section 2 four approaches to RTE that use some kind of inference mechanism, and one that is based on Natural Logic. We deal with Natural Logic in section 3, section 4 is devoted to construct the algorithms needed for the proof theory of an extension of *AB* grammars. Section 5 contains an adaptation of the algorithm of van Benthem to compute polarity in *AB* grammars, and an Automatic Theorem Prover for Natural Logic is developed. Later, section 6 shows some examples of the Automatic Theorem Prover. Finally, section 7 gives our conclusions, and future work directions.

2 Approaches based on an Inference Mechanism

The methods discussed in this section use some kind of inference mechanism to recognize textual entailment, excepting for the one of MacCartney and Manning, which is included because it is based on Natural Logic.

2.1 COGEX

The system of Hodges et al. [9] transforms the input text and hypothesis into logical forms. The transformation process includes part-of-speech tagging, parse tree generation, word sense disambiguation and semantic relations detection.

In order to use the logic prover COGEX, a list of clauses called ”set of support” is required, this is used to begin the search for inferences. Another list, called the usable list, contains clauses used by COGEX to produce inferences. The axioms are about knowledge of the world, linguistic rewriting rules, and synsets of WordNet.

The clauses in the set of support are weighted, a clause with a lesser weight is prefered to participate in the search. The negated hypothesis (COGEX proves by refutation) is added to the set of support with the largest weight, this guarantees that the hypothesis will be the last clause used in the search.

If a refutation is found the prover ends, if there is not a refutation the predicate arguments are relaxed. If despite arguments relaxation a refutation is not found, predicates are dropped from the negated hypothesis until a refutation is found.

When a refutation is found, a score for it is computed, beginning with a perfect score and subtracting points for axioms used, arguments relaxed, and predicates dropped.

If the score for a refutatiion is greater than a threshold, then it is considered that the entailment is true, otherwise it is considered false.

2.2 OTTER

In the proposal of Akhmatova [2] the meaning of a sentence is represented by the set of atomic propositions contained in it, then the sentences are compared by means of their associated propositions.

A syntax-driven semantic analysis is used to get the atomic propositions associated with a sentence. The output of the parser is used as input for the semantic analyser; from the output of the analyser, the representation of the sentence in first order logic, which is called *the logic formula*, can be derived.

For Akhmatova, there are many ways to describe meaning through logical form, but they are rigid and hard to produce. Because of that, a simplified representation is proposed.

The simplified representation is build from: three types of objects $Subj(x)$, $Obj(x)$ and $Pred(x)$, a meaning attaching element $iq(x, < meaning\ of\ x >)$, and two variants of relationships $attr(x, y)$ and $prep(x, y)$.

Later, usign WordNet, a relatedness score between words is computed from the paths between the senses of the words, the longer the path, the lesser is the relatedness. This score together with knowledge rules are given to the automatic theorem prover OTTER.

If for every proposition in the hypothesis sentence p_{h_i} there is one proposition in the text sentence p_{t_j}, such that $p_{t_j} \rightarrow p_{h_i}$, then the entailment holds, otherwise the entailment does not hold.

2.3 Abduction

Raina et al. [17] begin constructing a syntactic dependency graph using a parser, hand written rules are used to find the heads of all nodes in the parse tree. The relations represented in the dependency graph are translated into a logical formula representation. Each node in the graph is converted into a logical term and it is assigned a unique constant.

Later, abductive theorem proving is realized by the resolution method, where each abductive assumption, and its degree of plausabilty is quantified as a nonnegative cost using the *assumption cost model*. The objective is to find the proof of minimun cost, which is chosen automatically by a machine learning algorithm.

2.4 Vampire and Paradox

The approach of Bos and Markert [4, 5] is based on what they call *shallow semantic analysis* and *deep semantic analysis*.

Four features are obtained from the shallow semantic analysis, the overlap between words in text and hypothesis, the length of text, the length of hypothesis, and the relative length of hypothesis with respect to the text.

To achieve the deep semantic analysis, they use a robust wide-coverage parser, which produces proof trees of Combinatory Categorial Grammar [19]. Afterwards, the proof trees are used to build discourse representation structures, these are the semantic representations from Discourse Representation Theory. Later, the semantic representations are translated into first order logic expressions.

The model checker Paradox and the automatic theorem prover Vampire are used to prove wheter or not the text implies the hypothesis. Bos and Markert take two features from the automatic theorem prover, and six from the model checker.

A decision tree is trained with the twelve features, and it is used to decide if the text implies the hypothesis.

2.5 Natural Logic

MacCartney and Manning [13, 12] use Natural Logic to avoid logical forms, their system is called *NatLog*. They begin with a linguistic pre-processing, the text and the hypothesis are parsed with the Stanford parser, the main purpose of this step is monotonicity marking; nevertheless, they do not use polarity (see section 3) as an inference mechanism.

The second step consists of an alignment between the text and the hypothesis, alignments are represented by sequences of atomic edits over words.

Finally, taking as features the monotonicity infromation and the sequences of edits, a decision tree is trained.

2.6 Brief Analysis of the Methods

As it can be seen in Table 1, the methods based on some inference mechanism use first-order logic (FOL) as a form to represent the text and the hypothesis.

For us, it is unclear the decision mechanism that follows the system of Akhmatova, a relatedness score is computed, but its role in the decision process is never mentioned.

The other methods use a decision process different to the inference mechanism, as it has been explained, hiding why the entailment was not carried out.

Table 1. Summary of the main characteristics of some logical approaches

First Author	Inference Mechanism	Logic	BK	Challenge	Decided by
Hodges	COGEX	FOL	WordNet	RTE-2	Optimization
Akhmatova	OTTER	FOL	WordNet	RTE-1	Unclear
Raina	Abduction	FOL		RTE-1	Machine Learning
Bos	Vampire y Paradox	FOL	WordNet	RTE-1	Machine Learning
MacCartney			WordNet	RTE-3	Machine Learning

3 Natural Logic

Sánchez in his Ph. D. dissertation [18] formalizes the ideas of van Benthem about Natural Logic and monotonic reasoning [20, 21]. Even though the origins of Natural Logic go back to Aristotle [22, 11]; the central idea, in the program of Natural Logic of van Benthem et al., is that natural language, besides communicating ideas, serves to reason without having to use formal systems, as predicate calculus or high order logics. The idea is to use the syntactic structure of a sentence, semantic properties of their lexical constituents, and a functor constructor.

According to Icard and Moss [10], van Benthem [20] and Sánchez [18] define proof systems to reason about entailment using monotonicity in high order languages.

Both van Benthem and Sánchez use, for the syntactic analysis of a sentence, a version of categorial grammars called *calculus of Ajdukiewicz* [1, 3, 14]. This is based on basic types e (for entities), and t (for truth values), more complex types of the categorial language are constructed recursively by the creation of functors, formally:

Definition 3.1. *The calculus of Ajdukiewicz.* The categorial language of the calculus of Ajdukiewicz $\mathcal{LC}$ is given by:

1. e and t belong to $\mathcal{LC}$,
2. If α and β belong to $\mathcal{LC}$, then (α, β) also belong to $\mathcal{LC}$.

The unique inference rule in the calculus of Ajdukiewicz takes the form:

$$\frac{(\alpha,\beta) \qquad \alpha}{\beta} \tag{1}$$

and it does not matter if the type α appears either on the left, or on the right of the functor (α, β).

It is assumed that each word in the lexicon has a type, for example: common nouns have type (e, t), transitive verbs have type $(e, (e, t))$, intransitive verbs have type (e, t), adjectives and adverbs have type $((e, t), (e, t))$, noun phrases have type $((e, t), t)$, and determiners have type $((e, t), ((e, t), t))$.

Hence to know whether a sentence is well formed, the inference rule (1) is used to build a proof tree, if its root is t, then the sentence is well formed, otherwise the sentence is ill-formed.

Nevertheless, as there are words that play different roles (for example, *white* could either be an adjective, or a noun, or a verb), if a sentence contains words of this kind, the algorithm that constructs the proof tree for such a sentence would have to try with the different types of each word until the type t has been derived.

We have a proof tree in Figure 1 for the sentence *Dobie didn't bring every ball*. In this figure, it is worth noting that: the type of *every* has its first argument on the right; the type of *every ball* has its argument on the left; the type of *Dobie* has its argument on the right, and the type t has been derived from both of them, indicating that the expression in natural language is a well formed sentence.

The first semantic element of Natural Logic is that each type denotes a set, hence D_e denotes the set of entities, D_t denotes the set $\{0, 1\}$, $D_{(\alpha,\beta)}$ denotes the set whose elements are functions from α to β. Also, the following partial order relations are defined on each type.

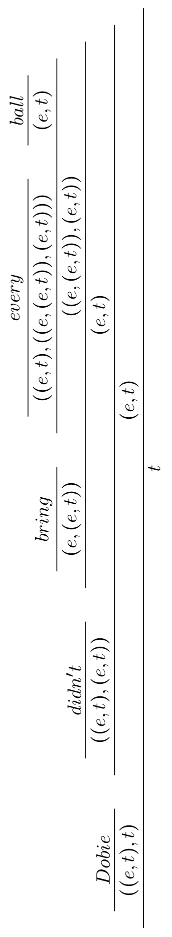


Fig. 1. Proof tree for the sentence *Dobie didn't bring every ball*

Definition 3.2. *Partial order relations.* Partial order relations on the denotations of types can be defined in the following way:

1. If $d, d' \in D_e$, then $d \leq_e d'$ if and only if $d = d'$,
2. If $d, d' \in D_t$, then $d \leq_t d'$ if and only if $d = 0$ or $d' = 1$,
3. If $d, d' \in D_{(\alpha,\beta)}$, then $d \leq_{(\alpha,\beta)} d'$ if and only if for all $x \in D_\alpha, d(x) \leq_\beta d'(x)$.

The second semantic element of Natural Logic is that working with a proof system based on functors, and taking into account a partial order relation on each type, it is possible to define static characteristics on functors, namely a functor can be either upward monotone, or downward monotone; obviously a functor can also be non-monotone, according to the following terms [8]:

Definition 3.3. *Monotonicity.* A function $d \in D_{(\alpha,\beta)}$ is:

1. *upward monotone* $(+d)$ if and only if:

   for all $x, y \in D_\alpha, x \leq_\alpha y$ implies that
   $$d(x) \leq_\beta d(y),$$

2. *downward monotone* $(-d)$ if and only if:

   for all $x, y \in D_\alpha, x \leq_\alpha y$ implies that
   $$d(y) \leq_\beta d(x).$$

3. *non-monotone* $(\cdot d)$ if and only if it is neither upward monotone, nor downward monotone.

As it has been stated, the inference rule (1) must be applied to construct a proof tree, therefore a functor node and an argument node are required. The resulting node will serve as either the functor node, or the argument node to construct the following level of the proof tree. In this way, the construction of a proof tree is done by composing functors: if an upward (downward) monotone functor α is argument of a functor β, then the static characteristic of α can change, depending of the static characteristic of β. It is seen in Table 2 [10] the result of composing upward monotone (+), downward monotone (−), and non-monotone (·) functors.

Table 2. Result of the composition of upward monotone (+), downward monotone (−), and non-monotone (·) functors

$\circ$	$+$	$-$	$\cdot$
$+$	$+$	$-$	$\cdot$
$-$	$-$	$+$	$\cdot$
$\cdot$	$\cdot$	$\cdot$	$\cdot$

From Table 2 we can infer that the composition of m upward and downward monotone functors will be upward monotone if the number of downward monotone functors is even, otherwise the composition will be downward monotone, and if one of the functors in composition is non-monotone, then the whole composition will be non-monotone. The composition of functors is called *polarity*, it is said that polarity is: positive $(+)$ when the composition is upward monotone, negative $(-)$ when the composition is downward monotone, and neutral $(\cdot)$ when the composition is non-monotone. Hence, polarity is a dynamic characteristic of some functors, which is given by the position of the functors in the composition.

In terms of the proof tree of a sentence, a functor node that is upward monotone will have positive polarity if it is the argument of a composition where an even number of downward monotone functors are involved, otherwise it will have negative polarity.

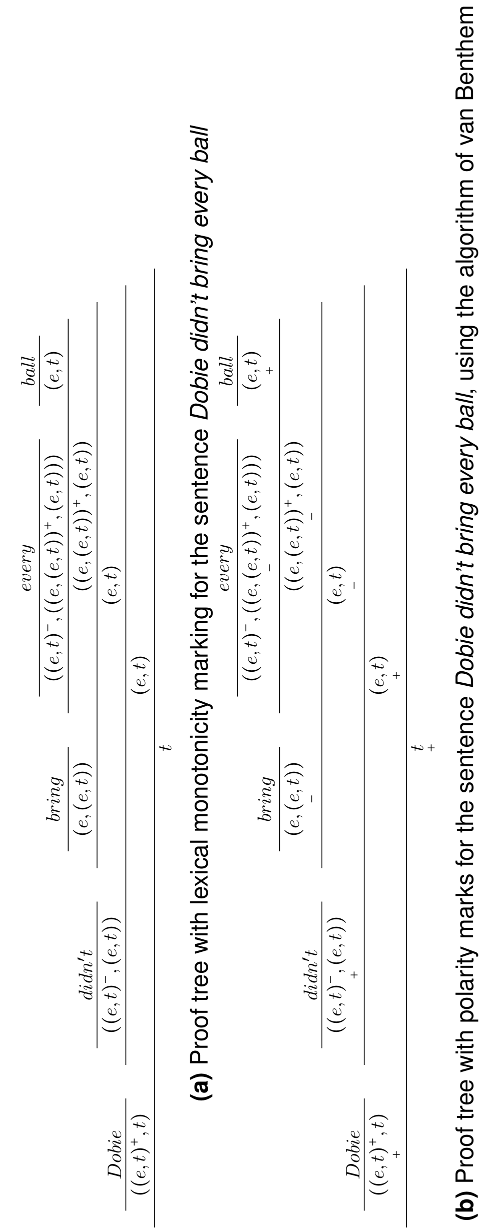
On the same terms, a functor node that is downward monotone will have positive polarity if it is the argument of a composition where an odd number of downward monotone functors are involved, else it will have negative polarity.

Once the polarity of a node i is known in the proof tree of the sentence S, the subtree whose root is node i can be replaced for a greater one (in the sense of Definition 3.2) if node i has positive polarity, giving as a result sentence S'; in a dual way, the subtree whose root is node i can be replaced for a lesser one (in the sense of Definition 3.2) if node i has negative polarity, giving as a result sentence S'. In both cases, it is said that S implies S'. Sánchez [18] proved the soundness of the above implication with respect to the Semantics associated with the proof system. This is the way of reasoning in Natural Logic.

In order to compute the polarity of the elements of a sentence, the following clause is added to definition 3.1.

If (α, β) is in $\mathcal{LL}$, then (α^+, β), and (α^-, β) are also in $\mathcal{LL}$.

This clause does not belong to the calculus of Ajdukiewicz, it has been included to mark either the upward monotonicity of a functor from α to β, (α^+, β), or the downward monotonicity of a functor from α to β, (α^-, β).



(a) Proof tree with lexical monotonicity marking for the sentence *Dobie didn't bring every ball*

(b) Proof tree with polarity marks for the sentence *Dobie didn't bring every ball*, using the algorithm of van Benthem

Fig. 2. The two steps to compute polarity with the algorithm of van Benthem, for the sentence *Dobie didn't bring every ball*

So that, it is supposed that the lexicon contains the information of monotonicity that some functors

ISSN 2007-9737

require, as shown below:

$Dobie : ((e,t)^+,t)$ $\quad didn't : ((e,t)^-,(e,t))$
$bring : (e,(e,t))$ $\quad every : ((e,t)^-,((e,(e,t))^+,(e,t)))$
$ball : (e,t)$

As an example, we have the proof tree in Figure 2a with lexical monotonicity marking for the sentence *Dobie didn't bring every ball*.

Once the proof tree has lexical monotonicity marks, the algorithm of van Benthem begins marking the root of the proof tree with polarity $+$, then if the functor in turn is upward monotone, the polarity mark is propagated. In case that the functor is downward monotone, then the polarity mark of the argument is reversed, because a downward monotone functor reverses the order relation of the elements of its domain. Figure 2b exemplifies Algorithm 3.1 for the sentence *Dobie didn't bring every ball*.

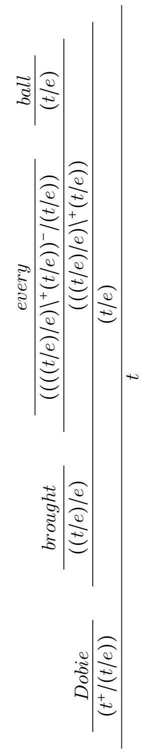
Algorithm 3.1. van Benthem polarity algorithm

1. Label the root with $+$.
2. Propagate notations up the tree.
   (a) If a node of type β is labeled l and its children are of type (α^+,β) and α, then both children are labeled l (diagrammatically, $\frac{\underset{l}{(\alpha^+,\beta)} \quad \underset{l}{\alpha}}{\underset{l}{\beta}}$).
   (b) If a node of type β is labeled l and its children are of type (α^-,β) and α, then the former child is to be labeled l and the latter child is to be labeled $-l$, that is, the flipped version of l (diagrammatically, $\frac{\underset{l}{(\alpha^-,\beta)} \quad \underset{-l}{\alpha}}{\underset{l}{\beta}}$).

To know when a sentence N' is entailed from a sentence N, first we have to define what a subexpression of a natural language expression is.

Definition 3.4. *Subexpression.* Let $N = w_1 w_2 \cdots w_n, n \geq 1$, be a natural language expression, where each word $w_i : \alpha_i$, it is said that M is a *subexpression* of N if and only if one of the following clauses holds:

1. $M = w_i, 1 \leq i \leq n$;
2. $M = w_i M', 1 \leq i \leq n-1$, where M' is a subexpression of N, and ($w_i \in D_{\alpha\to\beta}$ and $M' \in D_\alpha$) or ($w_i \in D_\alpha$ and $M' \in D_{\alpha\to\beta}$) holds;
3. $M = M' w_i, 2 \leq i \leq n$, where M' is a subexpression of N, and ($w_i \in D_{\alpha\to\beta}$ and $M' \in D_\alpha$) or ($w_i \in D_\alpha$ and $M' \in D_{\alpha\to\beta}$) holds;
4. $M = M'M''$, where M' and M'' are subexpressions of N, and ($M' \in D_{\alpha\to\beta}$ and $M'' \in D_\alpha$) or ($M' \in D_\alpha$ and $M'' \in D_{\alpha\to\beta}$) holds. □



$ball$
(t/e)
$every$
$((((t/e)/e)\backslash^+(t/e))^-/(t/e))$
$(((t/e)/e)\backslash^+(t/e))$
(t/e)
$brought$
$((t/e)/e)$
$Dobie$
$(t^+/(t/e))$
t

Fig. 3. Proof tree for the sentence *Dobie brought every ball*

Example 3.1. Let N = *Dobie brought every ball*, according to clause 1 of Definition 3.4, we have that *Dobie, brought, every*, and *ball* are subexpressions of N; looking at Figure 3 we have that *every ball* is also a subexpression by clause 2 of Definition 3.4, because *every:* $((((t/e)/e)\backslash^+(t/e))^-/(t/e))$, and *ball:* (t/e), by the same clause are also subexpressions *brought every ball*, and *Dobie brought every ball*. □

When we say that $N(M)$ is a natural language expression, we also mean that it has M as subexpression.

Definition 3.5. *Entailment on the same subexpression.* Let $N(M)$, and $N(M')$ be two natural language expressions with $M, M' : \alpha, M \neq M'$, we define that $N(M)$ *entails on the same subexpression* $N(M')$ (symbolically $N(M) \vdash N(M')$) in the following way:

$$N(M) \vdash N(M') \text{ if and only if}$$

$$\begin{cases} M \text{ has positive polarity and } M \leq_\alpha M', \text{ or} \\ M \text{ has negative polarity and } M' \leq_\alpha M. \end{cases}$$

□

4 Automatic Theorem Proving for an Extension of *AB* Grammars

We are going to extend a version of categorial grammars called *AB grammars* [14]; in this extension, types are constructed by $L ::= P|(L^s/L)|(L\backslash^s L)$, where P is the set of primitive types (in our case e and t), and functors are constructed using the operators $^s/$ and $\backslash^s$, with $s \in \{+, -, \cdot\}$; these operators are intended to distinguish if the argument of a functor is either on the right or on the left, respectively. Also to mark whether a functor is upward monotone ($+$), downward monotone ($-$), or non-monotone ($\cdot$).

As it has been stated, to prove that a natural language expression is well formed (it is a sentence), a proof tree with root t has to be constructed, using the inference rules for syntactic categories $\dfrac{X^s/Y \quad Y}{X}$, and $\dfrac{Y \quad Y\backslash^s X}{X}$; as an example is the proof tree in Figure 4.

A natural language expression $nlexp$ is represented by the list $[w_1 : X_1, \ldots, w_n : X_n]$, where w_i is an English word in $nlexp$, and X_i is its type, $1 \leq i \leq n$. As an example, the natural language expression *Dobie didn't bring every ball* is represented by the list $[Dobie : (t^+/(t/e)), didn't : ((t/e)^-/(t/e)), bring : ((t/e)/e), every : ((((t/e)/e)\backslash^+(t/e))^-/(t/e)), ball : (t/e)]$.

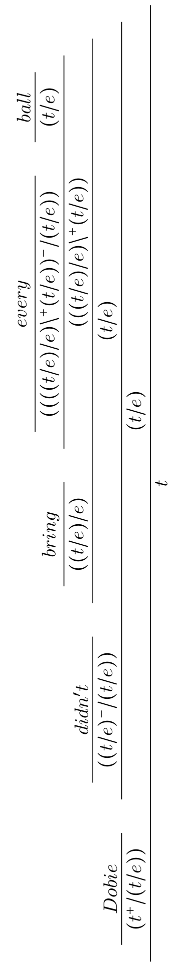


Fig. 4. Proof tree for the sentence *Dobie didn't bring every ball*, in an extension of *AB* grammars

As it is known, there are natural language expressions that admit more than one syntactic analysis. Evenmore, sometimes it is possible to use the inference rules in more than one pair of adjacent words.

Hence, we need to look for those pairs of words that may conform the initial subtrees of possible proof trees. Also, for each initial subtree we need to know the list of pairs $w : X$ on the left, and on the right of the subtree, these three elements will be called an environment.

For example, for the proof tree in Figure 4 its unique initial subtree is

$\dfrac{\dfrac{every}{((((t/e)/e)\backslash^{+}(t/e))^{-}/(t/e))} \quad \dfrac{ball}{(t/e)}}{(((t/e)/e)\backslash^{+}(t/e))}$, and its lists of pairs are $[Dobie : (t^{+}/(t/e)), didn't : ((t/e)^{-}/(t/e)), bring : ((t/e)/e)]$ on the left, and $[]$ on the right; the algorithm BFSTL returns a list of environments of the form $[(([w_1 : X_1, \ldots, w_{i_1-1} : X_{i_1-1}], \dfrac{\dfrac{w_{i_1}}{X_{i_1}} \quad \dfrac{w_{i_1+1}}{X_{i_1+1}}}{X_1}, [w_{i_1+2} : X_{i_1+2}, \ldots, w_n : X_n]), \ldots, ([w_1 : X_1, \ldots, w_{i_m-1} : X_{i_m-1}], \dfrac{\dfrac{w_{i_m}}{X_{i_m}} \quad \dfrac{w_{i_m+1}}{X_{i_m+1}}}{X_m}, [w_{i_m+2} : X_{i_m+2}, \ldots, w_n : X_n])]$, $1 \le i_1, i_m, m < n$.

```
FUNCTION BFSTL($left, current, envs$)
1   switch
2     case $current = []$ and $envs = []$ :
3         raise $NoFirstSubTree$
4     case $current = []$ :
5         return $envs$
6     case $current = [w_n : X_n]$ and $envs = []$ :
7         raise $NoFirstSubTree$
8     case $current = [w_n : X_n]$ :
9         return $envs$
10    case $current = [w_i : X_i, w_{i+1} : X_{i+1}, w_{i+2} : X_{i+2}, \ldots, w_n : X_n]$ :
11        if $X_i = X^s/X_{i+1}$ or $X_{i+1} = X_i\backslash^s X$
12          then BFSTL($left \uplus [w_i : X_i], [w_{i+1} : X_{i+1}, \ldots, w_n : X_n]$,
13               $envs \uplus [(left, \dfrac{\dfrac{w_i}{X_i} \quad \dfrac{w_{i+1}}{X_{i+1}}}{X}$,
14               $[w_{i+2} : X_{i+2}, \ldots, w_n : X_n])]$)
15          else BFSTL($left \uplus [w_i : X_i], [w_{i+1} : X_{i+1}, \ldots, w_n : X_n]$,
16               $envs$)
17  end
```

The algorithm BFSTL has three input parameters: $left$ the pairs $w_j : X_j, 1 \le j < i$ already processed; $current$ the pairs $w_k : X_k, i \le n \le n$ which are not being processed; and $envs$ the list of environments found until now.

In general (line 10), if some inference rule can be applied to X_i and X_{i+1} (line 11), then a recursive call is performed appending ($\uplus$) $left$ and the list $[w_i : X_i]$; stating that $[w_{i+1} : X_{i+1}, \ldots, w_n : X_n]$ is the new current, and appending $envs$ and the list with the environment found $[(left, \dfrac{\dfrac{w_i}{X_i} \quad \dfrac{w_{i+1}}{X_{i+1}}}{X}, [w_{i+2} : X_{i+2}, \ldots, w_n : X_n])]$ (lines 12-14).

If no rule can be applied (lines 15 and 16), a recursive call is made indicating that a word has been processed, and leaving $envs$ without change.

If no environment was found (lines 2 and 6) then the $NoFirstSubTree$ exception is raised (lines 3 and 7). If there are no more elements to process (line 4) or there is only one element (line 8), then the work has been done and the list of environments $envs$ is returned (lines 5 and 9).

The algorithm BAWPT builds a proof tree from a list of environments. If the list of environments is not empty (lines 4 and 5), then the algorithm BAPT is called with the elements of the first environment in the list, and the type of the root of the first subtree (line 6).

```
FUNCTION BAWPT($environmentsList$)
1   switch
2     case $environmentsList = []$ :
3         raise $NoProofTree$
4     case $environmentsList =$
5         $[(left_1, fst_1, right_1), \ldots, (left_n, fst_n, right_n)]$ :
6         $tree \leftarrow$ BAPT($left_1, fst_1, right_1$, EXTRACTTYPE($fst_1$))
7         if $tree = \dfrac{s}{X}$
8           then BAWPT($[(left_2, fst_2, right_2), \ldots,$
9                $(left_n, fst_n, right_n)]$)
10          else return $tree$
11  end
```

If the algorithm BAPT could not build a proof tree, then a recursive call is performed with the rest of the list of environments (lines 8 and 9). If the algorithm BAPT built a proof tree, this is returned (line 10). If the environment list is empty, then it was not possible to build a proof tree and the exception $NoProofTree$ is raised (lines 2 and 3).

ISSN 2007-9737

The algorithm EXTRACTTYPE merely returns the root of a unary tree (lines 2 and 3), or the root of a binary tree (lines 4 and 5). This is used in line 6 of the algorithm BAWPT.

```
FUNCTION EXTRACTTYPE($tree$)
1 switch
2   case $tree = \frac{s}{X}$ :
3     return $X$
4   case $tree = \frac{proofSubTree_l \quad proofSubTree_r}{X}$ :
5     return $X$
```

The purpose of the algorithm BAPT is to build a proof tree. It takes four arguments: the list $left$ of pairs $w : X$ on the left of the proof subtree $proofSubTree$, the proof subtree $proofSubTree$ already built, the list $right$ of pairs $w : X$ on the right of the proof subtree $proofSubTree$, and the type X of the root of the proof subtree $proofSubTree$.

If $left$ and $right$ are empty, then a proof tree has been constructed, and there is nothing more to process (lines 2 and 3).

If $left$ is not empty, but $right$ is, then to construct a new proof subtree is needed that the root X of $subProofTree$ can combine with the type X_i of the last element of $left$, this is possible when $X = X_i\backslash^s X'$ is the functor and X_i is the argument, or when X is the argument and $X_i = X'^s/X$ is the functor, if that is the case, then a recursive call is performed pointing out that: $w_i : X_i$ has been processed, the new proof subtree $\frac{\frac{w_i}{X_i} \quad proofSubTree}{X'}$ has been constructed, $right$ is still empty, the root of the new proof subtree is X' (lines 4-7).

```
FUNCTION BAPT($left, proofSubTree, right, X$)
1  switch
2    case $left = []$ and $right = []$ :
3      return $proofSubTree$
4    case $left = [w_1 : X_1, \ldots, w_i : X_i]$ and $right = []$ :
5      if $X = X_i\backslash^s X'$ or $X_i = X'^s/X$
6        then BAPT($[w_1 : X_1, \ldots, w_{i-1} : X_{i-1}]$,
7          $\frac{\frac{w_i}{X_i} \quad proofSubTree}{X'}$, $[], X'$)
8        else return $\frac{``"}{e}$
9    case $left = []$ and $right = [w_j : X_j, \ldots, w_n : X_n]$ :
10     if $X = X'^s/X_j$ or $X_j = X\backslash^s X'$
11       then BAPT($[]$, $\frac{proofSubTree \quad \frac{w_j}{X_j}}{X'}$,
12         $[w_{j+1} : X_{j+1}, \ldots, w_n : X_n], X'$)
13       else return $\frac{``"}{e}$
14   case $left = [w_1 : X_1, \ldots, w_i : X_i]$ and
15     $right = [w_j : X_j, \ldots, w_n : X_n]$ :
16     switch
17       case $X = X_i\backslash^s X'$ or $X_i = X'^s/X$ :
18         BAPT($[w_1 : X_1, \ldots, w_{i-1} : X_{i-1}]$,
19           $\frac{\frac{w_i}{X_i} \quad proofSubTree}{X'}$,
20           $[w_j : X_j, \ldots, w_n : X_n], X'$)
21       case $X = X'^s/X_j$ or $X_j = X\backslash^s X'$ :
22         BAPT($[w_1 : X_1, \ldots, w_i : X_i]$,
23           $\frac{proofSubTree \quad \frac{w_j}{X_j}}{X'}$,
24           $[w_{j+1} : X_{j+1}, \ldots, w_n : X_n], X'$)
25       case default :
26         return $\frac{``"}{e}$
27 end
```

If it was not possible to build a new proof subtree, then the tree $\frac{``"}{e}$ is returned (lines 8, 13 and 23), so that the algorithm BAWPT tries to build a proof tree with the remaining environments.

If $left$ is empty, but $right$ is not, then to construct a new proof subtree it is needed that the root X of $subProofTree$ can combine with the type X_j of the first element of $right$, this is possible when $X = X'^s/X_j$ is the functor and X_j is the argument, or when X is the argument and $X_j = X\backslash^s X'$ is the functor, if that is the case, then a recursive call is performed pointing out that: $left$ is still empty, the new proof subtree $\frac{proofSubTree \quad \frac{w_j}{X_j}}{X'}$ has been constructed, $w_j : X_j$ has been processed, the root of the new proof subtree is X' (lines 9-12).

ISSN 2007-9737

If $left$ and $right$ are not empty, then the root X of $proofSubTree$ can combine with the type X_i of the last element of $left$ (lines 17-20); or the root X of $proofSubTree$ can combine with the type X_j of the first element of $left$ (lines 21-24), just like the respective previous cases.

Finally, the algorithm BUILDPROOFTREE passes the appropriate initial values to BFSTL in order to get a list of environments, this list is passed to BAWPT, which constructs a proof tree.

```
FUNCTION BUILDPROOFTREE($nlexp$)
1  return BAWPT(BFSTL($[], nlexp, []$))
2  end
```

5 Automatic Theorem Proving for Natural Logic

To compute polarity in our extension to *AB* grammars, the algorithm of van Benthem is adapted as follows.

Algorithm 5.1. van Benthem's polarity algorithm adapted to an extension of *AB* grammars.

1. Label the root with +.
2. Propagate notations up the tree.
   (a) If a node of type X is labeled l and its children are of type (X^s/Y) and Y, then the former child is to be labeled l and the latter child is to be labeled $l \circ s$ (diagrammatically, $\frac{\underset{l}{(X^s/Y)} \quad \underset{l\circ s}{Y}}{\underset{l}{X}}$).
   (b) If a node of type X is labeled l and its children are of type Y and $(Y\backslash^s X)$, then the former child is to be labeled $l \circ s$ and the latter child is to be labeled l (diagrammatically, $\frac{\underset{l\circ s}{Y} \quad \underset{l}{(Y\backslash^s X)}}{\underset{l}{X}}$).

As an example with have the proof tree of Figure 5a.

The algorithm POLALG encodes the algorithm of van Benthem more precisely. It returns a proof tree with polarity marks. Its arguments are: the polarity label l for the root of $tree$, and the proof tree $tree$.

```
FUNCTION POLALG($l, tree$)
1   switch
2     case $tree = \frac{w}{X}$ :
3         return $\frac{w}{\underset{l}{X}}$
4     case $tree = \frac{\frac{\vdots}{(X^s/Y)} \quad \frac{\vdots}{Y}}{X}$ :
5         $lptp \leftarrow$ POLALG($l, \frac{\vdots}{(X^s/Y)}$)
6         $rptp \leftarrow$ POLALG($l \circ s, \frac{\vdots}{Y}$)
7         return $\frac{lptp \quad rptp}{\underset{l}{X}}$
8     case $tree = \frac{\frac{\vdots}{Y} \quad \frac{\vdots}{(Y\backslash^s X)}}{X}$ :
9         $lptp \leftarrow$ POLALG($l \circ s, \frac{\vdots}{Y}$)
10        $rptp \leftarrow$ POLALG($l, \frac{\vdots}{(Y\backslash^s X)}$)
11        return $\frac{lptp \quad rptp}{\underset{l}{X}}$
12  end
```

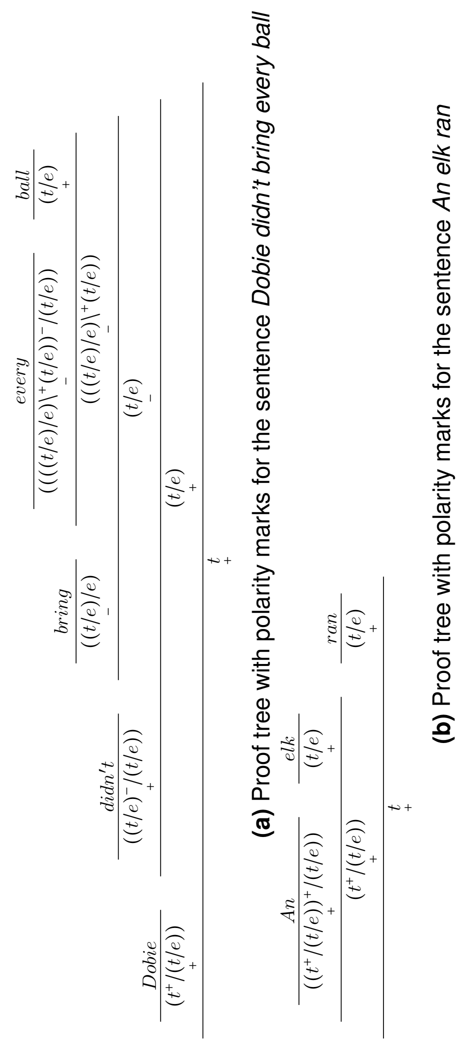
If the current tree is a unary tree, then it returns a unary tree marking the root with l (lines 2 and 3).

If the current tree is a binary tree and the functor is the left subtree, then it recursively propagates the polarity l on the left subtree $lptp$, and also it recursively propagates the polarity $l \circ s$ on the right subtree $rptp$. Finally it returns a binary tree marking the root with l, and having $lptp$ and $rptp$ as left and right subtrees, respectively (lines 4-7).

If the current tree is a binary tree and the functor is the right subtree, then it recursively propagates the polarity $l \circ s$ on the left subtree $lptp$, and also it recursively propagates the polarity l on the right subtree $rptp$. Finally it returns a binary tree marking the root with l, and having $lptp$ and $rptp$ as left and right subtrees, respectively (lines 8-11).

The algorithm POLARITY returns a proof tree with polarity marks, it receives as argument the representation of a natural language expression $nlexp$, as it was discussed in section 4. POLARITY calls POLALG with the mark of positive polarity and the proof tree for $nlexp$.

```
FUNCTION POLARITY($nlexp$)
1  return POLALG($+$, BUILDPROOFTREE($nlexp$))
2  end
```


$\frac{Dobie}{(t^+/(t/e))_+}$
$\frac{didn't}{((t/e)^-/(t/e))_+}$
$\frac{bring}{((t/e)/e)_-}$
$\frac{every}{((((t/e)/e)\backslash^+(t/e))^-/(t/e))_-}$
$\frac{ball}{(t/e)_+}$
$(((t/e)/e)\backslash^+(t/e))_-$
$(t/e)_-$
$(t/e)_+$
t_+
$\frac{An}{((t^+/(t/e))^+/(t/e))_+}$
$\frac{elk}{(t/e)_+}$
$(t^+/(t/e))_+$
$\frac{ran}{(t/e)_+}$
t_+

(a) Proof tree with polarity marks for the sentence *Dobie didn't bring every ball*

(b) Proof tree with polarity marks for the sentence *An elk ran*

Fig. 5. Proof trees with polarity marks using the adapted algorithm of van Benthem in an extension of *AB* grammars

Example 5.1. A proof tree with polarity marks for the natural language expression *An elk ran* is shown in Figure 5b.

If N = *An elk ran*, M = *elk* y M' = *mammal*, then we can say that *An elk ran* $\vdash$ *A mammal ran*, because *elk* has positive polarity, and *elk* $\leq_{(t/e)}$ *mammal*, because *elk* is a meronym of *mammal*. □

Example 5.2. If N = *An elk ran*, M = *elk* and M' = *animal* we can say that *An elk ran* $\vdash$ *An animal ran*, because *elk* has positive polarity and *elk* $\leq_{(t/e)}$ *animal*, because *elk* is a meronym of *animal*. □

Finally, we want to define an automatic theorem prover, to get this it is needed to chain entailments on more than one subexpression, this is done in the following way:

Definition 5.1. *Entailment.* Let N, and N' be two natural language expressions with $N \neq N'$, we define that N *entails* N' (symbolically $N \vdash N'$) as follows:

$$N \vdash N' \text{ if and only if}$$

$$\begin{cases} N' = N(M') \text{ and } N(M) \vdash N(M'), \text{ or} \\ N'' = N(M''), N(M) \vdash N(M'') \text{ and } N'' \vdash N'. \end{cases}$$

□

Now, we define the algorithms ENTAILS, ENTAILSALL, and ENTAILSONE.

The algorithm ENTAILS has as input the natural language expressions N, and N', it returns the result of the algorithm ENTAILSALL. The purpose of ENTAILS is to warrant that ENTAILSALL receives, in the set $DifSub$, all the pairs (M, M'), where M, M' are respectively subexpressions of N, N' that make $N \neq N'$. Also it receives $true$ the first time that it is called.

```
FUNCTION ENTAILS($N, N'$)
  Form the set $DifSub$ with all the pairs $(M, M')$,
  where $M, M'$ are subexpressions of $N, N'$
   such that $M \neq M'$
  return ENTAILSALL($DifSub, true$)
  end
```

ENTAILSALL tries to find counterexamples to the entailment of two natural language expressions. The algorithm ENTAILSALL codes Definition 5.1, it takes two parameters as input: $DifSub$ containing the pairs of subexpressions (M, M') that make different N and N', and the variable $flag$, which records (line 15) if a counterexample, that it is falsifying the entailment, has been found.

As it is implicit in Definition 3.5, a natural language expression $N(M)$ entails $N(M')$ if they vary in subexpressions M and M' of the same type, and $M \leq (\geq) M'$ according to their polarity.

```
FUNCTION ENTAILSALL($DifSub, flag$)
1  if $DifSub = \varnothing$
2     then return $flag$
3     else From $DifSub$ take a pair $(M, M')$
4        if $type(M) = type(M')$
5           then if not ENTAILSONE($M, M'$)
6              then if $polarity(M) = polarity(M')$
7                 then switch
8                    case $polarity(M) = +$ :
9                       write: $M \nleq M'$
10                   case $polarity(M) = -$ :
11                      write: $M' \nleq M$
12                   case default :
13                      write: There is not a relationship between $M$ and $M'$.
14                else write: There is a possible contradiction between $M$ and $M'$.
15                ENTAILSALL($DifSub - (M, M'), false \wedge flag$)
16             else ENTAILSALL($DifSub - (M, M'), true \wedge flag$)
17          else write: The syntactic structures of $M$ and $M'$ are not equal.
18             return $false$
19 end
```

Hence, the main purpose of ENTAILSALL is to process a pair (M, M') from $DifSub$ with the same type (lines 3 and 4), then if ENTAILSONE fails (line 5) and M has the same polarity as M' (line 6), it means that a counterexample has been found, and it writes the cause of the failure (lines 9, 11, 13, 14), then it calls itself removing the pair (M, M') from $DifSub$, and recording by $false \wedge flag$ that a counterexample was found (line 15).

If ENTAILSONE does not fail, then a recursive call is performed (line 16) removing the pair (M, M') from $DifSub$, and recording by $true \wedge flag$ that a counterexample was not found.

If subexpressions M, and M' have different types (line 17), then it is indicated that the subexpressions have not the same syntactic structure, because Natural Logic cannot reason with these kind of expressions, in this case the algorithm finishes returning $false$ (line 18).

When each pair of DifSub has been processed, the result of ENTAILSALL has to do with whether or not counterexamples have been found (lines 1 and 2).

The algorithm ENTAILSONE codes almost directly Definition 3.5, it takes subexpressions M and M', analysing if they meet the order relation according to their polarity.

```
FUNCTION ENTAILSONE($M, M'$)
1  switch
2     case $polarity(M) = +$ :
3        return $M \leq M'$
4     case $polarity(M) = -$ :
5        return $M' \leq M$
6     case default :
7        return $false$
8  end
```

To implement an automatic theorem prover, lexicons having pairs $word : type$ are needed, but, as far as we know, there are no such lexicons. Another possibility is to have a Part of Speech (POS) tagger that associates each word with its proper type.

The *C & C* tools [6] have a POS tagger, but it uses the inference rules of Combinatory Categorial Grammars, and there is not an algorithm to compute polarity for these kind of grammars, actually the known algorithms to compute polarity only work with Categorial Grammars which have the inference rules $\dfrac{X^s/Y \quad Y}{X}$, and $\dfrac{Y \quad Y\backslash^s X}{X}$ as the unique inference rules.

There are no domains partially ordered as it is supposed in section 3, therefore it is not possible to check if $M \leq_\alpha M'$, but it is possible to take advantage of tools such as WordNet [16], BabelNet [15], etc., to find synonyms, hyponyms, hyperonyms, meronyms, and troponyms.

6 Examples

At this time, we have a prototype that constructs possible counterexamples for the pair text-hypothesis of natural language expressions. It is implemented in Moscow ML version 2.10, for what has been discussed previously, the prototype asks the user for the veracity of the constructed relationships in the entailment process.

Example 6.1. *An elk ran* ⊢ *An animal moved*

```
- modChckNatLog([lxe("An", sl(u,
sl(u,t, sl(n,t,e)), sl(n, t,e ))),
lxe("elk", sl(n, t,e )), lxe("ran",
sl(n, t, e))],
[lxe("An", sl(u, sl(u,t, sl(n,t,e)),
sl(n, t,e ))),
lxe("animal", sl(n, t,e )),
lxe("moved", sl(n, t, e))]);

Is "ran" a kind of, or a manner of
"moved"?
Is "elk" a kind of, or a manner of
"animal"?
Done.
```

This exemplifies that a very specific statement can be generalized at the extreme that it loses information. Nevertheless, the entailment is true. □

Example 6.2. *Dobie brought every ball* ⊢ *Dobie brought every black ball*

```
- modChckNatLog([lxe("Dobie",sl(u,t,
sl(n,t,e))), lxe("brought",
sl(n,sl(n,t,e),e)),
lxe("every", sl(d,bs(u,sl(n,sl(n,t,e),e),
sl(n,t,e)),sl(n,t,e))),
lxe("ball",sl(n,t,e))],
[lxe("Dobie",sl(u,t,sl(n,t,e))),
lxe("brought",sl(n,sl(n,t,e),e)),
lxe("every", sl(d,bs(u,sl(n,sl(n,t,e),e),
sl(n,t,e)),sl(n,t,e))),
lxe("black",sl(u,sl(n,t,e),sl(n,t,e))),
lxe("ball",sl(n,t,e))]);

Is "black ball" a kind of, or a manner of
"ball"?
Done.
```

In this case we have the role of ”every” that is downward monotone on its first argument, therefore it sets the polarity of *ball* to negative. Hence it can be replaced with *black ball* that is a lesser expression. The entailment is true. □

Example 6.3. *Dobie didn't bring every ball* ⊢ *Dobie didn't bring every black ball*

```
- modChckNatLog([lxe("Dobie",sl(u,t,
sl(n,t,e))),
lxe("didn't", sl(d, sl(n,t,e), sl(n,t,e))),
lxe("bring",sl(n,sl(n,t,e),e)),
lxe("every", sl(d,bs(u,sl(n,sl(n,t,e),e),
sl(n,t,e)),
sl(n,t,e))), lxe("ball",sl(n,t,e))],
[lxe("Dobie",sl(u,t,sl(n,t,e))),
lxe("didn't", sl(d, sl(n,t,e),
sl(n,t,e))),
lxe("bring",sl(n,sl(n,t,e),e)),
lxe("every",sl(d,bs(u,sl(n,sl(n,t,e),e),
sl(n,t,e)),
sl(n,t,e))), lxe("black",sl(u,sl(n,t,e),
sl(n,t,e))),
lxe("ball",sl(n,t,e))]);

Is "ball" a kind of, or a manner of
"black ball"?
Done.
```

In this case the verb is negated, therefore it changes the polarity of the following constituents. Hence, the prototype asks if *ball* is a kind of *black ball*, i.e, if *ball* is lesser than *black ball*. Thence the entailment is false. □

Example 6.4. *Don't dig your grave with your own knife* ⊢ *Don't trench your grave with your own penknife*. For this example, refer to Figure 6.

```
- modChckNatLog([lxe("Don't",
sl(d, q ,q)),
lxe("dig", sl(u, q, sl(n, t, q)) ),
lxe("your", sl(u, sl(n, t, q), q)),
lxe("grave", q),
lxe("with", sl(u, bs(u, q, q),
sl(n, t, q))),
lxe("your", sl(u, sl(n, t, q), q)),
lxe("own", sl(u, q ,q)),
lxe("knife", q)],
```

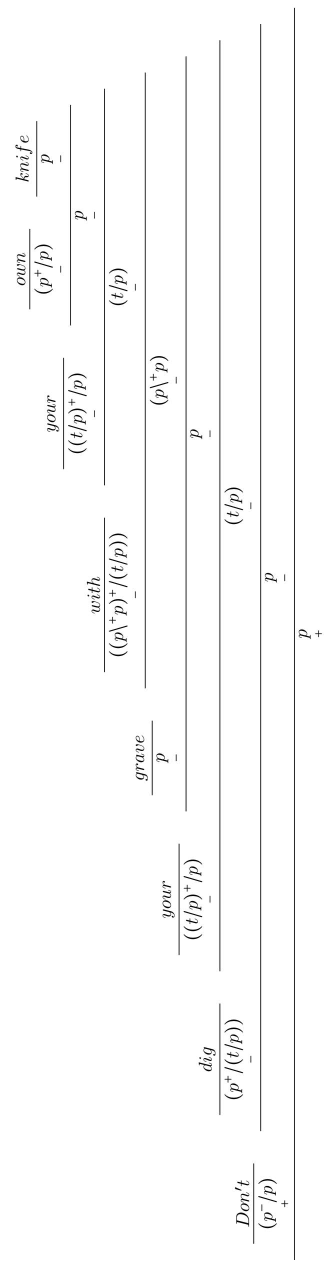


Fig. 6. Proof tree with polarity marks for the sentence *Don't dig your grave with your own knife*, in this tree $p = (t/e)$

```
[lxe("Don’t", sl(d, q ,q)),
lxe("trench", sl(u, q, sl(n, t, q)) ),
lxe("your", sl(u, sl(n, t, q), q)),
lxe("grave", q),
lxe("with", sl(u, bs(u, q, q),
sl(n, t, q))),
lxe("your", sl(u, sl(n, t, q), q)),
lxe("own", sl(u, q ,q)),
lxe("penknife", q)]);

Is "penknife" a kind of, or a manner of
"knife"?
Is "trench" a manner of "dig"?
Done.
```

If WordNet is consulted we find that *trench* is a direct troponym of *dig*. Also, that *penknife* is a hyponym of *knife*. The entailment is true. □

Example 6.5. *Don't dig your grave with your own knife* ⊢ *Don't trench your hole with your own penknife*

```
- modChckNatLog([lxe("Don’t",
sl(d, q ,q)),
lxe("dig", sl(u, q, sl(n, t, q)) ),
lxe("your", sl(u, sl(n, t, q), q)),
lxe("grave", q),
lxe("with", sl(u, bs(u, q, q),
sl(n, t, q))),
lxe("your", sl(u, sl(n, t, q), q)),
lxe("own", sl(u, q ,q)), lxe("knife", q)],
[lxe("Don’t", sl(d, q ,q)),
lxe("trench", sl(u, q, sl(n, t, q)) ),
lxe("your", sl(u, sl(n, t, q), q)),
lxe("hole", q),
lxe("with", sl(u, bs(u, q, q),
sl(n, t, q))),
lxe("your", sl(u, sl(n, t, q), q)),
lxe("own", sl(u, q ,q)),
lxe("penknife", q)]);

Is "penknife" a kind of, or a manner
of "knife"?
Is "hole" a kind of, or a manner
of "grave"?
Is "trench" a manner of "dig"?
Done.
```

penknife is a hyponym of *knife*, *trench* is a direct troponym of *dig*, but *hole* is an hypernym of *grave*, it is not an hyponym. The entailment is false.

ISSN 2007-9737

7 Conclusions and Future Work

We have developed an Automatic Theorem Prover for Natural Logic to Recognize Textual Entailment; this includes algorithms to: construct proof trees as the syntactic part of Natural Logic; compute polarity as the base of reasoning in Natural Logic; and look for subexpressions that falsify the entailment process.

The main advantage of the Automatic Theorem Prover is that it provides the list of counterexamples (pairs of subexpressions of the same type) that do not allow the entailment between two natural language expressions. As a consequence, the scope of Natural Logic in Recognizing Textual Entailment is restricted to pairs of expressions having the same syntactic structure.

As future work, in order to widen the scope of Natural Logic to Recognize Textual Entailment, it is desirable to be able to compare subexpressions of similar types; for example, the type of nouns is similar to the type of noun phrases.

Other points on the agenda for future work are: to construct lexicons where the words are associated with their types, to define an algorithm to compute polarity for Combinatory Categorial Grammars, and to build interfaces to take advantage of resources such as WordNet, and BabelNet.

Acknowledgements

This paper was partly supported by PRODEP-SEP under grant PROMEP/103.5/13/5618, by Benemérita Universidad Autónoma de Puebla under grant BUAP-803, and by CONACYT under the Thematic Networks program (Language Technologies Thematic Network projects 260178 and 271622).

We wish to thank anonymous reviewers for their comments, which helped improve the present manuscript.

References

1. **Ajdukiewicz, K. (1978).** Syntactic connexion (1936). In **Giedymin, J.**, editor, *The Scientific World-Perspective and Other Essays, 1931–1963*. Springer Netherlands, Dordrecht, pp. 118–139.
2. **Akhmatova, E. (2005).** Textual entailment resolution via atomic propositions. *Proceedings of the First PASCAL Challenges Workshop on Recognizing Textual Entailment*.
3. **Bach, E. (1988).** Categorial grammars as theories of language. In **Oehrle, R. T., Bach, E., & Wheeler, D.**, editors, *Categorial Grammars and Natural Language Structures*. Springer Netherlands, Dordrecht, pp. 17–34.
4. **Bos, J. & Markert, K. (2006).** Recognising textual entailment with robust logical inference. *MLCW 2005, volume LNAI 3944*, pp. 404–426.
5. **Bos, J. & Markert, K. (2006).** When logical inference helps determining textual entailment (and when it doesn't). *Proceedings of the Second Challenge Workshop, Recognizing Textual Entailment*, Pascal.
6. **Clark, S. & Curran, J. R. (2004).** Parsing the wsj using ccg and log-linear models. *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL, Association for Computational Linguistics, Stroudsburg, PA, USA.
7. **Dagan, I., Roth, D., Sammons, M., & Zanzotto, F. M. (2013).** *Recognizing Textual Entailment: Models and Applications*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.
8. **Dowty, D. (1994).** The role of negative polarity and concord marking in natural language reasoning. *Proceedings of the 4th. Conference on Semantics and Theoretical Linguistics*, Cornel University, CLC Publications, Rochester, NY.
9. **Hodges, D., Clark, C., Fowler, A., & Moldovan, D. (2006).** Applying COGEX to recognize textual entailment. In **Quiñonero-Candela, J., Dagan, I., Magnini, B., & d'Alché Buc, F.**, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, volume 3944 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 427–448.
10. **Icard, I., Thomas, F., & Moss, L. (2014).** Recent progress on monotonicity. *Linguistic Issues*

in Language Technology, Vol. 9, Perspectives on Semantic Representations for Textual Inference, pp. 167–194.

11. **Karttunen, L. (2015).** From natural logic to natural reasoning. In **Gelbukh, A.**, editor, *Computational Linguistics and Intelligent Text Processing*, volume 9041 of *Lecture Notes in Computer Science*. Springer International Publishing, pp. 295–309.

12. **MacCartney, B. (2009).** *Natural Language Inference*. Ph.D. thesis, Stanford University.

13. **MacCartney, B. & Manning, C. D. (2007).** Natural logic for textual inference. *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, Association for Computational Linguistics, Prague, pp. 193–200.

14. **Moot, R. & Retoré, C. (2012).** *The Logic of Categorial Grammars, A Deductive Account of Natural Language Syntax and Semantics*. Springer.

15. **Navigli, R. & Ponzetto, S. (2012).** BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, Vol. 193, pp. 217–250.

16. **Princeton University (2010).** Princeton University ”About WordNet”. `http://wordnet.princeton.edu`.

17. **Raina, R., Ng, A. Y., & Manning, C. D. (2005).** Robust textual inference via learning and abductive reasoning. **Veloso, M. M. & Kambhampati, S.**, editors, *AAAI*, AAAI Press / The MIT Press, pp. 1099–1105.

18. **Sánchez-Valencia, V. (1991).** *Studies on Natural Logic and Categorial Grammar*. Ph.D. thesis, Universiteit van Amsterdam.

19. **Steedman, M. & Baldridge, J. (2011).** Combinatory categorial grammar. In **Borsley, R. & Borjars, K.**, editors, *Non-Transformational Syntax: Formal and Explicit Models of Grammar*. Wiley-Blackwell.

20. **van Benthem, J. (1986).** *Essays in Logical Semantics*, volume 29 of *Studies in Linguistics and Philosophy*. Reidel, Dordrecht.

21. **van Benthem, J. (1991).** *Language in Action: Categories, Lambdas, and Dynamic Logic*, volume 130 of *Studies in Logic*. Elsevier, Amsterdam.

22. **van Benthem, J. (2007).** A brief history of natural logic. Technical report. Available at: `https://www.illc.uva.nl/Research/Publications/Reports/PP-2008-05.text.pdf`.

Article received on 06/08/2016; accepted on 15/10/2016.
Corresponding author is Jesús Lavalle.