Tableaus for Natural Logic

Reinhard Muskens

Tilburg Center for Logic and Philosophy of Science
`r.a.muskens@uvt.nl`
`http://let.uvt.nl/general/people/rmuskens/`

Abstract. In this paper we develop the beginnings of a tableau system for natural logic, the logic that is present in ordinary language and that us used in ordinary reasoning. The system is based on certain terms of the typed lambda calculus that can go proxy for linguistic forms and which we call Lambda Logical Forms. It is argued that proof-theoretic methods like the present one should complement the more traditional model-theoretic methods used in the computational study of natural language meaning.

1 Introduction

A standard approach to the semantics of natural language [17] provides language, or rather fragments of language, with a truth definition by means of translation into the language of some logic (such as Montague's IL) that already comes with one. The truth conditions of a translated sentence will then be identified with those of its translation. This also induces a relation of entailment on the translated fragment, for a sentence S can be taken to entail a sentence S′ if and only if the translation of the former entails that of the latter.

This provides a way to do automated inference on natural language. In order to check whether a given argument stated in ordinary language holds, its premises and conclusion are translated into logic with the help of some form of the typed lambda calculus, after which a theorem prover is invoked to do the actual testing. This procedure is described in [5] with great clarity and precision.

Here we will follow another route and define a tableau system that directly works on representations that are linguistically relevant. We will also place in focus tableau rules that are connected with certain properties of operators that seem important from a linguistic point of view. Our aim will not so much be to provide a proof system that is complete with respect to the semantics of our representations, but to provide rules that can be argued to come close to the rules implemented in human wetware. The purpose of this paper, therefore, is to contribute to the field of *natural logic*.[1]

[1] Early contributions to natural logic are [14] and [20]. The research line we base ourselves upon is exemplified in [9, 10, 2, 3, 19, 8, 4, 11, 22, 15, 16].

2 Lambda Logical Forms

For our purpose it will be of help to have representations of natural language expressions that are adequate both from a linguistic and from a logical point of view. At first blush, this may seem problematic, as it may be felt that linguistic and logic require completely different and competing properties from the representations they use, but in fact the typed lambda calculus provides what we need, or at least a good approximation to it. In order to obtain a class of terms with linguistic relevance we will restrict attention to those (simply typed) lambda terms that are built up from variables and *non-logical* constants, with the help of application and lambda abstraction and will delimit this class further by the restriction that only variables of individual type are abstracted over. The resulting terms, which will be called *Lambda Logical Forms* (LLFs), are often very close to linguistic expressions, as the following examples illustrate.

(1) a. `((a woman)walk)`
  b. `((if((a woman)walk))((no man)talk))`
  c. `(mary(think((if((a woman)walk))((no man)talk))))`
  d. `((a woman)(`λx`(mary(think((if(walk`$\,x$`))((no man)talk))))))`
  e. `(few man)`$\lambda x.$`(most woman)`$\lambda y.$`like`$\,xy$

The terms in (1) were built up in the usual way, but no *logical* constants, such as $=, \forall, \exists, \rightarrow, \wedge, \vee, \neg$ and the like, were used in their composition. The next section will make a connection between some of the non-logical constants used in (1) and logical ones, but this connection will take us from natural representations of linguistic expressions to rather artificial ones. Lambda terms containing no logical constants will therefore continue to have a special status.

Lambda Logical Forms come close to the Logical Forms that are studied in generative grammar. For example, in [13] trees such as the one in (2a) are found, strikingly similar to the λ-term in (2c).

(2) a. $[_{\mathrm{S}}[_{\mathrm{DP}}$ every linguist$][1[_{\mathrm{S}}$ John$[_{\mathrm{VP}}$ offended $t_1]]]]$
  b. `((every linguist)(`λx_1`(john(offend `x_1`))))`

3 A Natural Logic Tableau System

In this section we will discuss a series of rules for a tableau system directly based on LLFs. While tableau systems usually only have a handful of rules (roughly two for each logical operator under consideration), this system will be an exception. There will be many rules, many of them connected with special classes of expressions. Defining a system that comes even close to adequately describing what goes on in ordinary language will be a task far greater than what can be accomplished in a single paper and we must therefore contend ourselves with giving examples of rules that seem interesting. Further work should lead to less incomplete descriptions. Since the rules we consider typically are connected to some algebraic property or other (such as monotonicity or anti-additivity—see below), it will also be necessary to specify to which class of expressions each rule applies. Describing exactly, for example, which expressions are monotone increasing in any given language requires a lot of careful linguistic work and for the moment we will be satisfied with providing examples (here: `some`, `some N`, `every N`, `many N`, and `most N`).

Familiarity with the method of tableaus will be assumed. Our tableaus will be based upon a (signed variant of) the KE calculus ([6]).

3.1 Tableau Entries

We will work with signed tableaus in which entries can have one of the following forms.[2]

– If A is an LLF of type $\langle\vec{\alpha}\rangle$ and $\vec{C}$ is a sequence of constants or LLFs of types $\vec{\alpha}$, then $T\vec{C} : A$ and $F\vec{C} : A$ are tableau entries;
– If A and B are LLFs of type $\langle\vec{\alpha}\vec{\beta}\rangle$ and $\vec{a}$ is a sequence of constants of types $\vec{\beta}$ then $T\vec{a} : A \subset B$ and $F\vec{a} : A \subset B$ are tableau entries.

An entry $T\vec{C} : A$ ($F\vec{C} : A$) intuitively states that $A\vec{C}$ is true (false), while $T\vec{a} : A \subset B$ ($F\vec{a} : A \subset B$) states that it is true (false) that $(\lambda\vec{x}.A\vec{x}\vec{a}) \subset (\lambda\vec{x}.A\vec{x}\vec{a})$ (where the $\vec{x}$ are of types $\vec{\alpha}$). For example, Ti : `man` $\subset$ `talk` states that, as a matter of contingent fact, in world i all men are talking, while T : `sparrow` $\subset$ `bird` says that, in all worlds, all sparrows are birds.

3.2 Closure Rules

There will be two cases of outright contradiction in which a branch can be closed.

(3)
$$\begin{array}{c} T\vec{C} : A \\ F\vec{C} : A \\ | \\ \times \end{array} \qquad\qquad \begin{array}{c} F\vec{a} : A \subset A \\ | \\ \times \end{array}$$

3.3 Rules Deriving from the Format

The format we have chosen also validates some rules. First, we are only interested in LLFs up to $\beta\eta$ equivalence and lambda conversions can be performed at will. Second, the $X\vec{C} : A$ format (where X is T or F) validates the following rules.

(4)
$$\begin{array}{c} X\vec{C} : AB \\ | \\ XB\vec{C} : A \end{array} \qquad\qquad \begin{array}{c} XB\vec{C} : A \\ | \\ X\vec{C} : AB \end{array}$$

So we can shift arguments to the front and shift them back again.

[2] Types will be relational, as in [18].

3.4 The Principle of Bivalence

The KE calculus, which we base ourselves upon, allows for a limited version of the cut rule, called the *Principle of Bivalence* (PB). It runs as follows.

(5) $$\begin{tikzcd} & {} \arrow[dl, no head] \arrow[dr, no head] & \\ T\vec{C}:A & & F\vec{C}:A \end{tikzcd}$$ provided A and all $\vec{C}$ are already in the tableau.

The provision here is essential in order to maintain analyticity of the method. A should be a subterm of a term that already occurs in the tableau and all the C should also already be present (not as subterms).

Splitting a tableau is a very costly step in view of memory resources and if we want to devise a system that comes close to human reasoning (at the moment we are just exploring the logic behind such a system not developing such a system itself) we should start investigating under what conditions the human reasoner in fact takes this step. Here we have opted to let PB be our only tableau-splitting rule, as it is in the calculus KE.

3.5 Rules for $\subset$

The following rules seem reasonable for our inclusion statements.

(6) $$\begin{array}{c} T\vec{a}:A\subset B \\ T\vec{a}:B\subset C \\ | \\ T\vec{a}:A\subset C \end{array} \qquad \begin{array}{c} T\vec{a}:A\subset B \\ T\vec{C}\vec{a}:A \\ | \\ T\vec{C}\vec{a}:B \end{array} \qquad \begin{array}{c} F\vec{a}:A\subset B \\ | \\ T\vec{b}\vec{a}:A \\ F\vec{b}\vec{a}:B \end{array}$$

While the first two rules in (6) do not introduce any new material, the second does. The witnesses $\vec{b}$ that are introduced here must be fresh to the branch.

3.6 Hyponomy Rule

We will suppose that many basic entailments between words[3] are given in the lexicon and are freely available within the tableau system. This leads to the following rule.

(7) If $A\subset B$ is lexical knowledge: $$\begin{array}{c} | \\ T\vec{a}:A\subset B \end{array}$$

Tableau validity will thus be a notion that is dependent on the set of entailments that are considered lexical knowledge.

[3] In natural language there are entailment relations within many categories [12]. If $A\subset B$ is true in all models under consideration, we say that A entails B. For example, `sparrow` entails `bird` and `each` entails `most`.

3.7 Boolean Rules

We can now give rules for the operators `and`, `or` and `not`, the first two of which we write between their arguments, much as the rules for $\wedge$, $\vee$ and $\neg$ would be in a signed variant of the KE calculus. What is different here is that these rules are given for conjunction, disjunction and complementation in all categories, not just the category of sentences.

(8)
$$\begin{array}{ccc} T\vec{C} : A \ \texttt{and}\ B & F\vec{C} : A \ \texttt{and}\ B & F\vec{C} : A \ \texttt{and}\ B \\ | & T\vec{C} : A & T\vec{C} : B \\ T\vec{C} : A & | & | \\ T\vec{C} : B & F\vec{C} : B & F\vec{C} : A \end{array}$$

(9)
$$\begin{array}{ccc} F\vec{C} : A \ \texttt{or}\ B & T\vec{C} : A \ \texttt{or}\ B & T\vec{C} : A \ \texttt{or}\ B \\ | & F\vec{C} : A & F\vec{C} : B \\ F\vec{C} : A & | & | \\ F\vec{C} : B & T\vec{C} : B & T\vec{C} : A \end{array}$$

(10)
$$\begin{array}{cc} T\vec{C} : \texttt{not}\ A & F\vec{C} : \texttt{not}\ A \\ | & | \\ F\vec{C} : A & T\vec{C} : A \end{array}$$

Here is a tableau showing that `not(man or woman)` entails `(not man) and (not woman)`.

(11)
$$\begin{array}{c} Tci : \texttt{not(man or woman)} \\ Fci : \texttt{(not man) and (not woman)} \\ Fci : \texttt{man or woman} \\ Fci : \texttt{man} \\ Fci : \texttt{woman} \\ \begin{array}{cc} / & \backslash \\ Tci : \texttt{not man} & Fci : \texttt{not man} \\ Fci : \texttt{not woman} & Tci : \texttt{man} \\ Tci : \texttt{woman} & \times \\ \times & \end{array} \end{array}$$

In order to refute the possibility that some object c and some world i satisfy `not(man or woman)` but do not satisfy `(not man) and (not woman)` a tableau was developed which starts from the counterexample set

$$\{Tci : \texttt{not(man or woman)},\ Fci : \texttt{(not man) and (not woman)}\}\ .$$

Since the tableau closes the possibility is indeed refuted.

While `and`, `or` and `not` seem to be operative in all categories, `if` is sentential. We formulate its rules as follows. Note that sentences still need a parameter (here: i) since their type is $\langle s \rangle$, not just $\langle\rangle$.

(12)
$$\begin{array}{c} Ti : \texttt{if}\, AB \\ Ti : A \\ | \\ Ti : B \end{array} \qquad\qquad \begin{array}{c} Fi : \texttt{if}\, AB \\ | \\ Ti : A \\ Fi : B \end{array}$$

3.8 Rules for Monotonic Operators

The rules we have discussed until now were either completely general or operated on specific words (constants), but it has been observed that natural reasoning hinges on properties that attach to certain *groups* of expressions. Let us write $\subset_i$ for the relation that obtains between relations M and M' of the same type $\langle\vec{\gamma}s\rangle$ if $(\lambda\vec{x}.M\vec{x}i) \subset (\lambda\vec{x}.M'\vec{x}i)$. A relation A of type $\langle\langle\vec{\alpha}s\rangle\vec{\beta}s\rangle$ is called *upward monotone* if $\forall XY\forall i(X \subset_i Y \rightarrow AX \subset_i AY)$ (where X and Y are of type $\langle\vec{\alpha}s\rangle$). Examples of upward monotone expressions (already mentioned above) are `some`, `some N`, `every N`, `many N`, `most N` (where `N` varies over expressions of type $\langle es\rangle$), but also `Mary`. Here is a tableau rule for upward monotone (mon↑) expressions.

(13) If A is mon↑:
$$\begin{array}{c} T\vec{C}i : AB \\ Ti : B \subset B' \\ | \\ T\vec{C}i : AB' \end{array}$$

And here is a dual rule for expressions that are downward monotone, i.e. that satisfy the property $\forall XY\forall i(X \subset_i Y \rightarrow AY \subset_i AX)$. Examples are `no`, `no N`, `every`, `few`, and `few N`.

(14) If A is mon↓:
$$\begin{array}{c} T\vec{C}i : AB \\ Ti : B' \subset B \\ | \\ T\vec{C}i : AB' \end{array}$$

Using the second of these rules, the first tableau in Table 1 shows, by way of example, that `no bird moved` entails `no lark flew`.[4]

A central theme of [19] is that monotonicity reasoning is at the hart of traditional logic. The second tableau in Table 1 shows the validity of the syllogism known as *Disamis.*

The crucial step here makes use of the upward monotonicity of `some`. We have used a rule to the effect that `all` essentially is $\subset_i$ (where i is the current world) which will be introduced below.

3.9 Other Rules Connected to Algebraic Properties

Upward and downward monotonicity are not the only algebraic properties that seem to play a pivotal role in language. There is a literature starting with [23]

[4] We follow the convention, usual in type-logical work, that association in terms is to the left, i.e. ABC is short for $(AB)C$ (which in its turn is short for $((AB)C)$).

Ti : `no bird moved`	Ti : `some AB`
Fi : `no lark flew`	Ti : `all AC`
Ti : `flew` $\subset$ `moved`	Fi : `some CB`
Ti : `no bird flew`	Ti : `A` $\subset$ `C`
T`flew`, i : `no bird`	T`B`i : `some A`
F`flew`, i : `no lark`	T`B`i : `some C`
Ti : `lark` $\subset$ `bird`	Ti : `some CB`
T`flew`, i : `no lark`	$\times$
$\times$	

Table 1. Two Tableaus

singling out *anti-additivity*, as linguistically important. An operator A is anti-additive if it is downward monotone and satisfies the additional property that $\forall XY((AX \cap AY) \subset A(X \cup Y))$. Rules for anti-additive operators, examples of which are `no-one` and `without`, but also `not`, are easily given:

(15) If A is anti-additive:

$$\begin{array}{c} F\vec{C} : A(B \texttt{ or } B') \\ T\vec{C} : AB \\ | \\ F\vec{C} : AB' \end{array} \qquad \begin{array}{c} F\vec{C} : A(B \texttt{ or } B') \\ T\vec{C} : AB' \\ | \\ F\vec{C} : AB \end{array}$$

We can continue in this vein, isolating rules connected to semantic properties that have been shown to be linguistically important. For example, [7] mentions *splittingness*, $\forall XY(A(X \cup Y) \subset (AX \cup AY))$, and *having meet*, $\forall XY((AX \cap AY) \subset A(X \cap Y))$, which we can provide with rules as follows.

(16) If A has meet:

$$\begin{array}{c} F\vec{C} : A(B \texttt{ and } B') \\ T\vec{C} : AB \\ | \\ F\vec{C} : AB' \end{array} \qquad \begin{array}{c} F\vec{C} : A(B \texttt{ and } B') \\ T\vec{C} : AB' \\ | \\ F\vec{C} : AB \end{array}$$

(17) If A is splitting:

$$\begin{array}{c} T\vec{C} : A(B \texttt{ or } B') \\ F\vec{C} : AB \\ | \\ T\vec{C} : AB' \end{array} \qquad \begin{array}{c} T\vec{C} : A(B \texttt{ or } B') \\ F\vec{C} : AB' \\ | \\ T\vec{C} : AB \end{array}$$

`no N` and `every N` have meet, while `some N` is splitting.

3.10 Getting Rid of Boolean Operators

Many of the rules we have seen thus far allow one to get rid of Boolean operators, even if the operator in question is not the main operator in the LLF under consideration. Here are a few more. If a Boolean is the main connective in the functor of a functor-argument expression it is of course always possible to distribute it over the argument and Booleans can likewise be pulled out of lambda-abstractions.

(18)
$$\begin{array}{c} X\vec{C} : (A \;\texttt{and}\; A')B \\ | \\ X\vec{C} : AB \;\texttt{and}\; A'B \end{array} \qquad \begin{array}{c} X\vec{C} : (\lambda x.A \;\texttt{and}\; B) \\ | \\ X\vec{C} : (\lambda x.A) \;\texttt{and}\; (\lambda x.B) \end{array}$$

These rules were given for `and`, but similar rules for `or` and `not` are also obviously correct.

Other rules that help removing Booleans from argument positions are derivable from rules that are already present, as the reader may verify. Here are a few.

(19) If A is mon$\uparrow$:
$$\begin{array}{c} T\vec{C} : A(B \;\texttt{and}\; B') \\ | \\ T\vec{C} : AB \\ T\vec{C} : AB' \end{array} \qquad \begin{array}{c} F\vec{C} : A(B \;\texttt{or}\; B') \\ | \\ F\vec{C} : AB \\ F\vec{C} : AB' \end{array}$$

(20) If A is mon$\downarrow$:
$$\begin{array}{c} T\vec{C} : A(B \;\texttt{or}\; B') \\ | \\ T\vec{C} : AB \\ T\vec{C} : AB' \end{array} \qquad \begin{array}{c} F\vec{C} : A(B \;\texttt{and}\; B') \\ | \\ F\vec{C} : AB \\ F\vec{C} : AB' \end{array}$$

It is clear that not all cases are covered, but the rules allow us to get rid of `and` and `or` at least in *some* cases.

3.11 Rules for Determiners

Let us look at rules for determiners, terms of type $\langle\langle es\rangle\langle es\rangle s\rangle$. It has often been claimed that determiners in natural language all are *conservative*, i.e. have the property $\forall XY(DXY \equiv DX(X \cap Y))$ ([1]). Leaving the question whether really *all* determiners satisfy this property aside, we can establish that for those which do we can use the following tableau rule.

(21) If D is conservative:
$$\begin{array}{c} Xi : DA(A \;\texttt{and}\; B) \\ | \\ Xi : DAB \end{array}$$

This again is a rule that removes a Boolean operator from an argument position. Here is another. If determiners D and D' are *duals* (the pair `some` and `every` are prime examples), the following rule can be invoked. (We let $\overline{T} = F$ and $\overline{F} = T$.)

(22) If D and D' are duals:
$$\begin{array}{c} Xi : DA(\texttt{not}\, B) \\ | \\ \overline{X}i : D'AB \end{array}$$

The following rule applies to *contradictory* determiners, such as `some` and `no`.

(23) If D and D' are contradictories:
$$\begin{array}{c} Xi : DAB \\ | \\ \overline{X}i : D'AB \end{array}$$

There must also be rules for the logical determiners `every` and `some`. The first of these determiners is of course closely related to $\subset$ and we obtain the following.

(24)
$$\begin{array}{c} Xi : \texttt{every}\, AB \\ | \\ Xi : A \subset B \end{array}$$

The second may be given its own rules.

(25)
$$\begin{array}{c} Ti : \texttt{some}\, AB \\ | \\ Tbi : A \\ Tbi : B \end{array} \qquad \begin{array}{c} Fi : \texttt{some}\, AB \\ Tci : A \\ | \\ Fci : B \end{array} \qquad \begin{array}{c} Xi : \texttt{some}\, AB \\ | \\ Xi : \texttt{some}\, BA \end{array}$$

The b in the first rule must again be fresh to the branch. Such taking of witnesses typically leads to undecidability of the calculus and it would be an interesting topic of investigation how the linguistic system avoids the ‘bleeding and feeding’ loops that can result from the availability of such rules.

3.12 Further Rules

In a full paper we will add rules for the modal operators `may` and `must`, `think` and `know`. We will also consider rules that are connected to comparatives and other expressions.

4 Conclusion

One way to describe the semantics of ordinary language is by means of translation into a well-understood logical language. If the logical language comes with a model theory and a proof theory, the translation will then induce these on the fragment of language that is translated as well. A disadvantage of this procedure is that precise translation of expressions, taking heed of all their logical properties, often is difficult. Whole books have been devoted to the semantics of a few related words, but while this often was done with good reason and in some cases has led to enlightening results, describing language word by word hardly seems a good way to make progress. Tableau systems such as the one developed here provide an interesting alternative. They interface with the usual model theory, as developing a tableau can be viewed as a systematic attempt to find a model refuting the argument, but on the other hand they seem to give us a better chance in obtaining large coverage systems approximating natural logic. The format allows us to concentrate on rules that really seem linguistically important and squares well with using representations that are close to the Logical Forms in generative syntax.

References

1. J.F.A.K. van Benthem. Questions about Quantifiers. *Journal of Symbolic Logic*, 49:447–478, 1984.
2. J.F.A.K. van Benthem. *Essays in Logical Semantics*. Reidel, Dordrecht, 1986.
3. J.F.A.K. van Benthem. *Language in Action*. North-Holland, Amsterdam, 1991.
4. R. Bernardi. *Reasoning with Polarity in Categorial Type Logic*. PhD thesis, Utrecht University, 2002.
5. Patrick Blackburn and Johan Bos. *Representation and Inference for Natural Language. A First Course in Computational Semantics*. CSLI, 2005.
6. M. D'Agostino and M. Mondadori. The Taming of the Cut. Classical Refutations with Analytic Cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.
7. Jaap van der Does. *Applied Quantifier Logics*. PhD thesis, University of Amsterdam, 1992.
8. D. Dowty. The Role of Negative Polarity and Concord Marking in Natural Language Reasoning. In Mandy Harvey and Lynn Santelmann, editors, *Proceedings from SALT IV*, pages 114–144. Cornell University, Ithaca, 1994.
9. J. van Eijck. Generalized Quantifiers and Traditional Logic. In J. van Benthem and A. ter Meulen, editors, *Generalized Quantifiers in Natural Language*. Foris, Dordrecht, 1985.
10. J. van Eijck. Natural Logic for Natural Language. In B. ten Cate and H. Zeevat, editors, *TbiLLC 2005*, LNAI 4363, pages 216–230. Springer-Verlag, Berlin Heidelberg, 2007.
11. F. Fyodorov, Y. Winter, and N. Francez. Order-Based Inference in Natural Logic. *Logic Journal of the IGPL*, 11(4):385–416, 2003.
12. J. Groenendijk and M. Stokhof. Type-shifting rules and the semantics of interrogatives. In G. Chierchia, B. Partee, and R. Turner, editors, *Properties, Types and Meanings, vol. 2: Semantic Issues*, pages 21–68. Kluwer, 1989.
13. I. Heim and A. Kratzer. *Semantics in Generative Grammar*. Blackwell, Oxford, 1998.
14. G. Lakoff. Linguistics and Natural Logic. In D. Davidson and G. Harman, editors, *Semantics of Natural Language*, pages 545–665. Reidel, Dordrecht, 1972.
15. B MacCartney and C. Manning. Natural Logic for Textual Inference. In *ACL 2007 Workshop on Textual Entailment and Paraphrasing*, 2007.
16. B MacCartney and C. Manning. An Extended Model of Natural Logic. In H. Bunt, V. Petukhova, and S. Wubben, editors, *Proceedings of the 8th IWCS*, pages 140–156, Tilburg, 2009.
17. R. Montague. The Proper Treatment of Quantification in Ordinary English. In J. Hintikka, J. Moravcsik, and P. Suppes, editors, *Approaches to Natural Language*, pages 221–242. Reidel, Dordrecht, 1973. Reprinted in [21].
18. R.A. Muskens. *Meaning and Partiality*. CSLI, Stanford, 1995.
19. Víctor Sánchez. *Studies on Natural Logic and Categorial Grammar*. PhD thesis, University of Amsterdam, 1991.
20. F. Sommers. *The Logic of Natural Language*. The Clarendon Press, Oxford, 1982.
21. R. Thomason, editor. *Formal Philosophy, Selected Papers of Richard Montague*. Yale University Press, 1974.
22. Anna Zamansky, Nissim Francez, and Yoad Winter. A 'Natural Logic' Inference System Using the Lambek Calculus. *Journal of Logic, Language and Information*, 15:273–295, 2006.
23. F. Zwarts. Negatief-polaire Uitdrukkingen I. *Glot*, 6:35–132, 1981.