

Towards a Wide-Coverage Tableau Method for Natural Logic

Lasha Abzianidze^(✉)

Tilburg University - TiLPS, Tilburg, The Netherlands
L.Abzianidze@uvt.nl

Abstract. The first step towards a wide-coverage tableau prover for natural logic is presented. We describe an automatized method for obtaining Lambda Logical Forms from surface forms and use this method with an implemented prover to hunt for new tableau rules in textual entailment data sets. The collected tableau rules are presented and their usage is also exemplified in several tableau proofs. The performance of the prover is evaluated against the development data sets. The evaluation results show an extremely high precision above 97% of the prover along with a decent recall around 40%.

Keywords: Combinatory Categorical Grammar · Lambda Logical Form · Natural logic · Theorem prover · Tableau method · Textual entailment

1 Introduction

In this paper, we present a further development of the analytic tableau system for natural logic introduced by Muskens in [12]. The main goal of [12] was to initiate a novel formal method of modeling reasoning over linguistic expressions, namely, to model the reasoning in a signed analytic tableau system that is fed with Lambda Logical Forms (LLFs) of linguistic expressions. There are three straightforward advantages of this approach:

- (i) since syntactic trees of LLFs roughly describe semantic composition of linguistic expressions, LLFs resemble surface forms (that is characteristic for natural logic); hence, obtaining LLFs is easier than translating linguistic expressions in some logical formula where problems of expressiveness of logic and proper translation come into play;
- (ii) the approach captures an inventory of inference rules (where each rule is syntactically or semantically motivated and is applicable to particular linguistic phrases) in a modular way;
- (iii) a model searching nature of a tableau method and freedom of choice in a rule application strategy seem to enable us to capture *quick* inferences that humans show over linguistic expressions.

The rest of the paper is organized as follows. First, we start with the syntax of LLFs and show how terms of syntactic and semantic types can be combined; then we briefly discuss a method of obtaining LLFs from surface forms as we aim to develop a wide-coverage natural tableau system (i.e. a tableau prover for natural logic). A combination of automatically generated LLFs and an implemented natural tableau prover makes it easy to extract a relevant set of inference rules from the data used in textual entailment challenges. In the end, we present the performance of the prover on several training data sets. The paper concludes with a discussion of further research plans.

Throughout the paper we assume the basic knowledge of a tableau method.

2 Lambda Logical Forms

The analytic tableau system of [12] uses LLFs as logical forms of linguistic expressions. They are simply typed λ -terms with semantic types built upon $\{e, s, t\}$ atomic types. For example, in [12] the LLF of *no bird moved* is (1) that is a term of type st .¹ As we aim to develop a wide-coverage tableau for natural logic, using only terms of semantic types does not seem to offer an efficient and elegant solution. Several reasons for this are given below.

$$(\text{no}_{(est)(est)st} \text{bird}_{est}) \text{moved}_{est} \quad (1)$$

$$(\text{no}_{n,(np,s),s} \text{bird}_n) \text{moved}_{np,s} \quad (2)$$

First, using only terms of the semantic types will violate the advantage (i) of the approach. This becomes clear when one tries to account for event semantics properly in LLFs as it needs an introduction of an event entity and *closure* or *existential closure* operators of [3,14] that do not always have a counterpart on a surface level.

Second, semantic types provide little syntactic information about the terms. For instance, bird_{est} and moved_{est} are both of type est in [12], hence there is no straightforward way to find out their syntactic categories. Furthermore, $M_{(est)est}H_{est}$ term can stand for adjective and noun, adverb and intransitive verb, or even noun and complement constructions. The lack of syntactic information about a term makes it impossible to find a correct tableau rule for the application to the term, i.e. it is difficult to meet property (ii). For example, for $A_{est}B_e$, it would be unclear whether to use a rule for an intransitive verb that introduces an event entity and a thematic relation between the event constant and B_e ; or for $M_{(est)set}H_{est}$ whether to use a rule for adjective and noun or noun and complement constructions.²

¹ Hereafter we assume the following standard conventions while writing typed λ -terms: a type of a term is written in a subscript unless it is omitted, a term application is left-associative, and a type constructor comma is right-associative and is ignored if atomic types are single lettered.

² The latter two constructions have the same semantic types in the approach of [2], who also uses the C&C parser, like us, for obtaining logical forms but of first-order logic. The reason is that both PP and N categories for prepositional phrases and nouns, respectively, are mapped to et type.

Finally, a sentence generated from an open branch of a tableau proof can give us an explanation about failure of an entailment, but we will lose this option if we stay only with semantic types as it is not clear how to generate a grammatical sentence using only information about semantic types.³

In order to overcome the lack of syntactic information and remain LLFs similar to surface forms, we incorporate syntactic types and semantic types in the same type system. Let $\mathcal{A} = \{e, t, \mathbf{s}, \mathbf{np}, \mathbf{n}, \mathbf{pp}\}$ be a set of atomic types, where $\{e, t\}$ and $\{\mathbf{s}, \mathbf{np}, \mathbf{n}, \mathbf{pp}\}$ are sets of semantic and syntactic atomic types, respectively. Choosing these particular syntactic types is motivated by the syntactic categories of Combinatory Categorical Grammar (CCG) [13]. In contrast to the typing in [12], we drop *s* semantic type for states for simplicity reasons. Let $\mathcal{I}_{\mathcal{A}}$ be a set of all types, where complex types are constructed from atomic types in a usual way, e.g., $(\mathbf{np}, \mathbf{np}, \mathbf{s})$ is a type for a transitive verb. A type is called semantic or syntactic if it is constructed purely from semantic or syntactic atomic types, respectively; there are also types that are neither semantic nor syntactic, e.g., *ees*. After extending the type system with syntactic types, in addition to (1), (2) also becomes a well-typed term. For better readability, hereafter, we will use a boldface style for lexical constant terms with syntactic types.

The interaction between syntactic and semantic types is expressed by a subtyping relation (\sqsubseteq) that is a partial order, and for any $\alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathcal{I}_{\mathcal{A}}$:

- (a) $e \sqsubseteq \mathbf{np}$, $\mathbf{s} \sqsubseteq t$, $\mathbf{n} \sqsubseteq et$, $\mathbf{pp} \sqsubseteq et$;
- (b) $(\alpha_1, \alpha_2) \sqsubseteq (\beta_1, \beta_2)$ iff $\beta_1 \sqsubseteq \alpha_1$ and $\alpha_2 \sqsubseteq \beta_2$

The introduction of subtyping requires a small change in typing rules, namely, if $\alpha \sqsubseteq \beta$ and A is of type α , then A is of type β too. From this new clause it follows that a term $A_{\alpha}B_{\beta}$ is of type γ if $\alpha \sqsubseteq (\beta, \gamma)$. Therefore, a term can have several types, which are partially ordered with respect to \sqsubseteq , with the least and greatest types. For example, a term $\mathbf{love}_{\mathbf{np}, \mathbf{np}, \mathbf{s}}$ is also of type *eet* (and of other five types too, where $(\mathbf{np}, \mathbf{np}, \mathbf{s})$ and *eet* are the least and greatest types, respectively). Note that all atomic syntactic types are subtypes of some semantic type except $e \sqsubseteq \mathbf{np}$. The latter relation, besides allowing relations like $(\mathbf{np}, \mathbf{s}) \sqsubseteq et$, also makes sense if we observe that any entity can be expressed in terms of a noun phrase (even if one considers event entities, e.g., $\mathbf{singing}_{\mathbf{NP}}$ is difficult).

Now with the help of this *multiple typing* it is straightforward to apply $\mathbf{love}_{\mathbf{np}, \mathbf{np}, \mathbf{s}}$ $\mathbf{mary}_{\mathbf{np}}$ term to c_e constant, and there is no need to introduce new terms $\mathbf{love}_{\mathbf{eet}}$ and \mathbf{mary}_e just because $\mathbf{love}_{\mathbf{eet}}$ \mathbf{mary}_e is applicable to c_e . For the same reason it is not necessary to introduce $\mathbf{man}_{\mathbf{et}}$ for applying to c_e as $\mathbf{man}_{\mathbf{n}}$ c_e is already a well-formed term. From the latter examples, it is obvious that some syntactic terms (i.e. terms of syntactic type) can be used as semantic terms, hence minimize the number of terms in a tableau. Nevertheless, sometimes it will be inevitable to introduce a new term since its syntactic counterpart is not able to give a fine-grained semantics: if $\mathbf{red}_{\mathbf{n}, \mathbf{n}}$ $\mathbf{car}_{\mathbf{n}}$ c_e is evaluated as true, then

³ An importance of the explanations is also shown by the fact that recently SemEval-2015 introduced a pilot task *interpretable STS* that requires systems to explain their decisions for semantic textual similarity.

one has to introduce \mathbf{red}_{et} term in order to assert the redness of c_e by the term $\mathbf{red}_{et}c_e$ as $\mathbf{red}_{n,n}c_e$ is not typable. Finally, note that terms of type \mathbf{s} can be evaluated either as true or false since they are also of type t .

Incorporating terms of syntactic and semantic types in one system can be seen as putting together two inference engines: one basically using syntactically-rich structures, and another one semantic properties of lexical entities. Yet another view from Abstract Categorical Grammars [7] or Lambda Grammars [11] would be to combine abstract and semantic levels, where terms of syntactic and semantic types can be seen as terms of abstract and semantic levels respectively, and the subtyping relation as a sort of simulation of the morphism between abstract and semantic types.⁴

3 Obtaining LLFs from CCG Trees

Automated generation of LLFs from unrestricted sentences is an important part in the development of the wide-coverage natural tableau prover. Combined with the implemented tableau prover, it facilitates exploring textual entailment data sets for extracting relevant tableau rules and allows us to evaluate the theory against these data sets.

We employ the C&C tools [4] as an initial step for obtaining LLFs. The C&C tools offer a pipeline of NLP systems like a POS-tagger, a chunker, a named entity recognizer, a super tagger, and a parser. The tools parse sentences in CCG framework with the help of a statistical parser. Altogether the tools are very efficient and this makes them suitable for wide-coverage applications [1]. In the current implementation we use the statistical parser that is trained on the *rebanked* version of CCGbank [8].

In order to get a semantically adequate LLF from a CCG parse tree (see Fig. 1), it requires much more effort than simply translating CCG trees to syntactic trees of typed lambda terms. There are two main reasons for this complication: (a) a trade-off that the parser makes while analyzing linguistic expressions in order to tackle unrestricted texts, and (b) accumulated wrong analyses in final parse trees introduced by the various C&C tools.

For instance, the parser uses combinatory rules that are not found in the CCG framework. One of such kind of rules is a lexical rule that simply changes a CCG category, for example, a category N into NP (see $1x[np, n]$ combinatory rule in Fig. 1) or a category $S \setminus NP$ into $N \setminus N$. The pipeline of the tools can also introduce wrong analyses at any stage starting from the POS-tagger (e.g., assigning a wrong POS-tag) and finishing at the CCG parser (e.g., choosing a wrong combinatory rule). In order to overcome (at least partially) these problems, we use a pipeline consisting of several filters and transformation procedures. The general structure of the pipeline is the following:

⁴ The connection between LLFs of [12] and the terms of an abstract level was already pointed out by Muskens in the project’s description “Towards logics that model natural reasoning”.

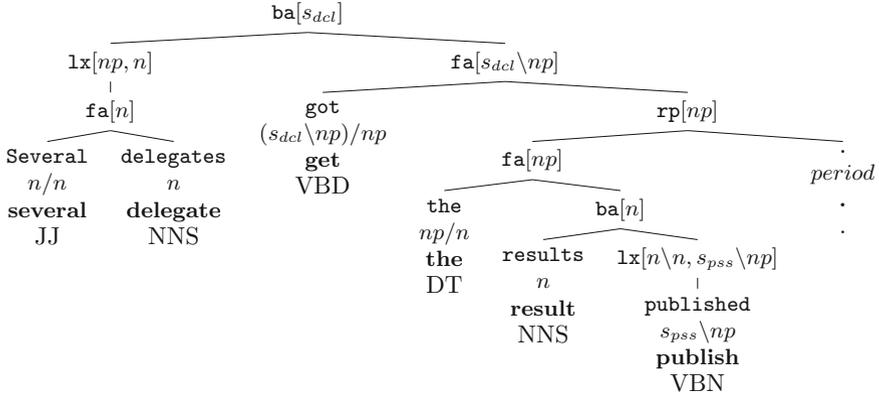


Fig. 1. A parse tree of several delegates got the results published. by the C&C parser

- *Transforming a CCG tree into a CCG term*: the procedure converts CCG categories in types by removing directionality from CCG categories (e.g., $S \backslash NP / NP \rightsquigarrow (\mathbf{np}, \mathbf{np}, \mathbf{s})$) and reordering tree nodes in a corresponding way.
- *Normalizing the CCG term*: since an obtained CCG term can be considered as a typed λ -term, it is possible to reduce it to $\beta\eta$ -normal form.⁵
- *Identifying proper names*: if both function and argument terms are recognized as proper names by the C&C pipeline, then the terms are concatenated; for instance, $\mathbf{Leonardo}_{n,n}(\mathbf{da}_{n,n} \mathbf{Vinci}_n)$ is changed in a constant term $\mathbf{Leonardo_da_Vinci}_n$ if all three terms are tagged as proper names.
- *Identifying multiword expressions (MWE)*: the CCG parser analyzes in a purely compositional way all phrases including MWEs like *a lot of*, *take part in*, *at least*, etc. To avoid these meaningless analyses, we replace them with constant terms (e.g., $\mathbf{a_lot_of}$ and $\mathbf{take_part_in}$).
- *Correcting syntactic analyses*: this procedure is the most complex and extensive one as it corrects a CCG term by inserting, deleting or replacing terms. For example, the type shifts like $n \rightsquigarrow \mathbf{np}$ are fixed by inserting corresponding determiners (e.g., $(\mathbf{oil}_n)_{\mathbf{np}} \rightsquigarrow \mathbf{a}_{n,\mathbf{np}} \mathbf{oil}_n$) or by typing terms with adequate types (e.g., $(\mathbf{Leonardo_da_Vinci}_n)_{\mathbf{np}} \rightsquigarrow \mathbf{Leonardo_da_Vinci}_{\mathbf{np}}$ and $(\mathbf{several}_{n,n} \mathbf{delegate}_n)_{\mathbf{np}} \rightsquigarrow \mathbf{several}_{n,\mathbf{np}} \mathbf{delegate}_n$). More extensive corrections, like fixing a syntactically wrong analysis of a relative clause, like (3), are also performed in this procedure.
- *Type raising of quantifiers*: this is the final procedure, which takes a more or less fixed CCG term and returns terms where quantified noun phrases of type \mathbf{np} have their types raised to $((\mathbf{np}, \mathbf{s}), \mathbf{s})$. As a result several LLFs are returned due to a scope ambiguity among quantifiers. The procedure makes sure that

⁵ Actually the obtained CCG term is not completely a λ -term since it may contain type changes from lexical rules. For instance, in $(\mathbf{several}_{n,n} \mathbf{delegate}_n)_{\mathbf{np}}$ subterm, $(\cdot)_{\mathbf{np}}$ operator changes a type of its argument into \mathbf{np} . Nevertheless, this kind of type changes are accommodated in the λ -term normalization calculus.

generalized quantifiers are applied to the clause they occur in if they do not take a scope over other quantifiers. For example, from a CCG term (4) only (5) is obtained and (6) is suppressed.

$$\mathbf{old}_{n,n}(\mathbf{who}_{(np,s),n,n} \mathbf{cry}_{np,s} \mathbf{man}_n) \rightsquigarrow \mathbf{who}_{(np,s),n,n} \mathbf{cry}_{np,s}(\mathbf{old}_{n,n} \mathbf{man}_n) \quad (3)$$

$$\mathbf{and}_{s,s,s}(\mathbf{sleep}_{np,s} \mathbf{john}_{np})(\mathbf{snore}_{np,s}(\mathbf{no}_{n,np} \mathbf{man}_n)) \quad (4)$$

$$\mathbf{and}_{s,s,s}(\mathbf{sleep}_{np,s} \mathbf{john}_{np})(\mathbf{no}_{n,(np,s),s} \mathbf{man}_n \mathbf{snore}_{np,s}) \quad (5)$$

$$\mathbf{no}_{n,(np,s),s} \mathbf{man}_n(\lambda x. \mathbf{and}_{s,s,s}(\mathbf{sleep}_{np,s} \mathbf{john}_{np})(\mathbf{snore}_{np,s} x_{np})) \quad (6)$$

The above described pipeline takes a single CCG tree generated from the C&C tools and returns a list of LLFs. For illustration purposes CCG term (7), which is obtained from the CCG tree of Fig. 1, and two LLFs, (8) and (9), generated from (7) are given below; here, *vp* abbreviates *(np, s)* and *s_{n, vp, s}* term stands for the plural morpheme.

$$\mathbf{got}_{np, vp}(\mathbf{s}_{n, np}(\mathbf{who}_{vp, n, n}(\mathbf{be}_{vp, vp} \mathbf{publish}_{vp}) \mathbf{result}_n))(\mathbf{several}_{n, np} \mathbf{delegate}_n) \quad (7)$$

$$\mathbf{several}_{n, vp, s} \mathbf{delegate}_n(\lambda x. \mathbf{s}_{n, vp, s}(\mathbf{who}_{vp, n, n}(\mathbf{be}_{vp, vp} \mathbf{publish}_{vp}) \mathbf{result}_n)(\lambda y. \mathbf{got}_{np, vp} y_{np} x_{np})) \quad (8)$$

$$\mathbf{s}_{n, vp, s}(\mathbf{who}_{vp, n, n}(\mathbf{be}_{vp, vp} \mathbf{publish}_{vp}) \mathbf{result}_n)(\lambda x. \mathbf{several}_{n, vp, s} \mathbf{delegate}_n(\mathbf{got}_{np, vp} x_{np})) \quad (9)$$

4 An Inventory of Natural Tableau Rules

The first collection of tableau rules for the natural tableau was offered in [12], where a wide range of rules are presented including Boolean rules, rules for algebraic properties (e.g., monotonicity), rules for determiners, etc. Despite this range of rules, they are insufficient for tackling problems found in textual entailment data sets. For instance, problems that only concentrate on quantifiers or Boolean operators are rare in the data sets. Syntactically motivated rules such as rules for passive and modifier-head constructions, structures with the copula, etc. are fruitful while dealing with wide-coverage sentences, and this is also confirmed by the problems found in entailment data sets. It would have been a quite difficult and time-consuming task to collect these syntactically motivated tableau rules without help of an implemented prover of natural tableau. For this reason the first thing we did was to implement a natural tableau prover, which could prove several toy entailment problems using a small inventory of rules mostly borrowed from [12].⁶ With the help of the prover, then it was easier to explore manually the data sets and to introduce new rules in the prover that help it to further build tableaux and find proofs.

⁶ Implementation of the prover, its computational model and functionality is a separate and extensive work, and it is out of scope of the current paper.

During collecting tableau rules we used a half portion of the FraCaS test suite [5] and the part of the SICK trial data [10] as a development set.⁷ The reason behind opting for these data sets is that they do not contain long sentences, hence there is a higher chance that a CCG tree returned by the C&C tools will contain less number of wrong analyses, and it is more likely to obtain correct LLFs from the tree. Moreover, the FraCas test suite is considered to contain difficult entailment problems for textual entailment systems since its problems require more complex semantic reasoning than simply paraphrasing or relation extraction. We expect that interesting rules can be discovered from this set.

Hereafter, we will use several denotations while presenting the collected tableau rules. Uppercase letters A, B, C, \dots and lowercase letters a, b, c, \dots stand for meta variables over LLFs and constant LLFs, respectively. A variable letter with an arrow above it stands for a sequence of LLFs corresponding to the register of the variable (e.g. \vec{C} is a sequence of LLFs). Let $[\]$ denote an empty sequence. We assume that exp is a variable type that can be either np or e and that vp abbreviates (np, s) . Let $(-, \alpha) \in \mathcal{I}_{\mathcal{A}}$ for any $\alpha \in \mathcal{A}$ where the final (i.e. the most right) atomic type of $(-, \alpha)$ is α ; for instance $(-, s)$ can be s , (np, s) , (vp, vp) , etc. While writing terms we may omit their types if they are irrelevant for discussions, but often the omitted types can be inferred from the context the term occurs in. Tableau rules will be followed by the names that are the current names of the rules in the natural tableau prover. The same rule names with different subscripts mean that these rules are implemented in the prover by a single rule with this name. For instance, both $mod_n_tr_1$ and $mod_n_tr_2$ are implemented by a single rule mod_n_tr in the prover. Finally, we slightly change the format of nodes of [12]; namely, we place an argument list and a sign on the right side of a LLF – instead of $Tci : man$ we write $man : ci : \mathbb{T}$. We think that the latter order is more natural.

4.1 Rules from [12]

Most rules of [12] are introduced in the prover. Some of them were changed to more efficient versions. For example, the two rules deriving from `format` are modified and introduced in the prover as `pull_arg`, `push_arg1`, and `push_arg2`. These versions of the rules have more narrow application ranges. Hereafter, we assume that \mathbb{X} can match both \mathbb{T} and \mathbb{F} signs.

$$\frac{\lambda x. A : c \vec{C} : \mathbb{X}}{(\lambda x. A) c : \vec{C} : \mathbb{X}} \text{ pull_arg} \quad \frac{A c_e : \vec{C} : \mathbb{X}}{A : c_e \vec{C} : \mathbb{X}} \text{ push_arg}_1 \quad \frac{A c_{np} : \vec{C} : \mathbb{X}}{A : c_{np} \vec{C} : \mathbb{X}} \text{ push_arg}_2$$

$$A : c_e \vec{C} : \mathbb{X}$$

The Boolean rules and rules for monotonic operators and determiners (namely, `some`, `every`, and `no`) are also implemented in the prover. It might be said that these rules are one of the crucial ones for almost any entailment problem.

⁷ The Fracas test suite can be found at <http://www-nlp.stanford.edu/~wcmac/downloads>, and the SICK trial data at <http://alt.qcri.org/semEval2014/task1/index.php?id=data-and-tools>.

4.2 Rules for Modifiers

One of the most frequently used set of rules is the rules for modifiers. These rules inspire us to slightly change the format of tableau nodes by adding an extra slot for *memory* on the left side of an LLF:

$$memorySet : LLF : argumentList : truthSign$$

An idea of using a memory set is to save modifiers that are not directly attached to a head of the phrase. Once a LLF becomes the head without any modifiers, the memory set is discharged and its elements are applied to the head. For example, if we want to entail **beautiful car** from **beautiful red car**, then there should be a way of obtaining (11) from (10) in a tableau. It is obvious how to produce (12) from (10) in the tableau settings, but this is not the case for producing (11) from (10), especially, when there are several modifiers for the head.

$$\mathbf{beautiful}_{n,n}(\mathbf{red}_{n,n}\mathbf{car}_n) : c_e : \mathbb{T} \quad (10)$$

$$\mathbf{beautiful}_{n,n}\mathbf{car}_n : c_e : \mathbb{T} \quad (11)$$

$$\mathbf{red}_{n,n}\mathbf{car}_n : c_e : \mathbb{T} \quad (12)$$

With the help of a memory set, **beautiful**_{n,n} can be saved and retrieved back when the bare head is found. Saving *subsecutive* adjectives in a memory is done by `mod_n_tr1` rule while retrieval is processed by `mods_noun1` rule. In Fig. 2a, the closed tableau employs the latter rules in combination with `int_mod_tr` and proves that (10) entails (11).⁸

$$\begin{array}{l} \text{if } b \text{ is subsecutive: } \frac{\mathcal{M} : b_{n,n}A : c_e : \mathbb{T}}{\mathcal{M} \cup \{b_{n,n}\} : A : c_e : \mathbb{T}} \text{ mod_n_tr}_1 \quad \frac{\mathcal{M} \cup \{m_{n,n}\} : a_n : c_e : \mathbb{T}}{m_{n,n}a_n : c_e : \mathbb{T}} \text{ mods_noun}_1 \\ \\ \text{if } b \text{ is intersective: } \frac{\mathcal{M} : b_{n,n}A : c_e : \mathbb{T}}{\mathcal{M} : A : c_e : \mathbb{T}} \text{ int_mod_tr} \quad \frac{b_{n,n}A : c_e : \mathbb{F}}{A : c_e : \mathbb{F} \quad A : c_e : \mathbb{T} \quad b_{et} : c_e : \mathbb{F}} \text{ int_mod_fl} \end{array}$$

Hereafter, if a rule do not employ memory sets of antecedent nodes, then we simply ignore the slots by omitting them from nodes. The same applies to precedent nodes that contain an empty memory set. In rule `int_mod_tr`, a memory of a premise node is copied to one of conclusion nodes while rule `int_mod_fl` attaches empty memories to conclusion nodes, hence, they are omitted. The convention about omitting memory sets is compatible with rules found in [12].

⁸ It is not true that `mod_n_tr1` always gives correct conclusions for the constructions similar to (10). In case of **small beer glass** the rule entails **small glass** that is not always the case, but this can be avoided in the future by having more fine-grained analysis of phrases (that **beer glass** is a compound noun), richer semantic knowledge about concepts and more restricted version of the rule; currently rule `mod_n_tr1` can be considered as a default rule for analyzing this kind of constructions.

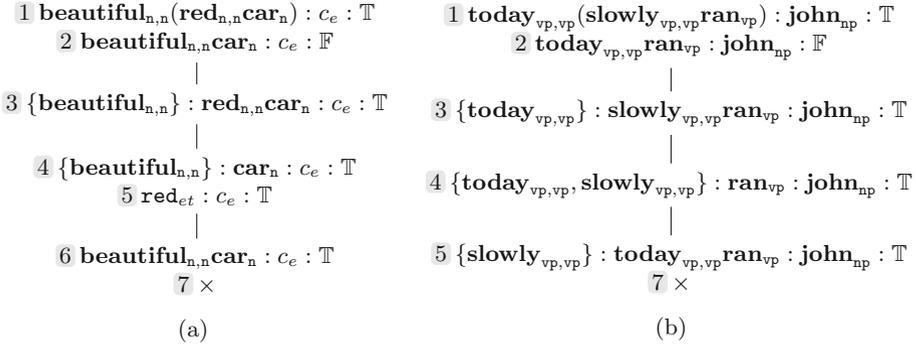


Fig. 2. Tableaux that use rules for pulling and pushing modifiers in a memory: (a) beautiful red car \Rightarrow beautiful car; (b) john ran slowly today \Rightarrow john ran today

$$\frac{\mathcal{M} : B_{vp,vp} A : \vec{C} : \mathbb{T}}{\mathcal{M} \cup \{B_{vp,vp}\} : A : \vec{C} : \mathbb{T}} \text{ mod_push} \qquad \frac{\mathcal{M} \cup \{B_{vp,vp}\} : A : \vec{C} : \mathbb{T}}{\mathcal{M} : B_{vp,vp} A : \vec{C} : \mathbb{T}} \text{ mod_pull}$$

if p is a preposition:

$$\frac{\mathcal{M} \cup \{p_{np,vp,vp} d_{ep}\} : A_n : c_e : \mathbb{T}}{p_{et} d_e : c_e : \mathbb{T}} \text{ mods_noun}_2 \qquad \frac{\mathcal{M} : p_{np,n,n} d_{ep} A_n : c_e : \mathbb{T}}{\mathcal{M} : A_n : c_e : \mathbb{T}} \text{ pp_mod_n}$$

$$p_{np,n,n} d_{ep} A_n : c_e : \mathbb{T} \qquad p_{et} d_e : c_e : \mathbb{T}$$

The other rules that save a modifier or discharge it are `mod_push`, `mod_pull` and `mods_noun2`. They do this job for any LLF of type (vp, vp) . For instance, using these rules (in conjunction with other rules) it is possible to prove that (13) entails (14); moreover, the tableau in Fig. 2b employs `push_mod` and `pull_mod` rules and demonstrates how to capture an entailment about events with the help of a memory set without introducing an event entity.

Yet another rules for modifiers are `pp_mod_n`, `n_pp_mod` and `aux_verb`. If a modifier of a noun is headed by a preposition like in the premise of (15), then `pp_mod_n` rule can treat a modifier as an intersective one, and hence capture entailment (15). In the case when a propositional phrase is a complement of a noun, rule `n_pp_mod` treats the complement as an intersective property and attaches the memory to the noun head. This rule with `mod_n_tr1` and `mods_noun1` allows the entailment in (16).⁹

$$\text{in}_{np,vp,vp} \text{paris}_{np} (\lambda x. \text{a}_{n,vp,s} \text{tourist}_n (\lambda y. \text{is}_{np,vp} y_{np} x_{np})) \text{john}_{np} \quad (13)$$

$$\text{in}_{np,n,n} \text{paris}_{np} \text{tourist}_n \text{john}_e \quad (14)$$

⁹ Note that the phrase in (16) is wrongly analyzed by the CCG parser; the correct analysis is $\text{for}_{np,n,n} C_{np}(\text{nobel}_{n,n} \text{prize}_n)$. Moreover, entailments similar to (16) are not always valid (e.g. $\text{short}_{n,n}(\text{man}_{pp,n}(\text{in}_{np,pp} \text{netherlands}_{np})) \not\Rightarrow \text{short}_{n,n} \text{man}_n$). Since the parser and our implemented filters, at this stage, are not able to give correct analysis of noun complementation and post-nominal modification, we adopt `n_pp_mod` as a default rule for these constructions.

$$\mathbf{in}_{np,n,n} \mathbf{paris}_{np} \mathbf{tourist}_n \mathbf{john}_e \Rightarrow \mathbf{in}_{et} \mathbf{paris}_e \mathbf{john}_e \quad (15)$$

$$\mathbf{nobel}_{n,n} (\mathbf{prize}_{pp,n} (\mathbf{for}_{np,pp} C_{np})) \Rightarrow \mathbf{nobel}_{n,n} \mathbf{prize}_n \quad (16)$$

Problems in data sets rarely contain entailments involving the tense, and hence `aux_verb` is a rule that ignores auxiliary verbs and an infinitive particle `to`. In Fig. 4, it is shown how `aux_verb` applies to 4 and yields 5. The rule also *accidentally* accounts for predicative adjectives since they are analyzed as $\mathbf{be}_{vp,vp} P_{vp}$, and when `aux_verb` is applied to a copula-adjective construction, it discards the copula. The rule can be modified in the future to account for tense and aspect.

$$\frac{\mathcal{M} : d_{pp,n} A_{pp} : c_e : \mathbb{T}}{\mathcal{M} : d_n : c_e : \mathbb{T} \quad A_{pp} : c_e : \mathbb{T}} \text{ n_pp_mod} \quad \frac{\mathcal{M} : b_{(-,s),(-,s)} A : \vec{C} : \mathbb{X}}{\mathcal{M} : A : \vec{C} : \mathbb{X}} \text{ aux_verb}$$

where $b \in \{\mathbf{do}, \mathbf{will}, \mathbf{be}, \mathbf{to}\}$

4.3 Rules for the Copula *be*

The copula `be` is often considered as a semantically vacuous word and, at the same time, it is sometimes a source of introduction of the equality relation in logical forms. Taking into account how the equality complicates tableau systems (e.g., a first-order logic tableau with the equality) and makes them inefficient, we want to get rid of `be` in LLFs whenever it is possible. The first rule that ignores the copula was already introduced in the previous subsection.

$$\text{If } p_{np,pp} \text{ is a preposition: } \frac{\mathcal{M} : \mathbf{be}_{pp,np,s} (p_{np,pp} c_{exp}) : d_{exp} : \mathbb{X}}{\mathcal{M} : p_{np,pp} c_{exp} : d_e : \mathbb{X} \quad p_{et} : c_e d_e : \mathbb{X}} \text{ be_pp}$$

The second rule that does the removal of the copula is `be_pp`. It treats a propositional phrase following the copula as a predicate, and, for example, allows to capture the entailment in (17). Note that the rule is applicable with both truth signs, and the constants c and d are of type e or np .

$$\mathbf{be}_{pp,np,s} (\mathbf{in}_{np,pp} \mathbf{paris}_{np}) \mathbf{john}_{np} \Rightarrow \mathbf{in}_{np,pp} \mathbf{paris}_{np} \mathbf{john}_e \quad (17)$$

The other two rules `a_subj_be` and `be_a_obj` apply to NP-be-NP constructions and introduce LLFs with a simpler structure. If we recall that quantifier terms like $\mathbf{a}_{n,vp,s}$ and $\mathbf{s}_{n,vp,s}$ are inserted in a CCG term as described in Sect. 3, then it is clear that there are many quantifiers that can introduce a fresh constant; more fresh constants usually mean a larger tableau and a greater choice in rule application strategies, which as a result decrease chances of finding proofs. Therefore, these two rules prevent tableaux from getting larger as they avoid introduction of a fresh constant. In Fig. 3, the tableau uses `be_a_obj` rule as the first rule application. This rule is also used for entailing (14) from (13).

$$\text{If } a \in \{\mathbf{a}, \mathbf{the}\} \text{ and } c \neq \mathbf{there} \quad \frac{\mathcal{M} : a_{n, \text{vp}, \text{s}} N_n(\mathbf{be} \ c_{\text{exp}}) : [] : \mathbb{X}}{\mathcal{M} : N_n : c_e : \mathbb{X}} \text{ a_subj_be}$$

$$\text{If } a \in \{\mathbf{a}, \mathbf{s}, \mathbf{the}\} \text{ and } c \neq \mathbf{there} \quad \frac{\mathcal{M} : a_{n, \text{vp}, \text{s}} N_n(\lambda x. \mathbf{be} \ x_{\text{exp}} \ c_{\text{exp}}) : [] : \mathbb{X}}{\mathcal{M} : N_n : c_e : \mathbb{X}} \text{ be_a_obj}$$

4.4 Rules for the Definite Determiner *the*

We have already presented several new rules in the previous section that apply to certain constructions with the copula and the determiner *the*. Here we give two more rules that are applicable to a wider range of LLFs containing the.

Since the definite determiner presupposes a unique referent inside a context, rule *the_c* requires two nodes to be in a tableau branch: the node with the definite description and the node with the head noun of this definite description. In case these nodes are found, the constant becomes the referent of the definite description, and the verb phrase is applied to it. The rule avoids introduction of a fresh constant. The same idea is behind rule *the* but it introduces a fresh constant when no referent is found on the branch. The rule is similar to the one for existential quantifier *some* of [12], except that *the* is applicable to false nodes as well due to the presupposition attached to the semantics of *the*.

$$\frac{\mathcal{M} : \mathbf{the}_{n, \text{vp}, \text{s}} N V : [] : \mathbb{X} \quad N : d_e : \mathbb{T}}{\mathcal{M} : V : d_e : \mathbb{X}} \text{ the_c} \quad \frac{\mathcal{M} : \mathbf{the}_{n, \text{vp}, \text{s}} N V : [] : \mathbb{X}}{N : c_e : \mathbb{T} \quad \mathcal{M} : V : c_e : \mathbb{X}} \text{ the where } c_e \text{ is fresh}$$

4.5 Rules for Passives

Textual entailment problems often contain passive paraphrases, therefore, from the practical point of view it is important to have rules for passives too. Two rules for passives correspond to two types the CCG parser can assign to a by-phrase: either *pp* while being a complement of VP, or (*vp*, *vp*) – being a VP modifier. Since these rules model paraphrasing, they are applicable to nodes with both signs. In Fig. 4, nodes 6 and 7 are obtained by applying *vp_pass2* to 5.

$$\frac{\mathcal{M} : V_{\text{pp}, \text{vp}} (\mathbf{by}_{\text{np}, \text{pp}} C_{\text{exp}}) : D_{\text{exp}} : \mathbb{X}}{\mathcal{M} : V_{\text{np}, \text{vp}} : D_{\text{exp}} C_{\text{exp}} : \mathbb{X}} \text{ vp_pass}_1 \quad \frac{\mathcal{M} : \mathbf{by}_{\text{np}, \text{vp}, \text{vp}} C_{\text{exp}} V_{\text{vp}} : D_{\text{exp}} : \mathbb{X}}{\mathcal{M} : V_{\text{np}, \text{vp}} : D_{\text{exp}} C_{\text{exp}} : \mathbb{X}} \text{ vp_pass}_2$$

4.6 Closure Rules

In general, closure rules identify or introduce an inconsistency in a tableau branch, and they are sometimes considered as closure conditions on tableau branches. Besides the revised version of the closure rule $\perp \leq$ found in [12], we add three new closure rules to the inventory of rules.

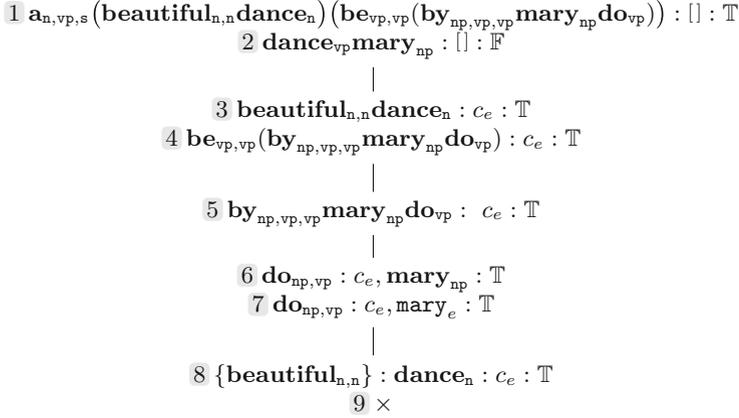


Fig. 4. A tableau proof for a beautiful dance was done by Mary \Rightarrow Mary danced

5 Evaluation

In order to demonstrate the productivity of the current inventory of tableau rules, we present the performance of the prover on the development set. As it was already mentioned in Sect. 4, we employ the part of the SICK trial data (100 problems) and a half of the FraCaS data (173 problems) as the development set. In these data sets, problems have one of three answers: entailment, contradiction, and neutral. Many entailment problems contain sentences that are long but have significant overlap in terms of constituents with other sentences. To prevent the prover from analyzing the common chunks (that is often unnecessary for finding the proof), we combine the prover with an optional simple aligner that aligns LLFs before a proof procedure. The prover also considers only a single LLF (i.e. semantic reading) for each sentence in a problem. Entailment relations between lexical words are modeled by the hyponymy and hypernymy relations of WordNet-3.0 [6]: $term_1 \leq term_2$ holds if there is a sense of $term_1$ that is a hyponym of some sense of $term_2$.

We evaluate the prover against the FraCaS development set (173 problems) and the whole SICK trial data (500 problems). The confusion matrix of the performance over the FraCaS development set is given in white columns of Table 1. As it is shown the prover was able to prove 31 true entailment and 2 contradiction problems. From all 173 problems, 18 problems are categorized as defected since LLFs of those problems were not obtained properly – either the parser could not parse a sentence in a problem or it parsed the sentences but they were not comparable as their CCG categories were different. If we ignore the obvious errors from the parser by excluding the defected problems, the prover

due to the poor treatment of cardinals by the prover – there is no distinction between them. The second problem was identified as entailment since *cry* may also have a meaning of *shout*. The last one was proved because the prover used LLFs, where *no hat* and a *backwards hat* had the widest scopes.

6 Future Work

Our future plan is to continue enriching the inventory of tableau rules. Namely, the SICK training data is not yet explored entirely, and we expect to collect several (mainly syntax-driven) rules that are necessary for *unfolding* certain LLFs. We also aim to further explore the FraCaS data and find the ways to accommodate in natural tableau settings semantic phenomena contained in plurals, comparatives, anaphora, temporal adverbials, events and attitude verbs.

At this stage it is early to compare the current performance of the prover to those of other entailment systems. The reason is that entailment systems usually do not depend on a single classifier but the combination of several shallow (e.g., word overlap) or deep (e.g., using semantically relevant rules) classifiers. In the future, we plan to make the prover more robust by combining it with a shallow classifier or by adding default rules (not necessarily sound) that are relevant for a development data set, and then compare it with other entailment systems. We also intend to employ other parsers for improving the LLF generator and knowledge bases for enriching the lexical knowledge of the prover.

Acknowledgements. I would like to thank Reinhard Muskens for his discussions and continuous feedback on this work. I also thank Matthew Honnibal, James R. Curran and Johan Bos for sharing the retrained CCG parser and anonymous reviewers of LENLS11 for their valuable comments. The research is part of the project “Towards Logics that Model Natural Reasoning” and supported by the NWO grant (project number 360-80-050).

References

1. Bos, J., Clark, S., Steedman, M., Curran, J.R., Hockenmaier, J.: Wide-coverage semantic representations from a CCG parser. In: Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004), pp. 1240–1246 (2004)
2. Bos, J.: Towards a large-scale formal semantic lexicon for text processing. from form to meaning: processing texts automatically. In: Proceedings of the Biennial GSCL Conference, pp. 3–14 (2009)
3. Champollioni, L.: Quantification and negation in event semantics. In: Baltic International Yearbook of Cognition, Logic and Communication, vol. 6 (2010)
4. Clark, S., Curran, J.R.: Wide-coverage efficient statistical parsing with CCG and log-linear models. *Comput. Linguist.* **33**(4), 493–552 (2007)
5. Cooper, R., Crouch, D., van Eijck, J., Fox, C., van Genabith, J., Jaspars, J., Kamp, H., Milward, D., Pinkal, M., Poesio, M., Pulman, S.: Using the framework. Technical Report LRE 62–051 D-16. The FraCaS Consortium (1996)

6. Fellbaum, Ch. (ed.): *WordNet: An Electronic Lexical Database*. MIT press, Cambridge (1998)
7. de Groot, Ph.: Towards abstract categorial grammars. In: *Proceedings of the Conference on ACL 39th Annual Meeting and 10th Conference of the European Chapter*, pp. 148–155 (2001)
8. Honnibal, M., Curran, J.R., Bos, J.: Rebanking CCGbank for improved NP interpretation. In: *Proceedings of the 48th Meeting of the Association for Computational Linguistics (ACL)*, pp. 207–215 (2010)
9. MacCartney, B., Manning, C.D.: Modeling semantic containment and exclusion in natural language inference. In: *Proceedings of Coling-2008, Manchester, UK* (2008)
10. Marelli, M., et al.: A sick cure for the evaluation of compositional distributional semantic models. In *Proceedings of LREC, Reykjavik* (2014)
11. Muskens, R.: Language, lambdas, and logic. In: Kruijff, G., Oehrle, R. (eds.) *Resource-Sensitivity, Binding and Anaphora. Studies in Linguistics and Philosophy*, vol. 80, pp. 23–54. Springer, Heidelberg (2003)
12. Muskens, R.: An analytic tableau system for natural logic. In: Aloni, M., Bastiaanse, H., de Jager, T., Schulz, K. (eds.) *Logic, Language and Meaning. LNCS*, vol. 6042, pp. 104–113. Springer, Heidelberg (2010)
13. Steedman, M., Baldrige, J.: *Combinatory Categorical Grammar*. In: Borsley, R.D., Borjars, K. (eds.) pp. 181–224. Blackwell Publishing (2011)
14. Winter, Y., Zwarts, J.: Event semantics and abstract categorial grammar. In: Kanazawa, M., Kornai, A., Kracht, M., Seki, H. (eds.) *MOL 12. LNCS*, vol. 6878, pp. 174–191. Springer, Heidelberg (2011)