LangPro: Natural Language Theorem Prover

Abzianidze, Lasha

LANGPRO: Natural Language Theorem Prover

Lasha Abzianidze
CLCG, University of Groningen
The Netherlands
L.Abzianidze@rug.nl

Abstract

LangPro is an automated theorem prover for natural language.[1] Given a set of premises and a hypothesis, it is able to prove semantic relations between them. The prover is based on a version of analytic tableau method specially designed for natural logic. The proof procedure operates on logical forms that preserve linguistic expressions to a large extent. The nature of proofs is deductive and transparent. On the FraCaS and SICK textual entailment datasets, the prover achieves high results comparable to state-of-the-art.

1 Introduction

Nowadays many formal logics come with their own proof systems and with the automated theorem provers based on these systems. If we share Montagues's famous belief that there is "no important theoretical difference between natural languages and the artificial languages of logicians", then there plausibly exists a proof system for natural languages too. On the other hand, studies on Natural Logic seek a formal logic whose formulas are as close as possible to linguistic expressions. Inspired by these research ideas, Muskens (2010) proposed an analytic tableau system for natural logic, where higher-order logic based on a simple type theory is used as natural logic and a version of analytic tableau method is designed for it. Later, Abzianidze (2015b,a, 2016a) made the tableau system suitable for wide-coverage reasoning by extending it and implementing a theorem prover based on it.

This paper presents the Prolog implementation of the theorem prover, called LangPro, in detail and completes the previous publications in terms of the system description. The rest of the paper is organized as follows. First, we briefly introduce the tableau system and the employed natural logic. Then we characterize the architecture and functionality of LangPro (see Figure 1). Before concluding, we briefly compare the prover to the related textual entailment systems.

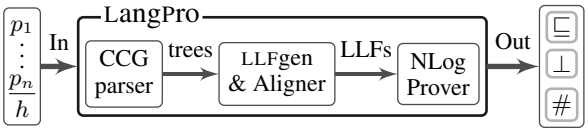

Figure 1: LangPro checks whether a set of premises $p_1, \ldots, p_n$ entails ($\sqsubseteq$), contradicts ($\bot$) or is neutral ($\#$) to a hypothesis h.

2 Natural Tableau

An *analytic tableau method* is a proof procedure which searches a model, i.e. a possible situation, satisfying a set of logic formulas. The search is performed by gradually applying inference rules, also called *tableau rules*, to the formulas. A tableau rule has antecedents and consequent and is easy to read, e.g., according to $\text{NO}_{\mathbb{T}}$ in Figure 3, if no A is B, then for any entity c, either it is not A or it is not B. A tableau proof, in short a *tableau*, is often depicted as an upside-down tree with initial formulas at its root (Figure 2). After each rule application, new inferred formulas are introduced in the tableau. Depending on the applied rules, the tableau can branch or grow in depth. A tableau branch models a situation that satisfies all the formulas in the branch. Closed branches, marked with $\times$, correspond to inconsistent situations. The search for a possible situation fails if all branches are closed—the tableau is closed.

The natural tableau is a tableau method for a

[1] https://github.com/kovvalsky/LangPro

Proceedings of the 2017 EMNLP System Demonstrations, pages 115–120
Copenhagen, Denmark, September 7–11, 2017.

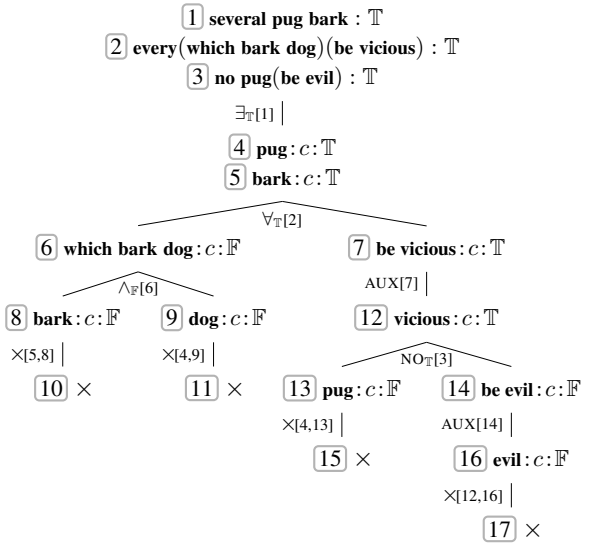

Figure 2: The tableau proves: *several pugs bark. every dog which barks is vicious.* $\perp$ *no pug is evil.*

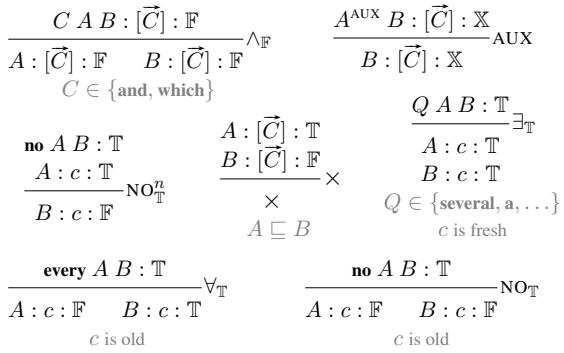

Figure 3: The inference rules employed in the tableau proof of Figure 2. An entity term is *old* (*fresh*) wrt a branch iff it is (not) in the branch.

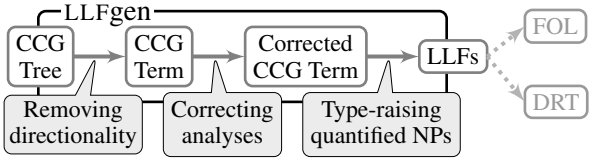

Figure 4: The LLF generator produces a list of LLFs from a single CCG derivation tree.

version of natural logic.[2] The terms of the natural logic, called *Lambda Logical Forms* (LLFs), are simply typed λ-terms built up from variables and constant lexical terms with the help of function application and λ-abstraction. The format of a tableau entry, i.e. node, is a tuple consisting of a modifier list, an LLF, an argument list and a truth sign. The parts are delimited with a colon. The empty lists are omitted for conciseness. For example, the entries (1) and (2) both mean that it is true that c barks loudly in Paris, where (α, β) is a functional type that expects an argument of type α and returns a value of type β.[3]

$$\mathbf{in}_{\mathtt{np},\mathtt{vp},\mathtt{vp}}\mathbf{Paris}_{\mathtt{np}} : \mathbf{loudly}_{\mathtt{vp},\mathtt{vp}}\mathbf{bark}_{\mathtt{vp}} : c_e : \mathbb{T} \quad (1)$$

$$(\mathbf{in}_{\mathtt{np},\mathtt{vp},\mathtt{vp}}\mathbf{Paris}_{\mathtt{np}})(\mathbf{loudly}_{\mathtt{vp},\mathtt{vp}}\mathbf{bark}_{\mathtt{vp}}\ c_e) : \mathbb{T} \quad (2)$$

In order to prove a certain logical relation between premises and a hypothesis, the natural tableau searches a situation for the counterexample of the relation. The relation is proved if the situation is not found, otherwise it is refuted. An example of a closed natural tableau is shown in Figure 2. It proves the contradiction relation as it fails to find a situation for the counterexample—the premises and the hypothesis being true. In order to facilitate reading tableau proofs, type information is omitted, the entries are enumerated and arcs are labeled with tableau rule applications. For example, 4 and 5 are obtained by applying $\exists_{\mathbb{T}}$ to 1: if it is true that several pugs bark, then there is some entity c which is a pug and which barks.

3 LLF Generator

A Natural Tableau-based theorem prover for natural language requires automatic generation of LLFs from raw text. To do so, we implement a module, called LLFgen, that generates LLFs from syntactic derivations of Combinatory Categorial Grammar (CCG, Steedman 2000). Given a CCG derivation, LLFgen returns several LLFs that model different orders of quantifier scopes (see Figure 4).[4] Figure 5 displays a CCG derivation where VP_i abbreviates $S_i\backslash NP$.

LLFs are obtained from a CCG tree in three major steps (Figure 4): (i) removing directionality from CCG trees, (ii) correcting semantically inadequate analyses, and (iii) type-raising quantified NPs (QNPs). Below we briefly describe each of these steps and give corresponding examples.

Directionality information encoded in CCG categories and combinatory rules is redundant from a semantic perspective, therefore we discard it in

[2] It is an extended version of Muskens' original tableau system. The extension is three-fold and concerns the type system, the format of tableau entries and the inventory of tableau rules (Abzianidze, 2015b).

[3] LLFs are typed with syntactic and semantic types. Interaction between these types is established via the subtyping relation, e.g., entities being a subtype of NPs, $e <: \mathtt{np}$, makes $\mathbf{bark}_{\mathtt{vp}}\ c_e$ well-formed, where vp abbreviates $(\mathtt{np}, \mathtt{s})$.

[4] See Abzianidze (2016a, Ch. 3) for a detailed description.

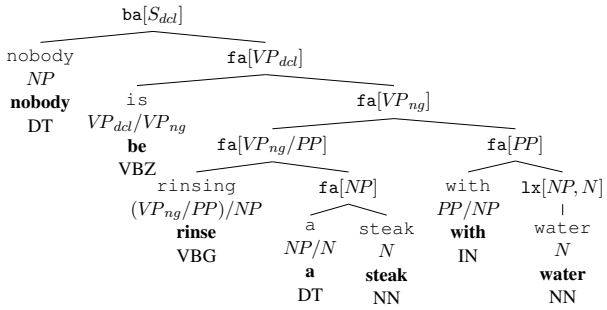

Figure 5: The CCG tree by C&C for *nobody is rinsing a steak with water* (SICK-1379).

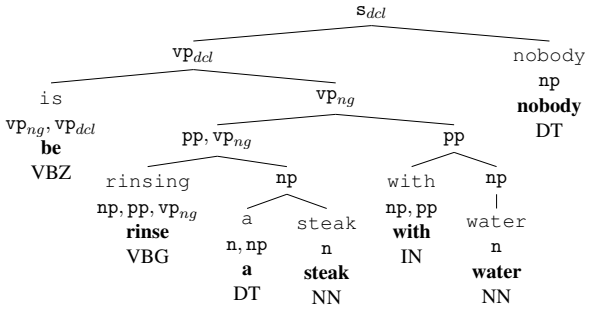

Figure 6: The CCG term obtained from the CCG tree of Figure 5. NB: the lexical rule remains.

the first step: CCG categories are converted into types ($Y\backslash X$ and $Y/X \rightsquigarrow (\mathtt{x}, \mathtt{y})$), and argument constituents are placed after function ones in binary combinatory rules. Resulted structures are called *CCG terms* (Figure 6).

Obtained CCG terms are often semantically inadequate. One of the reasons for this is *lexical* (i.e. *type-changing*) rules (e.g., $N \mapsto NP$ in Figure 5) of the CCG parsers which still remain in CCG terms (e.g., $[\mathbf{water}_{\mathtt{n}}]_{\mathtt{np}}$ in Figure 6). These rules are destructive from a compositional point of view. We designed 13 schematic rewriting rules of general type that *correct* CCG terms—make them semantically more adequate and transparent. The rules make use of types, part-of-speech (POS) and named entity (NE) tags to match semantically inadequate analyses:[5]

- Certain non-compositional multiword expressions are treated as constant terms: *a lot of*, *in front of*, *a few*, *because of*, *next to*, etc.
- Type-changing rules are *explained* by changing lexical types, decomposing terms or inserting new terms. This step carries out conversions like $[\mathbf{europe}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \mathbf{europe}_{\mathtt{np}}$, $[\mathbf{nobody}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \mathbf{no}_{\mathtt{n,np}}\mathbf{person}_{\mathtt{n}}$, and $[\mathbf{water}_{\mathtt{n}}]_{\mathtt{np}} \rightsquigarrow \mathbf{a}_{\mathtt{n,np}}\mathbf{water}_{\mathtt{n}}$ (see Figure 7). Inserted $\mathbf{a}_{\mathtt{n,np}}$ merely plays a role of an existential quantifier.
- Several CCG analyses are altered in order to reflect formal semantics, e.g., attributive modifiers are pushed *under* a relative clause: $\mathbf{big}\,(\mathbf{which\ run\ mouse}) \rightsquigarrow \mathbf{which\ run}\,(\mathbf{big\ mouse})$; and PPs are attached to nouns rather than NPs: $\mathbf{in}\,(\mathbf{a\ box})(\mathbf{every\ pug}) \rightsquigarrow \mathbf{every}\,(\mathbf{in}\,(\mathbf{a\ box})\,\mathbf{pug})$.

[5] To handcraft the rules, we used a development set of 1.7K CCG derivations obtained by parsing the sentences from FraCaS (Cooper et al., 1996) and the trial portion of SICK (Marelli et al., 2014) with CCG-based parsers: C&C (Clark and Curran, 2007) and EasyCCG (Lewis and Steedman, 2014).

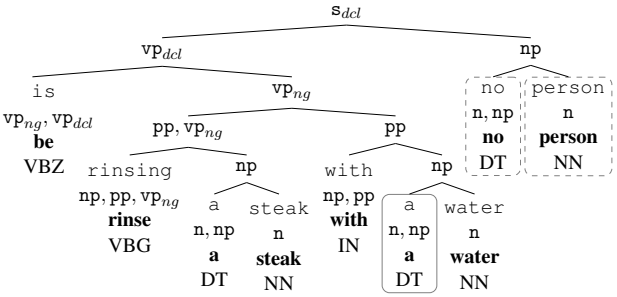

Figure 7: The corrected version of the CCG term of Figure 6, with inserted and decomposed terms.

LLFs are obtained from corrected CCG terms by type-raising QNPs from $\mathtt{np}$ to the type $(\mathtt{vp}, \mathtt{s})$ of generalized quantifiers. Hence, several LLFs are produced from a single CCG tree due to quantifier scope ambiguity, e.g., (3–5) are some of the LLFs obtained from the CCG term of Figure 7.[6]

$$N\big(\mathbf{be}\big(\lambda z.\, S\big(\lambda x.\, W\big(\lambda y.\, \mathbf{rinse}\, x\, (\mathbf{with}\, y) z\big)\big)\big)\big) \quad (3)$$

$$N\big(\mathbf{be}\big(\lambda z.\, W\big(\lambda y.\, S\big(\lambda x.\, \mathbf{rinse}\, x\, (\mathbf{with}\, y) z\big)\big)\big)\big) \quad (4)$$

$$W\big(\lambda y.\, S\big(\lambda x.\, N(\mathbf{be}\,\big(\mathbf{rinse}\ x\ (\mathbf{with}\ y)\big)\big)\big)\big) \quad (5)$$

Since LLFs encode instructions for semantic composition, they can be used to compositionally derive semantics in other meaning representations (Figure 4), e.g., first-order logic (FOL) or Discourse Representation Theory (DRT). For this application, LLFgen can be used as an independent tool. Given a CCG derivation in the Prolog format (supported by both C&C and EasyCCG), LLFgen can return LLFs in XML, HTML or LaTeXformats. For a CCG tree, it is also possible to get either only the first LLF, e.g., (3), often reflecting the natural order of quantifiers, or a list of LLFs with various quantifier scope orders (possibly including semantically equivalent LLFs, like (3) and (4)).

[6] The LLFs use the following abbreviations: $S = \mathbf{a}_{\mathtt{q}}\,\mathbf{steak}_{\mathtt{n}}$, $W = \mathbf{a}_{\mathtt{q}}\,\mathbf{water}_{\mathtt{n}}$, and $N = \mathbf{no}_{\mathtt{q}}\,\mathbf{person}_{\mathtt{n}}$, where $\mathtt{q} = (\mathtt{n}, \mathtt{vp}, \mathtt{s})$.

4 Natural Logic Theorem Prover

The tableau theorem prover for natural logic (NLogPro) represents a core part of LangPro (Figure 1). It is responsible for checking a set of linguistic expressions on (in)consistency. NLogPro consists of four components: the *Proof Engine* builds tableau proofs by applying the rules from the *inventory of Rules*; the rule applications are validated by the properties of lexical terms (encoded in the *Signature*) and the lexical knowledge (available from the *Knowledge Base*). We used the same development data for LLFgen and NLogPro.

4.1 Signature

The signature (SG) lists lexical terms that have algebraic properties relevant for inference, e.g., monotonicity, intersectivity, and implicativity. The lexical items in the SG come with an argument structure where each argument position is associated with a set of algebraic properties. For example, **every** is characterized in the SG as `[dw,up]`, meaning that in its first argument **every** is downward monotone while being upward monotone in the second one. Currently, the SG lists about 20 lexical items, mostly generalized quantifiers (GQs), that were found in the development data.

4.2 The Inventory of Rules

The inventory of rules (IR) contains all inference rules used by the prover. Currently there are ca. 80 rules in the IR (some in Figure 3). Around a quarter of the rules are from Muskens (2010) and the rest are manually collected while exploring the development data. The rules cover a plethora of phenomena. Some of them are of a formal nature like Boolean connectives and monotonicity and others of linguistic nature: adjectives, prepositions, definite NPs, expletives, open compound nouns, light verbs, copula, passives and attitude verbs.

The IR involves around 25 *derivable* rules—the rules that represent shortcuts of several rule applications. One such rule is ($\text{NO}^n_{\mathbb{T}}$) in Figure 3, which is a specific version of ($\text{NO}_{\mathbb{T}}$). Use of derivable rules yields shorter tableau proofs but raises a problem of performing the same rule application several times. NLogPro avoids this by maintaining a subsumption relation between the rules and keeping track of rule applications per branch.

A user can introduce new rules in the IR as Prolog rules (Code 1): the head of the rule encodes antecedent nodes ===> consequent nodes, and the body is a list of Prolog goals specifying the conditions the rule has to meet.

```
r(Name, Feats, ConstIndx, KeyWrd, KB,
 br([nd(Mod1, LLF1, Arg1, Sign1),...
     nd(ModN, LLFN, ArgN, SignN)],
   Signature) ===>
 [br([nd(Mod3, LLF3, Arg3, Sign3),...],
    Signature3),
  br([nd(Mod4, LLF4, Arg4, Sign4),...],
    Signature4)]
:- Goal1, ..., GoalN. %conditions
```

Code 1: The Prolog format of tableau rules. `Feats` denotes efficiency features, `ConstIndx` and `KB` are the KB and indexing of constants respectively (fixed for every rule), and `KeyWrd` denotes fixed lexical terms occurring in the rule. Each branch maintains its own signature of entities introduced during the proof.

4.3 Knowledge Base

The knowledge base (BS) is based on the Prolog version of WordNet 3.0 (Fellbaum, 1998). At this moment only the hyponymy/hypernymy, similarity and antonymy relations are included in the KB. For simplicity, LangPro does not do any word sense disambiguation (WSD) but allows multiple word senses for a lexical term. For example, $A \sqsubseteq B$ iff $SynSet_A$ is a hyponym of $SynSet_B$, or there are similar $Sense_A$ and $Sense_B$, where $Sense_A \in SynSet_A$ and $Sense_B \in SynSet_B$. In the prover, a user can restrict the number of word senses per word by specifying a cutoff N, i.e. the N most frequent senses per word.

In addition to the WN relations, a user can introduce new lexical relations in the KB as Prolog facts, e.g., `is_(crowd, group).`

4.4 The Proof Engine

The proof engine (PE) is the component that builds proof trees. While applying rules it takes into account computational efficiency of each rule where the efficiency depends on the following categories:

- *Branching*: a rule is either branching (e.g., $\forall_{\mathbb{T}}$) or non-branching (e.g., AUX).
- *Semantic equivalence*: this depends whether the antecedents of a rule is semantically equivalent to its consequents. For example, ($\wedge_F$) encodes the semantic equivalence while ($\text{NO}_{\mathbb{T}}$) does not.
- *Producing*: depending on whether a rule produces a fresh entity, it is a producer or a non-producer. ($\exists_{\mathbb{T}}$) is a producer while ($\forall_{\mathbb{T}}$) is not.

- *Consuming*: a rule is a consumer iff it employs an old entity from the branch during application. The consumer rules are ($\forall_{\mathbb{T}}$) and ($\text{NO}_{\mathbb{T}}$) but ($\exists_{\mathbb{T}}$).

The most efficient combination of these features is non-branching, semantic equivalence, non-producing and non-consuming. Depending on a priority order between these categories, called an efficiency criterion, one can define a partial efficiency order over the rules. In particular, (6) is one of the best efficiency criteria on SICK (Abzianidze, 2016a, Ch. 6). According to (6), ($\wedge_{\mathbb{F}}$) is more efficient than ($\text{NO}^n_{\mathbb{T}}$) since the equivalence is the most prominent category in (6), and ($\wedge_{\mathbb{F}}$) is equivalence in contrast to ($\text{NO}^n_{\mathbb{T}}$)

$$[\texttt{equi}, \texttt{nonBr}, \texttt{nonProd}, \texttt{nonCons}] \qquad (6)$$

A user can change the default criterion (6) by passing a criterion via the Prolog predicate `effCr/1`.

The PE builds two structures: a tree (see Figure 2) and a list. The latter represents a list of the tree branches. The list structure is the main data structure that guides the computation process while the tree structure is optional (activated with the predicate `prooftree/0`) and is used for displaying proofs in a compact way. A few of the predicates that control the proof procedure are:

- `ral/1` sets a rule application limit to n, which means that after n rules are applied the proof is terminated. $n = 400$ by default.
- `thE/0` always permits existential import from definite NPs: it makes ($\exists_{\mathbb{T}}$) applicable to the entry $\textbf{the}_{\text{n,vp,s}}\ \textbf{dog}_{\text{n}}\ \textbf{bark}_{\text{vp}} : \mathbb{F}$.
- `allInt/0` allows to treat lexical modifiers of the form $c^{\texttt{VB.|JJ|NN}}_{\text{n,n}}$ as intersective by default unless stated differently in the SG. This permits to infer $\textbf{baby}_{\text{n}} : c : \mathbb{T}$ and $\textbf{kangaroo}_{\text{n}} : c : \mathbb{T}$ from $\textbf{baby}_{\text{n,n}}\textbf{kangaroo} : c : \mathbb{T}$, for better or worse.
- `the/0`, `a2the/0`, and `s2the/0` are used as flags and treat bare, indefinite, and plural NPs as definite NPs, respectively.

5 LangPro: Natural Language Prover

The tableau-based theorem prover for natural language is obtained by chaining a CCG parser, LLFgen and NLogPro. In order to detect a semantic relation between a set of premises $\{p_i\}^n_{i=1}$ and a hypothesis h, first the corresponding LLFs $\{P_i\}^n_{i=1}$ and H are obtained via a CCG parser and LLFgen (i.e. for simplicity, a single LLF per sentence). Then based on the lexical terms of the LLFs, relevant sets of relations K and rules R are collected from the KB and the IR, respectively. To refute both entailment and contradiction relations NLogPro builds two proof trees using K and R. One starts with the counterexample (7) for entailment and another with the counterexample (8) for contradiction. The semantic relation which could not be refuted (i.e. its tableau for the counterexample was closed) is said to be proved. The relation is considered to be neutral iff both tableaux have the same closure status: open or closed.

$$\{P_1 : \mathbb{T},\ \ldots,\ P_n : \mathbb{T},\ \ H : \mathbb{F}\} \qquad (7)$$

$$\{P_1 : \mathbb{T},\ \ldots,\ P_n : \mathbb{T},\ \ H : \mathbb{T}\} \qquad (8)$$

Entailment relations often do not depend on semantics of phrases shared by premises and hypotheses. To bypass analyzing the common phrases, LangPro can use an optional CCG term aligner in LLFgen (Figure 1), which identifies the common CCG sub-terms and treats them as constants. The sub-terms that are downward monotone or indefinite NPs are excluded from alignments as they do not behave semantically as constants. After aligning CCG terms, aligned LLFs are obtained from them via the type-raising. Tableau proofs with aligned LLFs are shorter. Thus, first, a tableau with aligned LLFs is built, and if the tableau did not close, then non-aligned LLFs are used since alignment might prevent the tableau from closing. On SICK, the aligner boosts the accuracy by 1%. If stronger alignment is used (i.e. aligning indefinite NPs), the accuracy on SICK is increased by 2%. Both weak and strong alignment options can be chosen in LangPro.

The parser component of LangPro can be filled by C&C or EasyCCG. This results in two versions of LangPro, ccLangPro and easyLangPro respectively. Both versions achieve similar results on FraCaS and SICK, and a simple aggregation of their judgments (coLangPro) improves the accuracy on the unseen portion of SICK by 1%.

With respect to its rule-based nature, LangPro is fast. Given ready CCG derivations, on average 100 SICK problems are classified in 3.5 seconds.[7] Details about speed and impact of parameters on the performance are given in Abzianidze (2016a).

In addition to an entailment judgment, LangPro can output the actual tableau proof trees (similar

[7] This is measured on 8×2.4 GHz CPU machine, when proving problems in parallel (via the `paralle/0` predicate) with the strong aligner option and the rule application limit 50—the configuration that achieves high performance both in terms of speed and accuracy.

to Figure 2) in three formats: a drawing of a proof tree via the XPCE GUI, a LaTeX source code, an XML output, or an HTML file.

6 Related work

Theorem proving techniques (Bos and Markert, 2005) or ideas from Natural Logic (MacCartney, 2009) were already used in recognizing textual entailment (RTE). But the combination of these two is a novel approach to RTE. The underlying higher-order logic of LangPro guarantees sound reasoning over several premises, including some complex semantic phenomena. This is in contrast to the RTE systems that cannot reason over several premises or cannot account for Booleans and quantifiers, including the ones (MacCartney, 2009) inspired by Natural Logic, and in contrast to those ones that use FOL representations and cannot cover higher-order phenomena like generalized quantifiers or subsective adjectives.

LangPro achieves state-of-the-art semantic competence (with accuracy of 87%) on the FraCaS sections commonly used for evaluation (Abzianidze, 2016b,a). On SICK, the prover obtains 82.1% of accuracy (Abzianidze, 2015a, 2016a) while state-of-the-art systems score in the range of 81-87% and average performance of human on the dataset is around 84%. Detailed comparison of LangPro to the related RTE systems is discussed in (Abzianidze, 2015a, 2016b,a).

7 Conclusion

The presented natural language prover involves a unique combination of natural logic, higher-order logic and a tableau method. Its natural logic side simplifies generation of the logical forms and makes the prover to be relatively easily scaled up. Due to its higher-order virtue, the prover easily accounts for complex semantic phenomena untameable in FOL. Because of its high reliability (less than 3% of its entailment and contradiction judgments are incorrect), the judgments of the prover can be successfully borrowed by other RTE systems. Further scaling-up for longer sentences (e.g., newswire text) and automated knowledge acquisition present future challenges to the prover.

Acknowledgments

This work has been supported by the NWO-VICI grant "Lost in Translation – Found in Meaning" (288-89-003).

References

Lasha Abzianidze. 2015a. A tableau prover for natural logic and language. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2492–2502. ACL.

Lasha Abzianidze. 2015b. Towards a wide-coverage tableau method for natural logic. In Tsuyoshi Murata, Koji Mineshima, and Daisuke Bekki, editors, *New Frontiers in Artificial Intelligence: Revised Selected Papers of JSAI-isAI 2014 Workshops, LENLS, JURISIN, and GABA*, pages 66–82. Springer.

Lasha Abzianidze. 2016a. *A natural proof system for natural language*. Ph.D. thesis, Tilburg University.

Lasha Abzianidze. 2016b. Natural solution to fracas entailment problems. In *Proceedings of the Fifth Joint Conference on Lexical and Computational Semantics*, pages 64–74. ACL.

Johan Bos and Katja Markert. 2005. Recognising textual entailment with logical inference. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 628–635.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33.

Robin Cooper, Dick Crouch, Jan Van Eijck, Chris Fox, Josef Van Genabith, Jan Jaspars, Hans Kamp, David Milward, Manfred Pinkal, Massimo Poesio, Steve Pulman, Ted Briscoe, Holger Maier, and Karsten Konrad. 1996. *FraCaS: A Framework for Computational Semantics*. Deliverable D16.

Christiane Fellbaum, editor. 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

Mike Lewis and Mark Steedman. 2014. A* CCG parsing with a supertag-factored model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 990–1000. ACL.

Bill MacCartney. 2009. *Natural language inference*. Phd thesis, Stanford University.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A sick cure for the evaluation of compositional distributional semantic models. In *Proceedings of LREC'14*. ELRA.

Reinhard Muskens. 2010. An analytic tableau system for natural logic. In Maria Aloni, Harald Bastiaanse, Tikitu de Jager, and Katrin Schulz, editors, *Logic, Language and Meaning*, volume 6042 of *LNCS*, pages 104–113. Springer.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press, Cambridge, MA, USA.