CatLog: A Categorial Parser/Theorem-Prover*

Glyn Morrill
Universitat Politècnica de Catalunya
morrill@lsi.upc.edu.edu
http://www.lsi.upc.edu/~morrill

Abstract. We present CatLog, a parser/theorem prover for logical categorial grammar. The logical fragment implemented is a displacement logic the multiplicative basis of which is the displacement calculus of Morrill, Valentín & Fadda (2011)[8].

(Logical) categorial grammar (Morrill 1994[9], 2011[10]; Moortgat 1997[6]; Carpenter 1998[1]; Jäger 2006[4]) originated with Lambek’s (1958[3]) insight that a calculus of grammatical types (constituting a residuated monoid) can be formulated using Gentzen’s method. The result is an algebraic rendering of grammar as logic and parsing as deduction. Although the design is, really, architecturally perfect and, by now, well-understood, linguistically it is strictly limited to continuity by the fact that it deals with a residuated family with parent (the canonical extension of) concatenation: after all, the whole challenge of modern linguistics for 50 years has been the ubiquity in natural grammar of discontinuity. In this relation Morrill, Valentín & Fadda (2011)[8] provides for discontinuity the displacement calculus D, deductively a conservative extension of the Lambek calculus L with residuated families with respect to both concatenation and intercalation. Like L, D is free of structural rules and enjoys Cut-elimination and its corollaries the subformula property, decidability, and the finite reading property.

CatLog is a categorial parser/theorem prover implementing a categorial logic extending D. It employs Cut-free backward chaining sequent theorem proving. For L deductive spurious ambiguity can be removed by normalization (Hendriks 1993[3]). Because D is based on the same design principles, the same techniques can be adopted (Morrill 2011[7]) and CatLog depends on this. In addition to normalization CatLog uses sequent search space pruning by the count invariance of van Benthem (1991[11]). The type-constructors of the displacement logic of CatLog are shown in Fig. 1.

Version 1.2 of CatLog is provisional in a number of respects. In particular, not all spurious ambiguity is eliminated for the categorial logic fragment, and non-duplication of results is achieved by filtering according to a brute force duplication check. Furthermore, bracketing structure must be specified in the input, rather than be induced. And the count-invariance check for multiplicatives

* This research was partially supported by BASMATI MICINN project (TIN2011-27479-C04-03) and by SGR2009-1428 (LARCA).
is not adapted to additives and structural modalities. These issues remain topics for future improvement. Nevertheless CatLog 1.2 already provides fast and wide-coverage Montague-like parsing.

The program comprises 3000 lines of Prolog implementing some 80 inference rules for the categorial logic fragment, \(\text{\LaTeX} \) outputting, lexicon, and sample sentences. Among the examples four blocks are distinguished: Dutch examples (cross-serial dependencies), relativization including islands and parasitic gaps, the Montague example sentences of Dowty, Wall and Peters (1981) [2] Chapter 7, and the example sentences of Morrill, Valentín and Fudda (2011) [8].

The functionality is as follows. Once CatLog has been loaded into Prolog, the query \(-\text{pplex.} \) will cause the lexicon to be pretty printed in the console window, the Dutch part of which is as shown in Fig. 2. The query \(-\text{pplexlatex.} \) has no visible effect but will cause the lexicon to be output in \(\text{\LaTeX} \) to a file named \("s.tex"\). Querying \(t(N) \) will test the examples unifying with term \(N \). For example \(-\text{t(rel(6)).} \) tests the relativization example 6, \(-\text{t(rel(_)).} \) tests all the relativization examples, and \(-\text{t(_).} \) tests all the examples. The analyses — the examples, the derivational proofs, and the semantic readings — appear in the Prolog window, and this information but without duplicate equivalent analyses is written in \(\text{\LaTeX} \) to a file named \("t.tex"\). \(\text{\LaTeX} \)ing the file \("out.tex"\) will include s.tex and t.tex and format the lexicon and last analyses made. For example, \(-\text{t(d(2)).} \) produces the contents in Fig. 3 in Prolog. The \(\text{\LaTeX} \) output for the Dutch part of the lexicon and the same example is as shown in Figs. 4 and 5.

References

Fig. 2. Dutch part of lexicon

(d(2)) janboeken-kunstlessen S_647

Fig. 3. Dutch verb raising

wil: (NA\SsI)(NA\Sf) : LABAC((want (B C)) C)
will: Q((SA(NA\SsI))(NA\Sf))) : LABAC((want (C D)) D)
alle: (SA(NA\SsI))(NA\Sf) : LABAC((thing C) → (B C))
boeken: Np(n) : books
cellica: Nt(s(f)) :
de: Nt(s(A))/CNA: the
helpen: D1((NA\SsI))(NB\(NA\Sf)) : λCAD((help (C D)) D)
henk: Nt(s(m)) : h
jan: Nt(s(a))/CNA: the
kunnen: D1((NA\SsI))(NB\(NA\Sf)) : λCAD((able (B C)) C)
las: Nt(s(B))/Sf : read
lezen: D1((NA\Sf))(NB\(NA\Sf)) : read
mijnparen: CNp(m) : hippo
voeren: D1((NA\Sf))(NB\(NA\Sf)) : feed
zag: (Nt(s(A))/Sf)(NB\(Nt(s(A))/Sf)) : λCAD((see (C D)) D)

Fig. 4. Dutch part of lexicon
Fig. 5. Dutch verb raising