A Labelled Analytic Theorem Proving Environment for Categorial Grammar

Saturnino F. Luz-Filho[†]
Patrick Sturt[‡]
Centre for Cognitive Science
2 Buccleuch Place, Edinburgh EH8 9LW, Scotland, UK
{luz,sturt}@cogsci.ed.ac.uk

Abstract

We present a system for the investigation of computational properties of categorial grammar parsing based on a labelled analytic tableaux theorem prover. This proof method allows us to take a modular approach, in which the basic grammar can be kept constant, while a range of categorial calculi can be captured by assigning different properties to the labelling algebra. The theorem proving strategy is particularly well suited to the treatment of categorial grammar, because it allows us to distribute the computational cost between the algorithm which deals with the grammatical types and the algebraic checker which constrains the derivation.

1 Background

A current trend in logic is to attempt to incorporate semantic information into the domain of deduction, [11], [5]. An area for which this strategy is particularly useful is the problem of categorial grammar parsing. The categorial grammar research programme requires the use of a range of logical calculi for linguistic description. Some researchers have considered labelled deduction as a tool for implementing categorial parsers [17], [19], and this paper can be seen as a new contribution to this field.

In this paper we aim for a modular approach, in which the basic grammar is kept constant, while different calculi can be implemented and experimented with by constraining the derivations produced by the theorem prover. At present, our system covers the classical Lambek Calculus, L, as well as the non-associative Lambek calculus NL, [13], and variants such as Van Benthem's [6] LP, LPC, LPE and LPCE, and their non-associative counterparts. The system is based on labelled analytic deduction, particularly on the LKE method, developed by D'Agostino and Gabbay [7]. LKE is similar to a Smullyan-style tableau system, in which the derivations obey the sub-formula principle, but it improves on efficiency by restricting the number of branching rules to just one. Different categorial logics are handled by assigning different properties to the labelling algebra, while the basic syntactic apparatus remains the same. This allows the user to experiment with various linguistic properties without having in principle to modify the grammar itself.

The basic structure of the paper is as follows. In section 1.1, we introduce the family of categorial calculi, and discuss some of the linguistic arguments which have been put forward in the literature with regard to these calculi. In section 2, we introduce the logical apparatus on which the system is based, describe the algorithm and prove some of the properties mentioned in section 1.1 within this framework. We also show how different grammars can be characterised and present a worked example. In section 2.3 the system is compared with other strategies for dealing with multiple categorial logics, such as hybrid formalisms and unification-based Gentzen-style deduction. In this section, we also suggest some ways to improve the efficiency of the system, and strategies for dealing with the complexity of labelled unification.

1.1 A Family of Categorial Calculi and Their Linguistic Applications

Categorial Grammars can be formalised in terms of a hierarchy of well understood and mathematically transparent logics, which yield as theorems a range of combinatorial operations. However

[†]Supported by CNPq Brazilian Research Council Research Studentship No. 200210/93-9
[‡]Supported by ESRC Research Studentship No. R00429334338

the precise nature of the combinatorial power required for an adequate characterisation of natural language is still very much a matter of debate. For this reason, it is desirable to have a means of systematically testing the linguistic consequences of adopting various calculi. In this section we give an overview of the linguistic applications of some of the calculi in the hierarchy, with a view towards motivating the usefulness of a generic categorial theorem prover as a tool for linguistic study.

The combinatorial possibilities of expressions in general can be characterised in terms of *reduction laws*. In *R1-R6* below, we give some reduction laws discussed in [16], which have been found to be linguistically useful.[1].

R1:	*Application*	*R2:*	*Composition*	*R3:*	*Associativity*
	X/Y, Y ⊢ X		X/Y, Y/Z ⊢ X/Z		(Z\X)/Y ⊢ Z\(X/Y)
	Y, Y\X ⊢ X		Z\Y, Y\X ⊢ Z\X		Z\(X/Y) ⊢ (Z\X)/Y
R4:	*Lifting*	*R5:*	*Division (main functor)*	*R6:*	*Division (sub. functor)*
	X ⊢ Y/(X\Y)		X/Y ⊢ (X/Z)/(Y/Z)		X/Y ⊢ (Z/X)\(Z/Y)
	X ⊢ (Y/X)\Y		Y\X ⊢ (Z\Y)\(Z\X)		Y\X ⊢ (Y\Z)/(X\Z)

It is possible to define a hierarchy of logical calculi, each of which admits one or more of *R1-R6* as theorems; from the purely applicative calculus AB, of Ajdukiewicz and Bar-Hillel, [2], which supports only *R1*, to the full Lambek calculus L, which supports all the above laws. Calculi intermediate in power between AB and L have been explored (e.g. Dependency Categorial Grammar [20]), as well as stronger calculi which extend the power of L through the addition of structural rules.

Much of the interest in using categorial grammars for linguistic research derives from the possibilities they offer for characterizing a flexible notion of constituency. This has been found particularly useful in the development of theories of coordination, and incremental interpretation. For example, assuming standard lexical type assignments, the following right node raised sentence cannot be derived in AB, but does receive a derivation in a system which includes *R3*, with each conjunct assigned the type indicated.

(1) [John resents $_{S/NP}$] and [Peter envies $_{S/NP}$] Mary

A calculus which includes composition, *R2*, will allow a function to apply to an unsaturated argument, and it is this property which allows Ades and Steedman [1] to treat long distance dependencies, and motivates much of Steedman's later work on incremental interpretation.

Dowty [9] uses the combination of composition, *R2* and lifting, *R4*, to derive examples of non-constituent coordination such as *John gave mary a book and Susan a record.*

We can increase the power of L by adding the structural transformations *Permutation*, *Contraction* and *Expansion*, to derive the calculi LP, LPC, LPE and LPCE. The structural transformation *Permutation*, which removes the restrictions on the linear order of types, allows us to go beyond the purely concatenative derivations of L. This allows us to deal with sentences exhibiting non-standard constituent order. For example, Moortgat suggests using permutation for dealing with *heavy NP-shift* in examples similar to the following [16]:

(2) John gave [to his nephew $_{PP}$] [all the old comic books which he'd collected in his troubled adolescence $_{NP}$].

In (2), the bracketed constituents can be "rearranged" via permutation so that a derivation is possible that employs the standard type ((NP\ S)/PP)/NP for the ditransitive verb *gave.*

In L, while it is possible to specify a type missing an argument on its left or right periphery, it is not possible to specify a type missing an argument "somewhere in the middle", making it impossible to deal with non-peripheral extraction. However, as Morrill et al show, permutation provides the additional power necessary to account for this phenomenon [12].

[1] We adopt the Lambek notation, in which X/Y is a function which "takes" a Y to its right to yield an X, and Y\X is a function which "takes" a Y to its left to yield an X.

In addition to permutation, there are also linguistic examples which motivate contraction (e.g. gapping, [16]) and expansion (e.g. right dislocation, [16]). However it is universally recognized that a system employing the unrestricted use of structural transformations would be far too powerful for any useful linguistic application, since it would allow arbitrary word order variation, copying and deletion. For this reason, a goal of current research is to build a system in which the resource freedom of the more powerful calculi can be exploited when required, while the basic resource sensitivity of L is retained in the general case. One such approach is to employ structural modalities [12], which are operators that explicitly mark those types which are permitted to be manipulated by specific structural transformations.[2]

2 A framework for Categorial Deduction

In this section we describe the theorem proving framework for categorial deduction. We start by setting up basic ideas of categorial logic, giving formal definitions of the core logical language. Then we move on to the theorem proving strategy, introducing the LKE approach [7] and the algebraic apparatus used to characterise different calculi.

2.1 The core syntax

We assume that there is a finite set of atomic grammatical categories which will be represented by special symbols: NP for noun phrases, S for sentences, etc. So, the set of well-formed categories can be defined as below.

Definition 1 *The set of well-formed categories, C is the smallest set which contains every basic category and which is closed under the following rule:*

(i) If $X \in C$ and $Y \in C$, then X/Y, $X\backslash Y$ and $X \bullet Y \in C$

Our purpose in this section is to define a prcedure which will enable us to verify, given an entailment relation $\vdash$, whether or not such a relation holds for the logic being considered.

Many proof procedures for classical logic have been proposed: natural deduction, Gentzen's sequents, analytic (Smullyan style) tableaux, etc. Among these, methods which conform to the sub-formula principle are particularly interesting, as far as automation is concerned. See [10] for a survey. Most of these methods, along with proof methods developed for resource logics, such as Girard's proof nets (a variant of Bibel's connection method), can be used for categorial logic. Leslie [14] presents and compares some categorial versions of these procedures for the standard Lambek calculus L, taking into account complexity and proof presentation issues. Although tableau systems are not discussed in [14], a close relative, the cut-free sequent calculus is presented as being the one which represents the best compromise between implementability and display of the proof.

Smullyan style tableau systems, however, have been shown to be inherently inefficient [8]. They cannot even simulate truth-tables in polynomial time. The main reason for this is the fact that many of the Smullyan tableau expansion rules cause the proof tree to branch, thus increasing the complexity of the search. Moreover, keeping track of the structure of the derivations represents an extra source of complexity, which in most categorial parsers [17, 19] is reflected in expensive unification algorithms employed for dealing with substructural implication. In order to cope with efficiency and generality, we have chosen the LKE system [7] as the proof theoretic basis of our approach[3]. LKE is an analytic (its derivations exhibit the sub-formula property) method of proof by refutation[4] which has only one branching rule. In addition, its formulae are labelled according to a labelling algebra which will determine the closure conditions for the proof trees[5]. In what follows, we shall concentrate on explaining our version of the system, the heuristics that

[2]Systems which allow the selective use of structural transformations may be implemented in the general framework presented here, although we do not address this issue.

[3]For standard propositional logic, it has been shown that LKE can simulate standard tableau in polynomial time, but the converse is not true.

[4]A formula is proved by building a counter-model for its negation.

[5]See section 2.3 for a discussion of how LKE, unlike proof nets or standard tableaux, enables us to reduce the computational cost of label unification.

we have found useful for dealing with particularities of the calculi covered, and the relevant results for these calculi. The usual completeness and soundness results (with respect to the algebraic semantic provided) are already given in [7], so we will not discuss them here.

We have mentioned that the condition for a branch to be considered closed in a standard tableau is that both a formula and its negation occur on it. The calculus defined above presents no negation, though. So, we have to appeal to some extrinsic mechanism to express contradiction. In Smullyan's original formulation, the formulae occurring in a derivation were all preceded by *signs*: T or F. For instance, assume that we want to prove A $\Rightarrow$ A in classical logic. We start by saying that the formula is false, prefixing it by F, and try to find a refutation for F A $\Rightarrow$ A. For this to be the case both T A (the antecedent) and F A (the consequent) have to be the case, yielding a contradiction. In classical logic we can interpret T and F as assertion and denial respectively, and so we can incorporate F into the language as negation, obtaining uniform notation by eliminating the need for signed formulae. In our approach, since negation is not defined in the language, we shall make use of signed formulae as proof theoretic devices. T and F will be used to indicate whether or not a certain string available for combination to produce a new one.

2.2 The generalised parsing strategy

If we had restricted the system to dealing with signed formulae, we would have a proof procedure for an implicational fragment of standard propositional logic enriched with backwards implication and conjunction. However, we have seen that the Lambek calculus does not exhibit any of the structural properties of standard logic, and that different calculi may be obtained by varying structural transformations. Therefore, we need a mechanism for keeping track of the structure of our proofs. This mechanism is provided by labelling each formula in the derivation with *information tokens*[6].

Labels will act not only as mechanisms for encoding the structure of the proof, from a proof-theoretic perspective, but will also serve as means to propagate semantic information through the derivation. A label can be seen as an information token supporting the information conveyed by the signalled formula it labels. Tokens may convey different degrees of informativeness, so we shall assume that they are ordered by an anti-symmetric, reflexive and transitive relation, $\sqsubseteq$, so that an expression like x $\sqsubseteq$ y asserts that y is at least as informative as x (i.e. verifies at least as many sentences as x). We also assume that this semantic relation, "verifies", is closed under deductibility.

It is natural to suppose that, as well as categories, information tokens can be composed. We have seen that a type S/NP can combine with a type NP to produce an S. If we assume that there are tokens x and y verifying respectively S/NP and NP, how would we represent the token that verifies S? Firstly, we define a token composition operation $\circ$. Then, we assume that, a priori, the order in which the categories appear in the string matters. So, a minimal information token verifying S would be x $\circ$ y. As we shall see below, the constraints we impose on $\circ$ will ultimately determine which inferences will be valid. For instance, if we assume that the order in which the types occur is not relevant, then we may allow permutation on the operands, so that x $\circ$ y $\sqsubseteq$ y $\circ$ x; if we assume that contraction is a structural property of the calculus then the string [S/NP, NP, NP] will also yield an S, since y $\circ$ y $\sqsubseteq$ y, etc. Let's formalise these notions by defining an algebraic structure, called *Information frame.*

Definition 2 *An Information Frame is a structures* $\mathcal{L} = \langle \mathcal{P}, \circ, 1, \sqsubseteq \rangle$, *where (i)* $\mathcal{P}$ *is a non-empty set of information tokens; (ii)* $\mathcal{P}$ *is a complete lattice under* $\sqsubseteq$ *; (iii)* $\circ$ *is an order-preserving, binary operation on* $\mathcal{P}$ *which satisfies continuity, i.e., for* every *directed family* $\{z_i\}$, $\bigsqcup \{z_i \circ x\} = \bigsqcup \{z_i\} \circ x$ and $\bigsqcup \{x \circ z_i\} = x \circ \bigsqcup\{z_i\}$; *and (iv)* 1 *is an identity element in* $\mathcal{P}$.

Combinations of types are accounted for in the labelling algebra by the composition operator. Now, we need to define an algebraic counterpart for syntactic composition, • , itself. When a formula like S/NP • NP is verified by a token x, this is because its components were available for combination, and consequently were verified by some other tokens. Now, suppose S/NP was verified by a token, say a. What would be the appropriate token for NP, such that S/NP

[6]See [11] for a proof theoretic motivated, LDS approach, and [5] for an approach based on a finer-grained, semantically motivated information structure.

combined with NP would be verified by x? It certainly would not be more informative than x. Moreover, if the expression S/NP • NP were to stand for the composition of the (informational) meanings of its components, then the label for NP would have to verify, when combined with a, at most as much information as x. In order to express this, we define the label for NP as being *the greatest y s.t. x is at least as informative as a combined with y.* This token will be represented by $x \swarrow a$. In general, $x \swarrow y \stackrel{def}{=} \bigsqcup \{z |\ y \circ z \sqsubseteq x\}$. An analogous operation, $\nearrow$, can be defined to cope with cases in which it is necessary to find the appropriate label for the first operand by reversing the order of the tokens in the definition above. Some properties of $\swarrow$[7]:

$$y \circ (x \swarrow y) \sqsubseteq x \quad (3) \qquad 1 \sqsubseteq x \swarrow x \quad (4)$$

$$(x \swarrow y) \circ z \sqsubseteq (x \circ z) \swarrow y \quad (5) \qquad (x \swarrow y) \swarrow z \sqsubseteq x \swarrow (y \circ z) \quad (6)$$

Having set the basic elements of our proof-theoretic apparatus, we are now able to define the components of a derivation as follows:

Definition 3 *Signed labelled formulae (SLF) are expressions of the form S Cat : L, where* $S \in \{T.F\}$. $Cat \in \mathcal{C}$ *and* $L \in \mathcal{L}$

A derivation, or proof will be a tree structure built according to certain syntactic rules. These rules will be called *expansion* rules, since their application will invariably expand the tree structure. There are three sorts of expansion rules: those which expand the tree by generating two formulae from a single one occurring previously in the derivation, those which expand the tree by combining two formulae into a third one which is then added to the tree, and the branching rule. The first kind of rule corresponds to what is called α-rule in Smullyan tableaux; these rules will be called α-rules here as well. The second and third kinds have no equivalents in standard tableau systems. We shall refer to the second kind as σ-rules, and to the branching rule as β-rule – after Smullyan's, even though his branching rules are different. *Figure 1* summarises the expansion rules to be employed by the system. A deduction bar says that if the formula(e) appearing above it occurs in the tree, then the formula(e) below it should be added to the tableau. The rules are easily interpreted according to the intuitions assigned above to signs, formulae and information tokens. A rule like $\alpha_{(i)}$, for example, says that if A\B is not available for combination and x verifies such information, then this is because there is an A available at some token a, but the combination of a and x (notice that the order is relevant) does not produce B. Given the expansion rules,

α-rules	(i)	(ii)	(iii)	**β-rule**		
(α_1)	F A\B : x	F A/B : x	T A • B : x	$\dfrac{}{(\beta_1)\ TA : x \mid (\beta_2)\ FA : x}$		
(α_2)	T A : a*	T B : a*	T A : a*			
(α_3)	F B : a ∘ x	F A : x ∘ a	T B : x ↙ a			
σ-rules	(i)	(ii)	(iii)	(iv)	(v)	(vi)
(σ_1)	T A\B : x	T A\B : x	T A/B : x	T A/B : x	F A • B : x	F A • B : x
(σ_2)	T A : y	F B : y ∘ x	F A: x ∘ y	T B : y	T A : y	T B : y
(σ_3)	TB : y ∘ x	F A : y	F B : y	T A : x ∘ y	F B : x ↙ y	F A : x ↗ y

**: a new label* a *(not occurring previously in the derivation) must be introduced.*

Figure 1: Tableau expansion rules

the definition of the main data structure to be manipulated by the theorem proving (parsing) algorithm is straightforward: a derivation tree, $\mathcal{T}$, is simply a binary tree built from a set of given formulae by applying the rules. The next step is to define the conditions for a tree to be regarded as complete. Completion along with inconsistency are the notions upon which the algorithm's termination depends. It can be readily seen on *Figure 1* that for a finite set of formulae, the number of times α and σ rules can be applied increasing the number of SLFs (nodes) in $\mathcal{T}$ is finite. Unbounded application of β, however, might expand the tree indefinitely. In order to assure

termination, applications of β will be restricted to sub-formulae of formulae in $\mathcal{T}$. These notions are formalised in *Definition 4*.

Definition 4 *(Tree Completion) Given $\mathcal{T}$, a tree for a set of SLFs S, we say that a binary tree $\mathcal{T}^*$ is a tableau for S if $\mathcal{T}^*$ results from $\mathcal{T}$ through the application of an expansion rule. A tableau $\mathcal{T}^*$ is* linearly complete *if it satisfies the following conditions: (i) if $\alpha_1 \in \mathcal{T}$, then α_2 and $\alpha_3 \in \mathcal{T}$; (ii) if σ_1 and $\sigma_2 \in \mathcal{T}$, then $\sigma_3 \in \mathcal{T}$. A tree $\mathcal{T}^*$ is* complete *iff for every $A \in \mathcal{T}^*$ and every sub-formula A' of A, both F A' : x and T A' : x have been added to $\mathcal{T}^*$ by an application of the β-rule.*

Now, the first step towards building a counter-model for the denial of the formula to be proved is the search for a tree containing *potential* contradictions. Whether or not a potentially inconsistent tree is a counter-model for the formula will depend ultimately upon the constraints on the labelling algebra. This form of inconsistency is defined below.

Definition 5 *(Branch and Tree Inconsistency) A branch is* inconsistent *iff for some type X both T X and F X, labelled by any information token, occur in the branch. A tree is inconsistent iff its branches are all inconsistent.*

Given the definitions above, we are ready to define an algorithm for expanding linearly the derivation tree. For efficiency reasons non-branching rules will be exhaustively applied before we move on to employing β-rules. *Definition 6* presents the basic procedure for generating linear expansion for a branch.[7] The complete LKE algorithm, *Definition 8*, which uses the procedure below, will be presented after we have discussed tableau closure from the information frame perspective.

Definition 6 *(Algorithm: Linear Completion) Given $\mathcal{T}$, a LKE-tableau structure, we define the procedure:*

Linear-Completion($\mathcal{T}$)
1 do $\mathcal{T} \Leftarrow \alpha$-completion($\mathcal{T}$)
2 formula $\Leftarrow$ head[$\mathcal{T}$]
3 while ($\neg$completed($\mathcal{T}$) or consistent($\mathcal{T}$))
4 do if σ_1-type(formula)
5 then do formula$_{aux}$ $\Leftarrow$ search($\mathcal{T}$,σ_2) ▷ formula$_{aux}$ is a set of σ_2-type slf's
6 if formula$_{aux} \neq \emptyset$ ▷ σ_3-set results from combining σ_1 to each σ_2
7 then do σ_3-set $\Leftarrow$ combine-labels(σ_1,formula$_{aux}$)
8 σ_3-expansion $\Leftarrow$ α-completion(σ_3-set)
9 $\mathcal{T} \Leftarrow$ append($\mathcal{T}$,σ_3-expansion)
10 do formula $\Leftarrow$ next[$\mathcal{T}$]
11 return $\mathcal{T}$

We have seen above that the labels are means to propagate information about the formulae through the derivation tree. From a semantic viewpoint, the calculi addressed in this paper are obtained by varying the structure assigned to the set of formulae in the derivation[8]. Therefore, in order to verify whether a branch is closed for a calculus one has to verify whether the information frame satisfies the constraints which characterise the calculus. For instance, the standard Lambek calculus L does not allow any sort of structural manipulation of formulae apart from associativity; LP allows formulae to be permuted; LPE allows permutations and expansion (i.e. if B can be proved from the sequent Δ, A, Γ, then B can be proved from Δ, A, A, Γ); LPC allows permutation and contraction; etc. The definition below sets the algebraic counterparts of these properties.

Definition 7 *An information frame is: (i)* associative *if $x \circ (y \circ z) \sqsubseteq (x \circ y) \circ z$ and $(x \circ y) \circ z \sqsubseteq x \circ (y \circ z)$; (ii)* commutative *if $x \circ y \sqsubseteq y \circ x$; (iii)* contractive *if $x \circ x \sqsubseteq x$; (iv)* expansive *if $x \sqsubseteq x \circ x$; (v)* monotonic *if $x \sqsubseteq x \circ y$, for all $x, y, z \in \mathcal{P}$.*

[7] The reader will notice that if we had allowed σ_2 formulae to search for σ_1 types for combination, in the same way that σ_1s search for σ_2s, then linear expansion would not terminate for some cases. Consider for example the infinite sequence of σ applications: T A / B: x, T B / A : y, T A : z, T B : y ∘ z, T A : x ∘ (y ∘ z), T B : y ∘ (x ∘ (y ∘ z)),... A strategy to allow unrestricted σ-application without running into non-terminating procedures, as well as other practical and computational issues is discussed in [15].

[8] For instance, resource sensitive logics such as linear logic are frequently characterised in terms of multisets to keep track of the "use" of formulae throughout the derivation.

Now, we say that a branch is *closed* with respect to the labelling algebra if it contains SLFs of the form T X : x and F X : y, where x $\sqsubseteq$ y. Likewise, a tree is closed if it contains only closed branches. Checking for label closure will depend on the calculus being used, and consists basically of reducing information token expressions to a *normal form*, via properties (3)–(6), and then matching tokens and/or variables that might have been introduced by applications of the β-rule according to the properties or combination of properties (*Definition 7*) that characterise the calculus considered. It should be noticed that, in addition to the basic algorithm, heuristics might be employed to account for specific linguistic aspects. Some examples: (a) it could be assumed that all the bracketing for the strings is to the right thus favouring an incremental approach; (b) type reuse could be blocked at the level of the formulae, reducing the the computational cost of searches for label closure, since most of the calculi in the family covered by the system are resource sensitive; (c) priority could be given to juxtaposed strings for σ-rule application, etc. *Definition 8* gives the general procedure for tableau expansion, abstracted from the heuristics mentioned above.

Definition 8 *(Algorithm: LKE-completion) The complete tableau expansion for a LKE-tree $\mathcal{T}$ is given by the following procedure:*

```
expansion(T)
1  do closure-flag ⇐ no
2  while ¬( completed(T) or closed-T= yes)
3        do T⇐ linear-completion(T)
4           if ¬consistent(T) and label-closure(T)
5             then do closure-flag ⇐ yes
6             else do subformula ⇐ select-subformula(T)
7                     subformula_T ⇐ assign-label-T(subformula)
8                     subformula_F ⇐ assign-label-F(subformula)
9                     T_1 ⇐ append(T,{subformula_T})
10                    T_2 ⇐ append(T,{subformula_F})
11                    if ( expansion(T_1) = yes and expansion(T_2) = yes )
12                      then do closure-flag ⇐ yes
13 return closure-flag
```

As it is, the algorithm defined above constitutes a semi-decision procedure. This is due to the fact that even though the search space for signed formulae is finite, the search space for the labels is infinite. The labels introduced via β-rules are in fact universally quantified variables which must be instantiated during the label unification step. This represents no problem if we are dealing with theorems, i.e. trees which actually close. However, for completed trees with an open branch, the task might not terminate. In order to overcome this problem and bind the unification procedure we restrict label (variable) substitutions to the set of tokens occurring in the derivation — similarly to the way parameter instantiation is dealt with by liberalized quantification rules for first-order logic tableaux. In practice, the strategy adopted to reduce label complexity also employs the following refinements: (i) the tableau is linearly expanded keeping track of the choices made when σ-rules are applied (the options are kept in a stack); (ii) once this first step is finished, if the tableau is still open, then backtrack is performed until either the choices left over are exhausted or closure is achieved; (iii) only then is the β-rule applied. This explains the role played by the heuristics mentioned above.[9] We are now able to establish some results regarding the reduction laws mentioned in section 1.1.

Proposition 1 (Reduction Laws) *Let X, Y and Z be types, and $\mathcal{L}$ an information frame. The properties R1–R5 hold:*

Proof 1 *The proofs are obtained by straightforward application of* Definition 6 *and* Definition 8. *Below we illustrate the method by proving (R1) and (R2):*

(R1) To prove right application we start by assuming that it is verified by the identity token 1. *From this we have: 1- T X/Y • Y : m, 2- F X : 1 ∘ m = m. Then, we apply $\alpha_{(iii)}$ to 1*

[9] Furthermore, as we will show in section 2.3, if associativity is allowed at the syntactic level then it is possible to eliminate the branching rule for the class of calculi discussed here.

obtaining 3- T X/Y : n and 4- T Y : m ↙ n. The next step is to combine 3 and 4 via $\sigma_{(iv)}$ getting 5- T X : n ∘ (m ↙ n). Now we have a potential closure caused by 5 and 2. If we apply property (3) to the label for 5 we find that n ∘ (m ↙ n) ⊑ m, which satisfies the closure condition thus closing the tableau.

(R2) Let's prove left composition. As we did above, we start with: 1- T Z\Y • Y\X : m and 2- F Z\X : 1 ∘ m. Applying $\alpha_{(iii)}$ to 1 we get: 3- T Z\Y : a and 4- T Y\X : m ↙ a. Now, we may apply $\alpha_{(i)}$ to 2 and get: 5- T Z : b and 6- F X : b ∘ m. Then, combining 3 and 5 via $\sigma_{(i)}$: 7- T Y : b ∘ a. And finally 4 and 7 through the same rule: 8- T X : (b ∘ a) ∘ (m ↙ a). The closure condition for 8 and 6 is achieved as follows:

$$\begin{array}{lll} (b \circ a) \circ (m \swarrow a) & \sqsubseteq\ b \circ (a \circ (m \swarrow a)) & \textit{by associativity} \\ & \sqsubseteq\ b \circ m & \textit{by (3) and } \circ \textit{ being order-preserving} \end{array}$$

■

Even though L does not enjoy finite axiomatizability, the results above suggest that the calculus finds a natural characterization in LKE for associative information frames. In particular, the Division Rule (*R6*) can be regarded as L's characteristic theorem, since it is not derivable in weaker calculi such as AB, NL, and F. If we do not allow associative frames, we get NL. Stronger calculi such as LP, LPE, LPC and LPCE [18] can be obtained for the same general framework by assigning further properties to ∘ in the labelling algebra. Frames exhibiting combinations of monotonicity, expansivity, commutativity and contraction allow us to characterise these substructural calculi. Algebras that are both associative and commutative describe LP. Adding expansivity (weakening) to LP results in LPE. Associativity, commutativity and contraction describe LPC frames. LPCE is obtained by combining the properties of LPC and LPE algebras.

We end this section with a simple example requiring associativity: show, in L, that an NP (John), combined with a type (NP\S)/NP (likes) yields S/NP, i.e a type which combined with a NP will result in a sentence (Proof 2).

Proof 2 *Let's assume the following type–string correspondence: NP for* John, *(NP\S)/NP for* likes. *The expression we want to find a counter-model for is: 1- F NP • (NP\S)/NP ⊢$_L$ S/NP. Therefore, the following has to be proved: 2- T NP • (NP\S)/NP : m and 3- F S/NP : m. We proceed by breaking 2 and 3 down via $\alpha_{(iii)}$, obtaining: 4- T NP : a, 5- T (NP\S)/NP : m↙a, 6- T NP : b, and 7- F S : (m ∘ b).*

Now we start applying σ-rules (annotated on the right-hand side of each line):

8- T NP\S : (m ↙ a) ∘ b 5,6 $\sigma_{(i)}$
9- T S : a ∘ ((m ↙ a) ∘ b) 4,9 $\sigma_{(i)}$

We have derived a potential inconsistency between 7 and 9. Turning our attention to the information tokens, we verify closure for L *as follows:*

$$\begin{array}{lll} a \circ ((m \swarrow a) \circ b) & \sqsubseteq & (a \circ (m \swarrow a)) \circ b \quad \textit{by associativity} \\ & \sqsubseteq & m \circ a \quad \textit{by property (3)} \end{array}$$

■

2.3 Comparison with Existing Approaches

Early implementations of CG parsing relied on cut-free Gentzen sequents implemented via backward chaining mechanisms [16]. Apart from the fact that it lacks generality, since implementing more powerful calculi would involve modifying the code in order to accommodate new structural rules, this approach presents several sources of inefficiency. The main ones are: the generate-and-test strategy employed to cope with associativity, the non-determinism in the branching rules and in rule application itself. The impact of the latter form of non-determinism over efficiency can be reduced by testing branches for *count invariance* prior to their expansion and by performing sequent proof normalisation. However, non-determinism due to splitting in the proof structure still remains. As we move on to stronger logics and incorporate structural modalities such problems tend to get even harder.

An improved attempt to deal uniformly with multiple calculi is presented in [17]. In that paper, the theorem prover employed is based on proof nets, and the characterisation of different calculi is taken care of by labelling the formulae. For substructural calculi stronger than L, much

of the complexity (perhaps too much) is shifted to the label unification procedures. A strategy for improving such procedures by compiling labels into higher-order logic programming clauses is presented in [19] for NL and L. However, a comprehensive solution to the problem of binding label unification, a problem which arises as we move from sequents to labelled proof nets, has not been presented yet. Moreover, as discussed in [14], if we consider that the system is to be used as a parser, as a tool for linguistic study, the proof net style of derivation does not provide the clearest or most intuitive display of the proofs.[10]

In our approach, the burden of parsing is not so concentrated in label unification but is more evenly divided between the theorem prover and the algebraic checker. This is mainly due to the fact that the system allows for a controlled degree of non-determinism, present in the σ-rules, which enables us to reduce the introduction of variables in the labelling expressions to a minimum. We believe this represents an improvement on previous attempts. Besides this, controlling composition via bounded backtrack opens the possibility of implementing heuristics reflecting linguistic and contextual knowledge. In fact, we verify that, under the appropriate application of rules, we are able to eliminate the β-rule for a class of theorems.

Proposition 2 (Elimination Theorem) *All closed LKE-trees derivable by the application of the set of rules* $\mathcal{R} = \{\alpha_{(i)},...,\alpha_{(iii)},\sigma_{(i)},...,\sigma_{(vi)},\beta\}$ *can be also derived from* $\mathcal{R} - \{\beta\} + \{$ *assoc* $\}$.

The proof of this proposition can be done by defining an *abstract Gentzen relation*, proving a substitution lemma with respect to the labelling algebra (as in [7]), and showing that our consequence relation is closed under the relevant Gentzen conditions even if no β rule is employed. The proof appeals to the fact that no formula signed by F can occur in the sequents on the left-hand side of the entailment relation, since the calculi presented here do not have negation.[11] We believe that this result shows that, even though LKE label unification might be computationally expensive for substructural logics in general, the system seems to be well suited for categorial logics. We refer the reader to [15] for a more comprehensive discussion of these issues.

3 Conclusions and Further Work

We have described a framework for the study of categorial logics with different degrees of expressivity on a uniform basis, providing a tool for testing the adequacy of different CGs to a variety of linguistic phenomena. From a practical point of view, we have investigated the effectiveness and generality issues of a parsing strategy for CG opening an avenue for future developments. Moreover, we have pointed out some strategies for improving on efficiency and for dealing with more expressive languages, including structural modalities.

The architecture proposed seems promising. Its flexibility with respect to the variety of logics it deals with, and its modularity suggest some natural extensions to the present work. Among them: implementing a semantic module based on Curry-Howard correspondence between type deduction and λ-terms, adding local control of structural transformations (structural modalities) to the language, increasing expressivity in the information frames for covering calculi weaker than L (e.g. Dependency Categorial Grammar [20]), exploiting the derivational structure encoded in the labels to define heuristics for models of human attachment preferences etc. Problems for further investigation might include: the treatment of polymorphic types (by incorporating rules for dealing with quantification analogous to Smullyan's δ and γ rules [10] [21]), and complexity issues regarding how the general architecture proposed here would behave under more standard theorem proving methods.

References

[1] A. Ades and M. Steedman. On the order of words. *Linguistics and Philosphy*, 4:517–558, 1982.

[10] Proof nets and sequent normalisation have also been employed to get around spurious ambiguity (i.e. multiple proof for the same sentence, with the same semantics). Our approach does not exhibit this problem.

[11] Of course, without the β rule not all open trees generated will constitute downward saturated sets, since they might contain formulae which are not completely analysed.

[2] K. Ajdukiewicz. Die syntaktische konnexität. *Studia Philosophica*, 1(1–27), 1935. (Translation in S. McCall (ed) *Polish Logic 1920-1939* Oxford).

[3] Association for Computational Linguistics. *7th Conference of the European Chapter of the ACL*, Dublin, Ireland, March 1995. Morgan Kaufmann.

[4] Guy Barry and Glyn Morrill. *Studies in Categorial Grammar*, volume 5 of *Edinburgh Working Papers in Cognitive Science*. Centre for Cognitive Science, 1990.

[5] J. Barwise, D. Gabbay, and C. Hartonas. On the logic of information flow. *Bulletin of the IGPL*, 3(1):7–50, 1995. http://www.mpi-sb.mpg.de/guide/staff/ohlbach/igpl/Bulletin.html.

[6] Johan van Benthem. The semantics of variety. In Wojciech Buszkowski, Witold Marciszewski, and Johan van Benthem, editors, *Categorial Grammar*, volume 25, chapter 6, pages 141–151. John Benjamins Publishing Company, Amsterdam, 1988.

[7] Marcello D'Agostino and Dov Gabbay. A generalization of analytic deduction via labelled deductive systems I: Basic substructural logics. *Journal of Automated Reasoning*, To appear. Revised in March 1994.

[8] Marcello D'Agostino and Marco Mondadori. The taming of the cut. *Journal of Logic and Computation*, 4:285–319, 1994.

[9] David Dowty. Type raising, functional composition, and non-constituent conjunction. In Deirdre Wheeler Richard T. Oehrle, Emmon Bach, editor, *Categorial Grammars and Natural Language Structures*, pages 153–197. Reidel Publishing Co, Dordrecht, 1988.

[10] Melvin Fitting. *First-order Logic and Automatic Theorem Proving.* Texts and Monographs in Computer Science. Springer-Verlag, New York, 1990.

[11] Dov M. Gabbay. LDS – Labelled Deductive Systems, volume 1 — foundations. Technical Report MPI-I-94-223, Max-Planck-Institut fiir Informatik, 1994.

[12] Mark Hepple Glyn Morrill, Neil Leslie and Guy Barry. Categorial deductions and structural operations. In Barry and Morrill [4], pages 1 – 21.

[13] Joachim Lambek. On the calculus of syntactic types. In *Proceedings of the Symposia in Applied Mathematics*, volume XII, pages 166–178, Providence, Rhode Island, 1961. American Mathematics Society.

[14] Neil Leslie. Contrasting styles of categorial derivations. In Barry and Morrill [4], pages 113–126.

[15] Saturnino F. Luz Filho and Patrick Sturt. A new approach to categorial theorem proving. In preparation.

[16] Michael Moortgat. *Categorial Investigations.* Foris Publications, Dordrecht, 1988.

[17] Michael Moortgat. Labelled deductive systems for categorial theorem proving. Technical Report OTS-WP-CL-92-003, OTS, Utrecht, NL, 1992.

[18] Michael Moortgat. Lecture notes on categorial grammar. Reader for the course on Categorial Grammar given at the 5th ESSLLI - University of Lisbon, August 1993.

[19] Glyn Morril. Higher-order logic programming of categorial deduction. In EACL95 [3], pages 133–140.

[20] Martin Pickering and Guy Barry. Dependency categorial grammar and coordination. *Linguistics*, 31(5):855–902, 1993.

[21] Raymond M Smullyan. *First-Order Logic*, volume 43 of *Ergebnisse der Mathematik und ihrer Grenzgebiete*. Springer-Verlag, Berlin, 1968.