# Proof-Theoretic Aspects of Hybrid Type-Logical Grammars

Richard Moot, Symon Jory Stevens-Guille

# Proof-theoretic aspects of hybrid type-logical grammars

Richard Moot[1] and Symon Jory Stevens-Guille[2]

[1] LIRMM, Université de Montpellier, CNRS
[2] Ohio State University

**Abstract.** This paper explores proof-theoretic aspects of hybrid type-logical grammars, a logic combining Lambek grammars with lambda grammars. We prove some basic properties of the calculus, such as normalisation and the subformula property and also present a proof net calculus for hybrid type-logical grammars. In addition to clarifying the logical foundations of hybrid type-logical grammars, the current study opens the way to variants and extensions of the original system, including but not limited to a non-associative version and a multimodal version incorporating structural rules and unary modes.

**Keywords:** Lambek calculus · lambda grammar · type-logical grammar · proof theory · proof nets

## 1 Introduction

Hybrid type-logical grammars (HTLG), a logic introduced by kl12gap, combines the standard Lambek grammar implications with the lambda grammar operations. As a consequence, the lambda calculus term constructors of abstraction and application live side-by-side with the Lambek calculus operation of concatenation and its residuals. The logic is motivated by empirical limitations of its subsystems. It provides a simple account of many phenomena on the syntax-semantics interface, for which neither of its subsystems has equally simple solutions kl12gap,kl13dgap,kl13coord.

For instance, Lambek calculi struggle to account for medial extraction, as is required for the wide-scope reading of the universal in (1). Such cases are straightforwardly accounted for by lambda grammars. For the same reasons — namely the absence of directionality — lambda grammars cannot easily distinguish (2) from (3) worth14coord, whereas the distinction is trivial to implement in Lambek calculi.

1. Someone delivers every letter to its destination.
2. *Ahmed loves and dislikes dessert the pizza
3. Ahmed loves and Johani dislikes the pizza

In their paper on determiner gapping in hybrid type-logical grammar, [footnote 7]kl13dgap 'acknowledge that there remains an important theoretical issue:

the formal properties of our hybrid implicational logic are currently unknown'. In this paper, we prove basic properties of the natural deduction calculus of kl12gap (normalisation, decidability and the subformula property) and present a proof net calculus. This puts HTLG on a firm theoretical foundation, but also provides a framework for extensions of the logic. In addition, the proof net calculus provides a proof search method which is both flexible and transparent.

## 2 Natural deduction

HTLG syntactic terms are tuples of which the first element is a linear lambda term and the second is a type-logical formula drawn from the union of implicational linear logic and Lambek formulas. Given a set of atomic formulas $A$ (we will assume $A$ contains at least the atomic formula $n$ for noun, $np$ for noun phrase, $s$ for sentence, and $pp$ for prepositional phrase), the formula language of HTLG is the following.

- $T_{Logic} ::= T_{Lambek} \mid T_{Logic} \multimap T_{Logic}$
- $T_{Lambek} ::= A \mid T_{Lambek}/T_{Lambek} \mid T_{Lambek}\backslash T_{Lambek}$

Prosodic types are simple types with a unique atomic type $s$ (for *structure* or, in an associative context, *string*). Logical formulas are translated to prosodic types as follows.

$$\begin{aligned} \mathit{Pros}(T_{\text{Lambek}}) &= s \\ \mathit{Pros}(T_{\text{Logic}} \multimap T_{\text{Logic}}) &= \mathit{Pros}(T_{\text{Logic}}) \to \mathit{Pros}(T_{\text{Logic}}) \end{aligned}$$

The lambda terms of HTLG, called *prosodic terms*, are constructed as follows.

- Atoms: $+^{s\to s\to s}$, $\epsilon^{s}$, a countably infinite number of variables $x_0, x_1, \ldots$ for each type $\alpha$; by convention we use $p, q, \ldots$ for variables of type $s$.
- Construction rules:
  - if $M^{\alpha\to\beta}$ and $N^{\alpha}$, then $(MN)^{\beta}$
  - if $x^{\alpha}$ and $M^{\beta}$, then $(\lambda x.M)^{\alpha\to\beta}$

In what follows, we restrict the prosodic terms to linear lambda terms, requiring each $\lambda$ binder to bind exactly one occurrence of its variable $x$. This restriction is standard in HTLG.

The natural deduction rules for HTLG are given by Figure 1. The lexicon assigns to the word $p$ a formula $A$ and a linear lambda term $M$ of type $\mathit{Pros}(A)$. Since this term is linear, it contains exactly one free occurrence of $p$. When no confusion is possible (for example when a word appears several times in a sentence), we use the word itself instead of the unique variable $p$ (the formula $w$ has a purely technical role and cannot appear on the right hand side of the lexicon or axiom rule). An example would be $\lambda P.(P\mathit{everyone}) : (np \multimap s) \multimap s$. The elimination rules have the standard condition that no free variables are shared between $\Gamma$ and $\Delta$, which ensures $\Gamma, \Delta$ is a valid context. The introduction rules have the standard side-condition that $\Gamma$ contains at least one formula (ensuring

$$\frac{}{p^s : w \vdash M : A}\,\text{Lex} \qquad \frac{}{x^\alpha : A \vdash x^\alpha : A}\,\text{Ax}$$

$$\frac{\Gamma \vdash N^\alpha : A \qquad \Delta \vdash M^{\alpha\to\beta} : A \multimap B}{\Gamma, \Delta \vdash (MN)^\beta : B} \multimap E \qquad \frac{\Gamma, x^\alpha : A \vdash M^\beta : B}{\Gamma \vdash (\lambda x.M)^{\alpha\to\beta} : A \multimap B} \multimap I$$

$$\frac{\Gamma \vdash M^s : A/B \qquad \Delta \vdash N^s : B}{\Gamma, \Delta \vdash (M+N)^s : A} /E \qquad \frac{\Gamma, p^s : A \vdash (M+p)^s : B}{\Gamma \vdash M^s : B/A} /I$$

$$\frac{\Delta \vdash M^s : B \qquad \Gamma \vdash N^s : B\backslash A}{\Delta, \Gamma \vdash (M+N)^s : A} \backslash E \qquad \frac{p^s : A, \Gamma \vdash (p+M)^s : B}{\Gamma \vdash M^s : A\backslash B} \backslash I$$

$$\frac{\Gamma \vdash M[(\lambda x.N)P] : C}{\Gamma \vdash M[N[x := P]] : C}\,[\beta]$$

**Fig. 1.** Gentzen-Style ND Inference Rules for HTLG

that provable statements cannot have empty antecedents). The $\beta$ rule performs on-the-fly beta reduction on the lambda terms.

Before showing normalization, we first prove a standard substitution lemma.

**Lemma 1.** *Let $\delta_1$ be a proof of $\Gamma \vdash N : A$ and $\delta_2$ a proof of $\Delta, x : A \vdash M[x] : C$ such that $N$ and $M$ share no free variables, then there is a proof of $\Gamma, \Delta \vdash M[N] : C$.*

*Proof* We can combine the two proofs as follows, replacing the hypothesis $x : A$ of $\delta_2$ by the proof $\delta_1$.

$$\begin{array}{c} \vdots\ \delta_1 \\ \Gamma \vdash N : A \\ \vdots\ \delta_2[x := N] \\ \Gamma, \Delta \vdash M[N] : C \end{array}$$

Given that, by construction, $M$ and $N$ share no free variables, replacing $x$ by $N$ cannot make a rule application in $\delta_2$ invalid. □

Given that the $w$ atomic formula appearing on the left-hand side of the Lex rule is by construction forbidden to appear on the right-hand side of a sequent, this means that the substitution lemma can never apply to a lexical hypothesis (since there are no proofs of the form $\Gamma \vdash N : w$).

## 3 Normalisation

We show that HTLG is normalizing. A normal form for an HTLG proof is defined as follows.

**Definition 1.** *A derivation $D$ for HTLG is* normal *iff each major premiss of an elimination rule is either:*

1. *an assumption*
2. *a conclusion of an application of an E-rule.*

In general, we call a logic *normalizing* just in case there is an effective procedure for extracting normal proofs from arbitrary proofs. Based on this definition, any path in a normal proof starts with an axiom/lexicon rule, then passes through a (possibly empty) sequence of elimination rules as the major premiss, followed by a (possibly empty) sequence of introduction rules, ending either in the minor premiss of an elimination rule or in the conclusion of the proof.

To demonstrate HTLG is normalizing, we define a set of conversion rules — functions from derivations $D$ to derivations $D'$ — such that repeated application of the rules terminates in a normal derivation.

Figure 2 shows the conversion rules[3]. Note that, given the condition on the elimination rules, $N$ and $M$ cannot share free variables and that Lemma 1 therefore guarantees the reductions transform proofs into proofs. In what follows, we assume that $\beta$ reduction applies on an as-needed basis, ignoring its application for simplicity of presentation.

$$\dfrac{\begin{array}{c} x^s : A \\ \vdots\ \Pi_2 \\ \vdots\ \Pi_1 \quad (x+M)^s : B \\ N^s : A \quad M^s : A\backslash B \end{array}\ \backslash I}{(N+M)^s : B}\ \backslash E \quad \rightsquigarrow \quad \begin{array}{c} \vdots\ \Pi_1 \\ N^s : A \\ \vdots\ \Pi_2[x := N] \\ (N+M)^s : B \end{array}$$

$$\dfrac{\begin{array}{c} x^\alpha : A \\ \vdots\ \Pi_2 \\ \vdots\ \Pi_1 \quad M^\beta : B \\ N^\alpha : A \quad (\lambda x.M)^{\alpha\to\beta} : A \multimap B \end{array}\ \multimap I}{((\lambda x.M)N)^\beta : B}\ \multimap E \quad \rightsquigarrow \quad \begin{array}{c} \vdots\ \Pi_1 \\ N^\alpha : A \\ \vdots\ \Pi_2[x := N] \\ (M[x := N])^\beta : B \end{array}$$

**Fig. 2.** Conversion Rules

**Theorem 1.** *HTLG is strongly normalizing.*

*Proof* To show strong normalization, we need to show that there are no infinite reduction sequences. However, since each reduction reduces the size of the proof, this is trivial. □

**Theorem 2.** *Normalization for HTLG proofs is confluent.*

[3] The rule for the / directly parallels that for \, modulo directionality.

*Proof* It is easy to show weak confluence: whenever a proof can be reduced by two different reductions $R_1$ and $R_2$, then reducing either redex will preserve the other redex , and $R_1$ followed by $R_2$ will produce the same proof as $R_2$ followed by $R_1$. By Theorem 1 we therefore have strong confluence. □

**Corollary 1.** *HTLG proofs have a unique normal form.*

*Proof* Immediate by Theorem 1 and Theorem 2. Note that uniqueness is up to beta equivalence or, alternatively, each HTLG proof has a unique normal form proof with a beta normal term[4]. □

**Corollary 2.** *HTLG satisfies the subformula property.*

*Proof* This is a direct consequence of normalization (Theorem 1). In a normal form proof, every formula is either a subformula of one of the hypotheses or a subformula of the conclusion. □

Given that we only consider linear lambda terms, HTLG proofs have a number of beta reductions bounded from above by the total number of abstractions in the proof (those in the leaves plus those in the introduction rules). Therefore, decidability follows from the subformula property. However, we will give a more detailed complexity analysis in Section 6.

## 4 Proof nets

In proof theory, we generally have multiple proof systems for a single logic. Even though deductions in these systems are intertranslatable, shifting to a different proof system may make some properties of the logic easier to prove.

Proof nets are a graph theoretic representation of proofs introduced for linear logic by Girard. Proof nets remove the possibility of 'boring' rule permutations as they occur in the sequent calculus or natural deduction[5], solving the so-called problem of 'spurious ambiguity' in type-logical grammars.

We generally define proof nets as part of a larger class called *proof structures*. Proof nets are those proof structures which correspond to sequent (or natural deduction) proofs. We can distinguish proof nets from other proof structures by means of a correctness condition. As a guiding intuition, we have the following correspondence between sequent calculus/natural deduction proofs and proof nets.

logical rule = link + correctness condition
proof (proof net) = proof structure + correctness condition

A more procedural interpretation of this is that a proof structures represent the search space for proofs.

[4] As is usual in the lambda calculus, we do not distinguish alpha-equivalent lambda terms.

[5] For natural deduction, rule permutations are a problem only for the $\bullet E$ and the $\diamond E$ rules.

### 4.1 Proof structures

**Definition 2.** *A* link *is tuple consisting of a* type *(tensor or par), an* index *(from a fixed alphabet $I$, indicating the family of connectives it belongs to), a* list of premisses*, a* list of conclusions*, and an optional* main node *(either one of the conclusions or one of the premisses).*

A link is essentially a labelled hyperedge connecting a number of vertices in a hypergraph. The premisses of a link are drawn left-to-right above the central node, whereas the conclusions are drawn left-to-right below the central node. A par link displays the central node as a filled circle, whereas a tensor uses an open circle. For hybrid type-logical grammars, the set of indices is $\{\epsilon, +, @, \lambda\}$. The constructor $\epsilon$ represents the empty string (it doesn't correspond to a logical connective, although we can add one if desired). The (non-associative) Lambek calculus implications ($\backslash$, $/$) use the term constructor '$+$' for their links (in a multimodal context we can have multiple instances of '$+$', for example, '$+_1$', '$+_2$', but this doesn't change much), whereas the lambda grammar implication ($\multimap$) uses links labeled with @ (representing application, for its tensor link) and $\lambda$ (representing abstraction, for its par link).

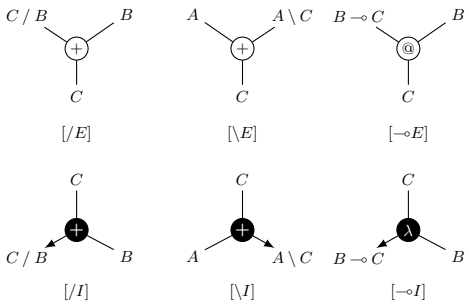


**Table 1.** Links for HTLG proof structures

From Table 1, it is clear that par links have one premiss and two conclusions, whereas tensor links have two premisses and one conclusion (we will see a tensor link with one premiss and two conclusions later). Par links have an arrow pointing to the main formula of the link, the main formulas of tensor links are not distinguished visually (but can be determined from the formula labels).

**Definition 3.** *A proof structure is a tuple $\langle F, L\rangle$, where $F$ is a set of formula occurrences (vertices labeled with formulas) and $L$ is a set of links such that each local neighbourhood is an instance of one of the links of Table 1, and such that:*

- *each formula is at most once the premiss of a link ,*
- *each formula is at most once the conclusion of a link.*

*The formulas which are not a conclusion of any link in a proof structure are its* hypotheses. *We distinguish between* lexical *hypotheses and* logical *hypotheses; lexical hypotheses are formulas from the lexicon, all other hypotheses are logical. The formulas which are not a premiss of any link in a proof structure are its* conclusions. *Formulas which are both a premiss and a conclusion of a link are* internal nodes *of the proof structure.*

*We say a proof structure with hypotheses $\Gamma$ and conclusions $\Delta$ is a proof structure of $\Gamma \vdash \Delta$, overloading the $\vdash$ symbol.*

**Definition 4.** *Given a proof structure $P$, a formula occurrence $A$ of $P$ is a* cut formula *if it is the main formula of two links. $A$ is an* axiomatic formula *in case it is not the main formula of any link.*

*Example 1.* As a very simple example, consider the lexicon containing only the words 'everyone' of type $(np \multimap s) \multimap s$ with prosodic term $\lambda P.(P\ everyone)$ and 'sleeps' of type $np \multimap s$ with prosodic term $\lambda z.(z + sleeps)$. Unfolding the lexical entries produces the proof structure shown in Figure 3. We use the convention of replacing lexical hypotheses with the corresponding word, so 'everyone' represents the formula $(np \multimap s) \multimap s$ and 'sleeps' the formula $np \multimap s$. There are no cut formulas in the figure, and all atomic formulas are axiomatic.

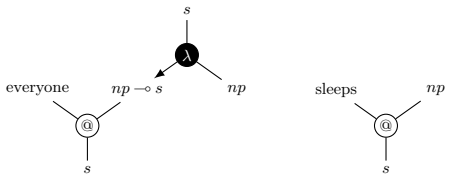


**Fig. 3.** Proof structure of 'everyone sleeps'.

**Definition 5.** *Given a proof structure $P$ and two distinct formula occurrences $x$, $y$ of $P$, both labeled with the same formula $A$, with $x$ a logical hypothesis of $P$ and $y$ a conclusion of $P$. Then $P'$, the* vertex contraction *of $x$ and $y$ in $P$, is the proof net obtained by deleting $x$ and $y$, adding a new node $z$ with label $A$ such that $z$ is the premiss of the link $x$ was a premiss of (if any) and the conclusion of the link that $y$ was the conclusion of (if any).*

The vertex contraction operation is a standard graph theoretic operation. In the current context, it operates like the cut or axiom rule in the sense that if $P_1$ is a proof net of $\Gamma, A \vdash \Delta$ and $P_2$ a proof net of $\Gamma' \vdash A, \Delta'$ with $x$ and $y$ the two occurrences of $A$, then the vertex contraction of $x$ and $y$ is a proof net of $\Gamma, \Gamma' \vdash \Delta, \Delta'$. Given that, in an intuitionistic context like the current one, all proof nets have a single conclusion we even have that if $P_1$ is a proof net of

$\Gamma, A \vdash C$ and $P_2$ a proof net of $\Gamma' \vdash A$, then the vertex contraction gives a proof net of $\Gamma, \Gamma' \vdash C$. Note that vertex contraction applies only to logical hypotheses and not to lexical ones.

Just like a logical link is a generalisation of a logical rule which is locally correct but need not be correct globally, a vertex contraction is a generalisation of the cut rule which is locally correct but need not be correct globally.

*Example 2.* Connecting the atomic formulas of the proof structure shown in Figure 3 produces the proof structure shown on the left of Figure 4. It has (the formulas corresponding to) 'everyone' and 'sleeps' as hypotheses (both lexical) and the formula $s$ as its conclusion.

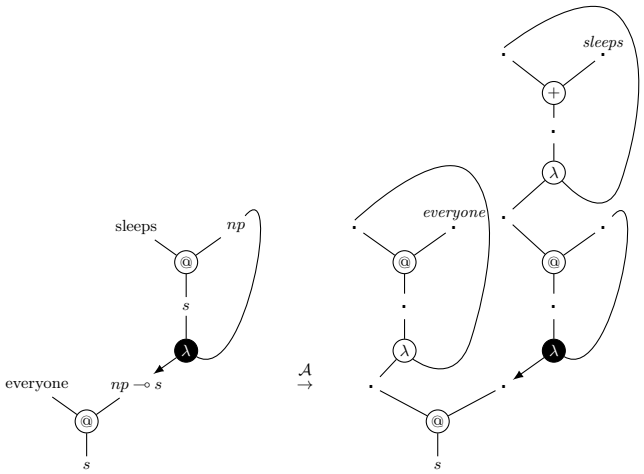


**Fig. 4.** Proof structure of 'everyone sleeps' after identification of the atomic formulas (left) and corresponding abstract proof structure (right).

**Definition 6.** *A* tensor graph *is a connected proof structure with a unique conclusion (root) node containing only tensor links. The trivial tensor graph is a single node.*

*Given a proof structure $P$, the* components *of $P$ are the maximal substructures of $P$ which are tensor graphs. A* tensor tree *is an acyclic tensor graph.*

For standard multimodal proof nets, we define correctness using tensor trees instead of the more general notion used here. Our results may be (graph theoretical representations of) lambda terms, and the $\lambda$ link represents the $\lambda$ binder for linear lambda terms. As is usual for lambda terms, we need to be careful about 'accidental capture' of variables. That is, we want avoid incorrect reductions such as $(\lambda x \lambda y (f\, x))(g\, y)$ (not a linear lambda term) to $\lambda y (f\, (g\, y))$.

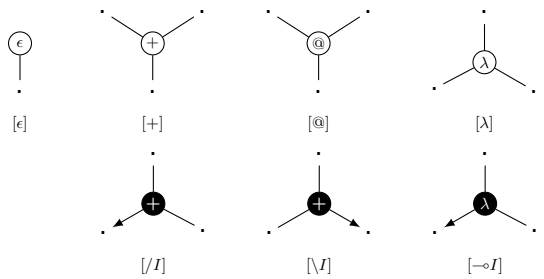


**Table 2.** Links for HTLG abstract proof structures

### 4.2 Abstract proof structures

As is usual for proof nets, correctness is defined on graph theoretic representations obtained from proof structures by forgetting some of the formula labels. We call these representations *abstract proof structures*. A more procedural way of seeing abstract proof structures is as ways of computing the structure of the antecedent. For hybrid type-logical grammars, this means abstract proof structures must contain some way of representing lambda terms in addition to the Lambek calculus structures.

**Definition 7.** *An abstract proof structure $A$ is a tuple $\langle V, L, l, h, c\rangle$ where $V$ is a set of vertices, $L$ is a set of the links shown in Table 2 connecting the vertices of $V$, $l$ is a function from the lexical hypothesis vertices of $A$ to the corresponding variables, $h$ is a function from logical hypothesis vertices to formulas, and $c$ is a function from the conclusion vertices of $A$ to formulas (a hypothesis vertex is a vertex which is not the conclusion of any link in $L$, and a conclusion vertex is a vertex which is not the premiss of any link in $L$).*

The links for abstract proof structures are shown in Table 2. The tensor links are shown in the topmost row, the par links in the bottom row, with the par links for the Lambek connectives on the left and in the middle, and the par link for the linear implication on the bottom right.

The $\lambda$ tensor link is the only non-standard link. Even though it has the same shape as the link for the Grishin connectives of mm12pnlg, it is used in a rather different way. The $\lambda$ tensor link does not correspond to a logical connective but rather to lambda abstraction over terms (or rather their graph theoretical representation). To keep the logic simple and the number of connectives as small as possible, we have chosen to make the $\epsilon$ link, corresponding to the empty string, a non-logical link as well. As a consequence, $\epsilon$ can appear only in lexical terms. However, if needed, it would be easy to adapt the logic by adding a logical connective 1 corresponding to $\epsilon$.

**Definition 8.** *Given a proof structure $P$, we obtain the corresponding abstract proof structure $\mathcal{A}(P) = A$ as follows.*

1. *we keep the set of vertices $V$ and the set of links $L$ of $P$ (but we forget the formula labels of the internal nodes),*
2. *logical hypotheses are kept as simple vertices, but we replace each lexical hypothesis $M : A$ of the proof structure by a graph $g$ corresponding to its lambda term $M$, the conclusion of $g$ is the vertex which was the lexical hypothesis of $P$, making the word subterm $w$ of $M$ a lexical hypothesis of the new structure,*
3. *we define $l$ to assign the corresponding word for each lexical hypothesis of the resulting graph, $h$ to assign a formula for all logical hypotheses, and $c$ to assign a formula to all conclusions.*

*Example 3.* Converting the proof structure on the left of Figure 4 to an abstract proof structure produces the abstract proof structure shown on the right. We have replaced 'everyone' by the structure corresponding to its lexical lambda term and similarly for 'sleeps'.

**Definition 9.** *A* lambda graph *is an abstract proof structure such that:*

1. *it has a single conclusion,*
2. *it contains only tensor links,*
3. *each right conclusion of a lambda link is an ancestor of its premiss,*
4. *removing the connection between all lambda links and their rightmost conclusion produces an acyclic and connected structure.*

Condition 3 avoids vacuous abstraction and accidental variable capture in the corresponding lambda term. Condition 4 is the standard acyclicity and connectedness condition for abstract proof structures, but allowing for the fact that lambda abstraction (but no other tensor links) can produce cycles.

Lambda graphs correspond to linear lambda terms in the obvious way, with the rightmost conclusion of the lambda link representing the variable abstracted over. This is a standard way of representing lambda terms in a way which avoids the necessity of variable renaming (alpha conversion).

**Proposition 1.** *a lambda term with free variables $x_1, \ldots, x_n$ corresponds to a lambda graph with hypotheses $x_1, \ldots, x_n$, with the @ tensor link corresponding to application, the $\lambda$ tensor link to abstraction, and the $+$ link and the $\epsilon$ link to the term constants of type $s \to s \to s$ and $s$ respectively. To keep the terms simple, we will write $(X + Y)$ instead of $((+X)Y)$.*

### 4.3 Structural rules and contractions

To decide whether or not a given proof structure is a *proof net* (that is, corresponds to a natural deduction proof), we will introduce a system of graph rewriting. The structural rules for the non-associative version of hybrid type-logical grammars are shown on the left-hand column of Table 3. We can obtain the standard associative version simply by adding the associativity rules for '+'; more generally, we can add any structural conversion for multimodal grammars, rewriting a tensor tree into another tensor tree with the same leaves (though not

necessarily in the same order) provided they do not overlap with the beta redex. The $\epsilon$ structural rules simply stipulate that '$\epsilon$' functions as the identity element for '+' (both as a left identity and as a right identity).

The key rewrite is the beta conversion rule. It is the graph theoretical equivalent of performing a beta reduction on the corresponding term. For the beta rewrite, we replace the two links (and the internal node) and perform two vertex contractions: $h_1$ with $c_1$ and $h_2$ with $c_2$. We update the functions $h$, $l$ and $c$ accordingly (if one of the $h_i$ was in the domain of $h$ and $l$ then so is the resulting vertex, and similarly for the $c_i$ and the $c$ function of the abstract proof structure).

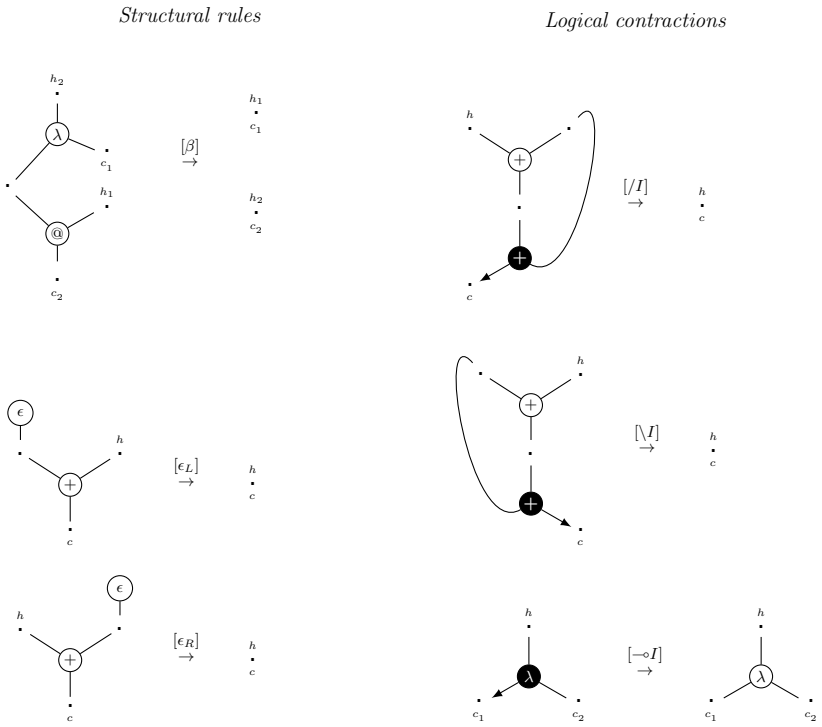


**Table 3.** Structural rules (left) and logical contractions (right) for HTLG proof nets.

To make the operation of the beta reduction clearer, a 'sugared' version of the contraction is shown in Table 4. Term labels have been added to the vertices of the graph to make the correspondence with beta-reduction explicit. This second picture is slightly misleading in that it suggests that $A_1$, $A_2$ and $A_3$ are disjoint substructures. This need not be the case: for example, $A_3$ can contain a lambda link whose right conclusion is a premiss of either $A_1$ or $A_2$. Similarly, in a logic

with the Lambek calculus product, the link for $[\bullet E]$ may connect premisses of both $A_1$ and $A_2$. A side condition on the $-\circ I$ conversion combined with the restriction of lexical entries to linear lambda terms will guarantee that $x$ ($c_1$) in the beta reduction is always a descendant of $N(h_2)$.

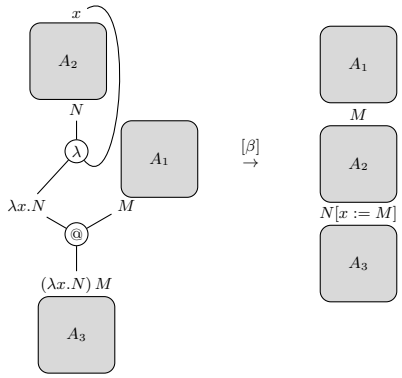


**Table 4.** Beta conversion as a structural rule with term labels added.

**Definition 10.** *We say a lambda graph is* normal *or beta*-normal *when it doesn't contain any redexes for the beta conversion.*

In addition to the structural rules, there are contractions for each of the logical connectives. Table 3 shows, on the right-hand column, the contractions for HTLG. For the Lambek implications, these are just the standard contractions. They combine a concatenation mode '+' with one of its residuals[6]. The contractions for the Lambek calculus implications are the standard contractions from mp.

The contraction for $-\circ I$ has the side condition that the rightmost conclusion of the $\lambda$ par link is a descendant of its premiss, passing only through tensor links. This is essentially the same condition as the one used by reductions, only without performing the actual contractions. This is because we want our abstract proof structures to represent the prosodic structure of a proof, which may contain lambda terms, just like the standard goal of abstract proof structures is always to compute the structure which would make the derivation valid.

Our rewrite calculus can be situated in the larger context of adding rewrite rules to the lambda calculus klop93crs,ck96alglambda. Even though the contractions for $[/I]$ and $[\backslash I]$ are not left-linear, since they correspond to terms

[6] To ensure confluence of '/' and '\' in the presence of $\epsilon$ we can add the side condition to the $[/I]$ and $[\backslash I]$ contractions that the component to which the par link is attached has at least one hypothesis other than the auxiliary conclusion of the par link. This forbids empty antecedent derivations and restores confluence.

$(M + x)/x$ and $x\backslash(x + M)$ respectively, this is not a problem because the occurrences of $x$ are bound occurrences klop93crs. In general, confluence can not be maintained in the presence of structural rules (or of the unary connectives) since the structural rules themselves need not be confluent. Confluence of beta reduction is guaranteed by not allowing any structural rewrite to overlap with the beta redex klop93crs.

## 5 Correctness of the proof net calculus

**Definition 11.** *A proof structure is a* proof net *whenever its abstract proof structure converts to a lambda graph.*

The reader can easily verify that the abstract proof structure shown on the right hand side of Figure 4 back on page 8 converts to the lambda graph *everyone* + *sleeps* — after one application of the $[\multimap I]$ conversion and three applications of the $\beta$ conversion — and is therefore a proof net.

We show that a proof net with premisses $A_1, \ldots A_k$ and conclusion $C$ converts to a lambda graph $M$ whenever $N_1 : A_1, \ldots, N_k : A_k \vdash M : C$ (where $N_i$ is the lexical lambda term or variable corresponding to $A_i$) is derivable, and vice versa.

**Lemma 2.** *If $\delta$ is a natural deduction proof of $N_1 : A_1, \ldots, N_k : A_k \vdash M : C$, then we can construct a proof net with premisses $A_1, \ldots, A_n$ and conclusion $C$ contracting to $M$.*

*Proof* Induction on the length $l$ of $\delta$. If $l = 0$, then we have either an axiom $x : A$ or a lexicon rule $M : A$ (with $M$ a linear lambda term with a single free variable $w$). In either case, the abstract proof structure will convert in zero steps to the required graph $M$.

If $l > 0$, we look at the last rule of the proof and proceed by case analysis.

If the last rule of the proof is the $/I$ rule, we are in the follow case.

$$\frac{\begin{array}{c}\vdots\ \delta \\ \Gamma, x : B \vdash N + x : A\end{array}}{\Gamma \vdash N : A/B}\ /I$$

Removing the last rule gives us the short proof $\delta$, and induction hypothesis gives us a proof net of $\Gamma, x : B \vdash N + x : A$. In other words, induction hypothesis gives us a proof net of $\Gamma, B \vdash A$ such that the underlying abstract proof structure converts, using a reduction sequence $\rho$, to $N + x$, with $x$ corresponding to $B$, as shown schematically in Figure 5.

We need to produce a proof net of $\Gamma \vdash N : A/B$. But this is done simply by adding the $/I$ link to the proof net of the induction hypothesis and adding a final $/I$ reduction as shown in Figure 6.

The cases for $\backslash I$ and $\multimap$ are similar, adding the corresponding link and conversion to the proof net obtained by induction hypothesis.

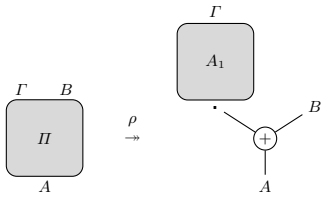


**Fig. 5.** Conversion sequence of Figure **??** with the final $[/I]$ contraction removed.

.

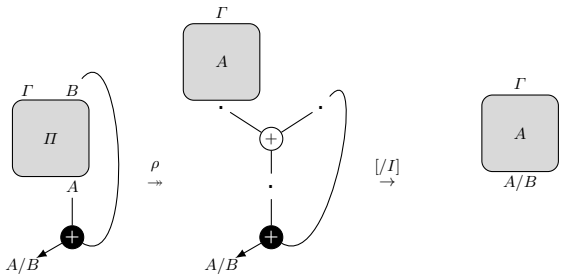


**Fig. 6.** Conversion sequence of a proof net ending with a $[/I]$ contraction

.

The cases for the elimination rules $/E$, $\backslash E$ and $\multimap E$ simply combine the two proof nets obtained by induction hypothesis with the corresponding link.

If the final rule is the $\beta$ rule or a structural rule, we simply add, respectively, the $\beta$ reduction and the corresponding structural conversion. □

**Lemma 3.** *Given a proof net $\Pi$ with premisses $A_1, \ldots A_n$ and conclusion $C$ converting to a lambda graph $M$, there is a natural deduction proof $N_1 : A_1, \ldots, N_k : A_n \vdash M : C$.*

We proceed by induction on the number of conversions $c$.

If $c = 0$ there are no conversions. As a consequence, there are no par links in the proof net. We proceed by induction on the number of tensor links $t$ in the proof net.

If $t = 0$, the proof net consists of a single formula $A$ and the abstract proof structure is either a single vertex $x$ (in the case of a hypothesis), corresponding to a proof $x : A$ or a term $M$ corresponding to a lexical entry, corresponding to a proof $M : A$.

If $t > 0$, one of the hypotheses of the proof structure must be the main formula of its link. Removing this link will produce two proof nets and we can apply the induction hypothesis to obtain natural deduction proofs for each, combining them with the appropriate elimination rule.

If $c > 0$, we look at the last conversion and proceed by case analysis.

If the last conversion is a $\beta$ conversion or a structural rule, then induction hypothesis gives us a proof $\delta$ of $\Gamma \vdash M' : C$, which we can extend using the $\beta$ rule (or structural rule) on $M'$ to produce $M$ and a proof of $\Gamma \vdash M : C$.

If the last conversion is a $\multimap I$ contraction, we are in the case shown in Figure 7. The final lambda graph corresponds to the linear lambda term $M[\lambda x.N]$.

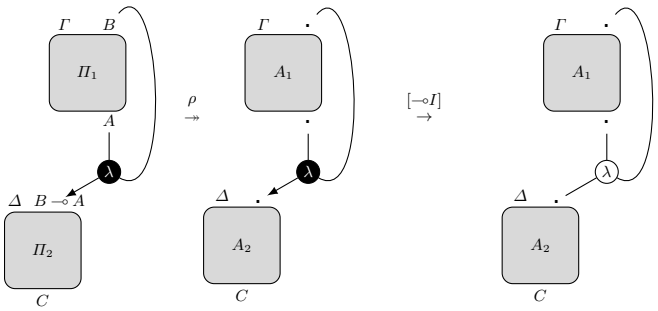


**Fig. 7.** Conversion sequence of a proof net ending with a $[\multimap I]$ contraction

.

Removing the final $[\multimap I]$ conversion produces two proof nets $\Pi_1$ and $\Pi_2$ with strictly shorter conversion sequences (again because we removed the final conversion and divided the other conversions) shown in Figure 8. Therefore, induction hypothesis gives us a proof $\delta_1$ of $\Gamma, x : B \vdash N : A$ and a proof $\delta_2$ of $\Delta, z : B \multimap A \vdash M[z] : C$ and we need to combine these into a proof of $\Gamma, \Delta \vdash M[\lambda x.N] : C$. This is done as follows.

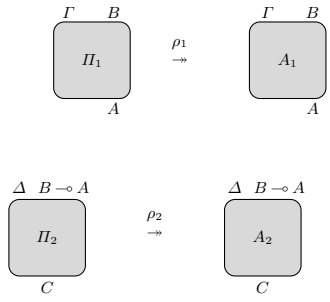


**Fig. 8.** Conversion sequence of Figure 7 with the final $[\multimap I]$ conversion removed

$$\begin{array}{lc} \Gamma & \begin{array}{c} x : B \\ \vdots\ \delta_1 \\ N : A \end{array} \\ \Delta & \dfrac{}{\lambda x.N : B \multimap A}\ \multimap I \\ & \begin{array}{c} \vdots\ \delta_2, z := \lambda x.N \\ M[\lambda x.N] : C \end{array} \end{array}$$

The substitution of $\lambda x.N$ for $z$ presupposes that the formula $B \multimap A$ is not a lexical hypothesis, but given that it is the conclusion of a link in the full structure, it cannot be a hypothesis of the full structure and therefore we can apply the substitution. This is the reason we distinguish between a par link for $\lambda$ (which corresponds to the introduction rule of the logical connective $\multimap$) and a tensor link (which corresponds to abstraction at the term level and need not correspond to a logical rule since it can come from a complex lexical entry as well).

The cases for $/I$ and $\backslash I$ are similar, and simply follow mp. □

## 6 Complexity

Given the proof net calculus described in the previous sections, complexity analysis of hybrid type-logical grammars and several of its variants becomes simple.

**Theorem 3.** *HTLG parsing is NP complete*

*Proof* Since HTLG contains lexicalized ACG as a fragment, NP-hardness follows from Proposition 5 of yk05acg, so all that remains to be shown is that HTLG is in NP.

In order to shown that HTLG parsing is NP-complete we only need to show that, given a non-deterministic proof search procedure we can verify whether a proof candidate is an actual proof in polynomial time. Given a sentence, we non-deterministically select a lexical formula for each word, then non-deterministically enumerate all proof structures for these lexical formulas. □

The proof of Theorem 3 is very general and can easily be adapted to variants and extensions of HTLG. For example, we can add the connectives for '$\bullet$', '$\diamond$' and '$\Box$' and mode information (as in the multimodal versions of the Lambek calculus M95) while maintaining NP-completeness.

When adding structural rules, complexity analysis becomes more delicate. Adding associativity, as in the original formulation of hybrid type-logical grammars, doesn't change the complexity, since we can simply use the strategy of [Section 7]mp to ensure polynomial contraction of proof structures. So we can actually strengthen Theorem 3 to the following.

**Theorem 4.** *$HTLG_{/_i,\bullet_i,\backslash_i,\diamond_i,\Box_i}$ parsing, with associativity for some modes $i$, is NP complete.*

In general, NP completeness will be preserved whenever we provide the set of structural rules with a polynomial time contraction algorithm. When we do not have a polynomial contraction algorithm, we can still show information about the complexity class: when we add structural rules but use the standard restriction that the tree rewrites allowed by the structural rules are linear (no copying or deletion of leaves) and do not increase the size of the tree, then the resulting logic is PSPACE complete, following the argument of [Section 9.2]diss.

**Theorem 5.** *$HTLG_{/_i,\bullet_i,\backslash_i,\diamond_i,\Box_i}$ parsing with any finite set of non-expanding structural rules is PSPACE complete.*

This gives an NP lower bound and a PSPACE upper bound for any HTLG augmented with the multimodal connectives and a fixed set of structural rules, and NP completeness can be shown by providing a polynomial contraction algorithm.

## 7 Conclusion

We have investigated the formal properties of hybrid type-logical grammars and proved several standard results for them. This solves the question of the theoretical foundations of the system, left open by kl13dgap.