# Application Description
# A Short Introduction to Grail

Richard Moot

## 1 Introduction

Grail is an automated theorem prover designed for prototyping and debugging grammar fragments in the multimodal Lambek calculus. This paper will give a short description of the multimodal Lambek calculus and its implementation in Grail.

The main differences between Grail and other theorem provers for the multimodal Lambek calculus, such as Glyn Morrill's CATLOG resolution theorem prover and Bob Carpenter's natural deduction theorem prover, are that Grail has no restrictions on the form of formulas and has an open-ended, user-defined set of structural rules.

In addition, Grail allows for interactive operation by portraying a representation of the current state of the computation and allowing the user to interact with this. It is used both as an education and research tool.

## 2 The multimodal Lambek calculus

Before we start with a description of the theorem prover itself, it is useful to give at least a short introduction to the multimodal Lambek calculus as used by Moortgat (1997). The multimodal Lambek calculus **NL**$\Diamond_{\mathcal{R}}$ is the nonassociative Lambek calculus **NL** (Lambek 1961), extended with unary modalities '$\Diamond$' and '$\Box^{\downarrow}$' and a set $\mathcal{R}$ of structural postulates.

**Definition 1** *The formulas of* **NL**$\Diamond_{\mathcal{R}}$ *are defined from a set of atomic formulas* $\mathcal{A}$*, a set* $I$ *of binary indices and a set* $J$ *of unary indices as follows, where* $i \in I$ *and* $j \in J$*.*

$$\mathcal{F} ::= \mathcal{A} \mid \Diamond_j \mathcal{F} \mid \Box_j^{\downarrow} \mathcal{F} \mid \mathcal{F} /_i \mathcal{F} \mid \mathcal{F} \bullet_i \mathcal{F} \mid \mathcal{F} \backslash_i \mathcal{F}$$

Intuitively, a formula of the form $A/_i B$ (resp. $B\backslash_i A$) looks to the right (resp. left) for a formula of type $B$ to yield a formula of type $A$.

The full logic supported by Grail contains three additional connectives, '$\bullet_i$', '$\Diamond_j$' and '$\Box_j^{\downarrow}$'. $A \bullet_i B$ corresponds to an $A$ and $B$ resource composed in mode $i$. '$\Diamond_j$' and '$\Box_j^{\downarrow}$' behave like the future possibility and past necessity operator from temporal logic. In

$$
\begin{array}{ll}
V(\Box_j^{\downarrow} A) & = \{y \mid \forall x.(xR_j^2 y \rightarrow x \in V(A))\} \\
V(\Diamond_j A) & = \{x \mid \exists y.(xR_j^2 y \wedge y \in V(A))\} \\
V(A/_i B) & = \{y \mid \forall x.z.((xR_i^3 yz \wedge z \in V(B)) \rightarrow x \in V(A))\} \\
V(B\backslash_i A) & = \{z \mid \forall x.y.((xR_i^3 yz \wedge y \in V(B)) \rightarrow x \in V(A))\} \\
V(A \bullet_i B) & = \{x \mid \exists y.z.(xR_i^3 yz \wedge y \in V(A) \wedge z \in V(B))\}
\end{array}
$$

Table 1: Model theoretic evaluation of complex formulas

$$
\begin{array}{rr}
\textit{livia} \in V(np) & \textit{hates} \in V((np\backslash_a s)/_a np) \\
\textit{richie} \in V(np) & \textit{shot} \in V((np\backslash_a s)/_a np) \\
\textit{tony} \in V(np) & \textit{someone} \in V((s/_a np)\backslash_a s)
\end{array}
$$

Table 2: Example lexical assignments

our linguistic applications, however, the unary connectives will correspond to adding and removing linguistic features.

It is important to note that we use a *multimodal* system and that a formula $A/_a B$ may have different behaviour than, for example, a formula $A/_c B$. In Example 4 we will see how different structural rules can apply depending on which mode we use.

Kripke frames $\mathcal{F}$ for $\mathbf{NL}\Diamond_{\mathcal{R}}$ are tuples $\langle W, \{R_j^2\}_{j\in J}, \{R_i^3\}_{i\in I}\rangle$. The 'worlds' $w \in W$ are our linguistic resources and the accessibility relations $R_j^2$ and $R_i^3$ model the unary and binary composition of these resources.

Kripke models $\mathcal{M}$ for the multimodal Lambek calculus are tuples of the form $\langle \mathcal{F}, V\rangle$ where $\mathcal{F}$ is a Kripke frame and $V$ is a valuation function. $V$ assigns subsets of $W$ to formulas in such a way that the statements of Table 1 hold for complex formulas.

A *lexicon* for $\mathbf{NL}\Diamond_{\mathcal{R}}$ is an assignment of worlds to the valuation of a certain formula.

**Example 2** *An example of a lexicon for $\mathbf{NL}\Diamond_{\mathcal{R}}$ is shown in Table 2. Given that these worlds corresponds to the words of the grammar, we choose to given them mnemonic names.*

The problem we are interested in is the following: given words $w_1, \ldots, w_n$ which are given an valuation according to the lexicon, we want to compute which compositions of these words by means of the accessibility relations $R_j^2$ and $R_i^3$ are in $V(s)$, or, depending on our application, in the valuation of some other formula. The resources in $V(s)$ then correspond to the valid sentences of our grammar. In other words, we want to generate a tree of words $w_1, \ldots, w_n$ in such a way that the root of this tree is in $V(s)$.

**Example 3** *For the lexicon of the previous example, we can show that for 'tony hates richie' when the worlds are related as shown in Figure 1 on the following page on the left, $x_1 \in V(s)$.*

We can require $V$ to satisfy additional constraints corresponding to the structural postulates of $\mathcal{R}$. We refer the reader to Kurtonina (1995) for the restrictions on the form of these postulates and for soundness and completeness results.

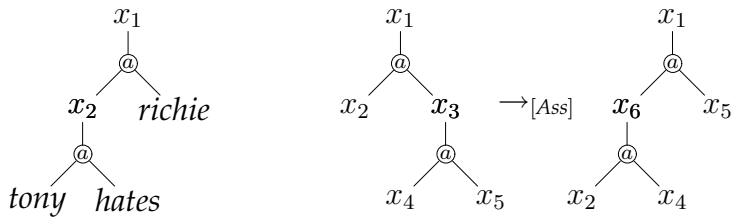


Figure 1: Graphical representation of the accessibility relation

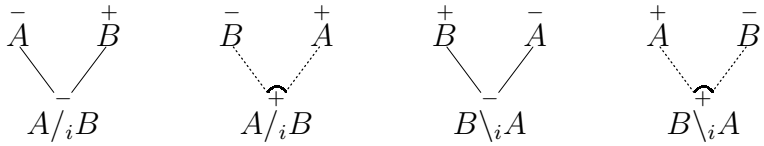


Table 3: Formula unfoldings for '/' and '\'

**Example 4** *We can require mode $a$ to be associative using two frame constraints, one of which is shown below.*

$$\forall x_1.x_2.x_3.x_4.x_5.\ x_1R_ax_2x_3 \wedge x_3R_ax_4x_5 \rightarrow \exists x_6.\ x_1R_ax_6x_5 \wedge x_6R_ax_2x_4 \qquad \textit{(Ass)}$$

*At the graphical level, this constraint can be seen as shown in Figure 1 on the right.*

# 3 Proof nets

In addition to having a model theoretic side, there are several proof theoretic representations of $\mathbf{NL}\Diamond_{\mathcal{R}}$. The Grail theorem prover uses a proof theoretic innovation introduced by Girard (1987), called *proof nets*. The advantage of proof nets over other representations is that they are inherently redundancy-free, that is, different proof nets correspond to different linguistic objects.

Obtaining a proof net for a given set of lexical assignments and a given goal formula consists of three stages.

In the first stage we unfold the formulas, depending on their polarity, according to Table 3. Note the difference between dotted and solid links. Grail displays these unfolded formulas as shown in Figure 2 on the next page. Negative atomic formulas are portrayed in black, positive atomic formulas are drawn in white.

For the second stage, we have to connect axiomatic formulas to formulas of opposite polarity. It is possible to let Grail do this automatically or to perform the axiom links manually. The latter is especially recommended in the case of larger proof nets, because in the worst case there might be $n!$ different linkings for lexical formulas with $2n$ atomic subformulas. To make an axiom link manually, the user can click one of the atomic formulas. We start with the leftmost, negative $np$ which corresponds to 'livia'. After

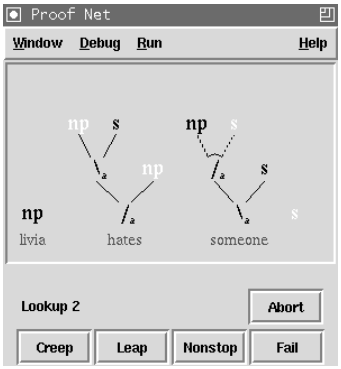


Figure 2: The formula unfolding for 'Livia hates someone'

clicking this $np$ all possible positive $np$'s to which we can link it will be marked by a white box, as shown in Figure 3 on the left.

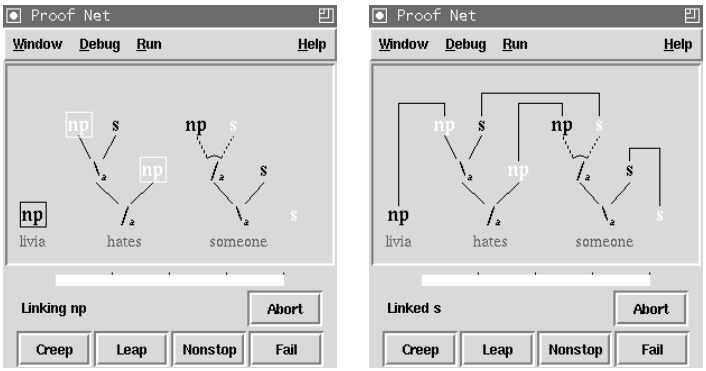


Figure 3: Producing a proof net

We select the leftmost positive $np$. An axiom link connecting the two $np$'s will now appear. Grail will keep track of any other possibilities for connecting the initial negative $np$, so we don't have to worry about mistakes. If the current choice fails to provide a solution for any reason the computation will backtrack and try the rightmost positive $np$. The second stage is completed when every atomic formula is connected to an atomic formula of opposite polarity. We call the resulting structure a *proof structure*.

In the final stage we check if it is also a proof net. There are many ways of doing this, but the one which is conceptually (though not computationally) the simplest is probably the switching criterion of Danos & Regnier (1989). For this criterion, we see the dotted links as switches which we can be set to either the left or the right. For a proof net, every such switching must be acyclic and connected. We can easily verify the both switchings for Figure 3 on the right satisfy this.

Grail checks acyclicity and connectedness already during the second stage where we are connecting the axioms, using an algorithm proposed by Danos (1990) which has an

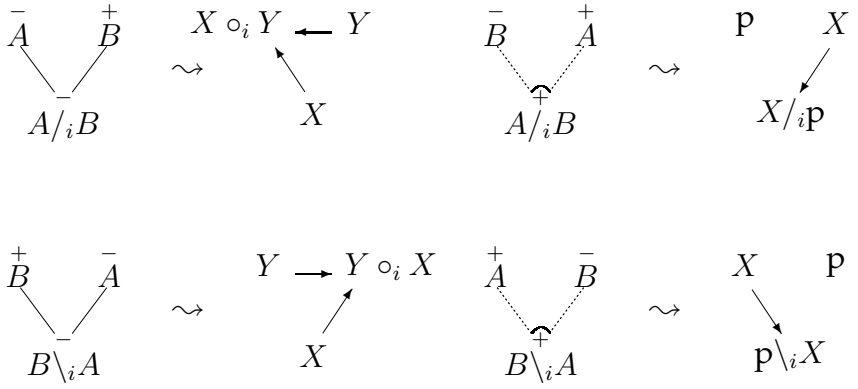


Figure 4: Computing the label from the proof net

$O(n^2)$ complexity. For example, connecting the two $s$ formulas of 'someone' in Figure 2 on the preceding page would result in a cyclic proof structure.

# 4 Semantic labeling

Proof nets as described in the previous section are only discriminate enough for validity in multiplicative linear logic. This is not enough for our linguistic applications because it makes the incorrect prediction that every permutation of a valid sentence is itself also a valid sentence. Checking acyclicity and connectedness is also not helpful in the case of the unary connectives and would allow us to conclude, for example, $\Diamond_j A \dashv\vdash A$ for all $A$ and $j$.

We introduce a new level of description in the form of semantic labeling to add the required discriminatory power to the calculus. We will apply the semantic labels as constraints on the derivability. This method is quite close to constructing a canonical model for $\mathbf{NL}\Diamond_{\mathcal{R}}$.

Figure 4 shows how to assign a label to the output formula of a proof structure based on the labels assigned to the input formulas and on the way they are connected. See Moortgat (1997) for the label assignments for the other links.

The arrows indicate the flow of information, for example, the negative $/_i$ link in the top left of Figure 4 indicates that if the formula $A/_iB$ is assigned label $X$ and the formula $B$ is assigned label $Y$ then the formula $A$ is assigned the label $X \circ_i Y$. Lamarche (1994) shows that for every proof net the assignments above generate a unique semantic label.

When portrayed as a tree, the semantic label for the current proof structure looks as shown in Figure 5 on the next page on the left.

We call the '/' constructor, drawn in gray in Figure 5 on the following page, an *auxiliary constructor*, and say a proof net is *correct* whenever we can remove all auxiliary constructors by their conversion. The conversion for the '\' and '/' constructors are

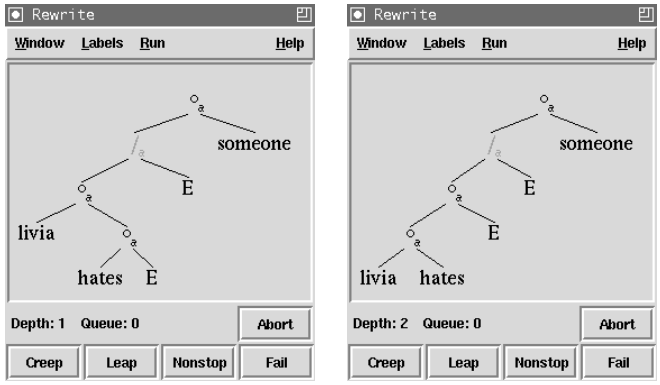


Figure 5: Label for 'Livia hates someone'

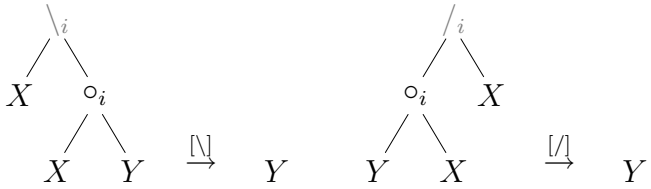


Figure 6: Conversions for '\' and '/'

shown in Figure 6. The other connectives each have their own conversion, see Moortgat (1997) for details.

Like with the axiom linking, the user can guide the computation at this stage; by clicking on a node of the label a list of possible conversions with that node at its root will appear. In this case if our logic has the associativity postulate we discussed in Example 4, it can apply to the parent node of 'livia', producing the label shown in Figure 5 on the right.

At this moment we are in the right position to apply the '/' conversion. We can do so by clicking on the '/' node and applying the appropriate conversion there. The resulting label looks much like the configuration shown in Figure 1 on page 3 on the left.

On the other hand if we do *not* have an associativity postulate we will not be able to remove the '/' auxiliary constructor and the proof net will be incorrect.

A second proof net exists for 'livia hates someone' and it is shown in Figure 7 on the following page on the left. The corresponding semantic label is shown in the same figure on the right. Note that after the '/' conversion this label would correspond to the string 'someone hates livia' for which we can generate the required 'livia hates someone' only through application of some commutativity postulate. For the current grammar, however, this would be undesirable, since it would implicitly claim that 'livia hates someone' and 'someone hates livia' have the same denotation.

Performing the conversions manually is a good way of debugging a fragment, since it can point to missing or insufficiently general structural postulates.

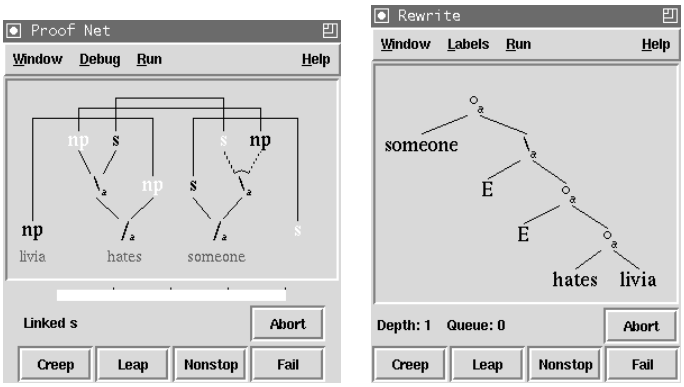


Figure 7: Second proof net and label

# 5 Computational Aspects

When running automatically, Grail employs a number of strategies for early failure.

First of all, the detection of cyclic or disconnected proof structures, which we talked about in Section 3 on page 3, can be performed during the linking stage and exclude many axiom links.

Secondly, for non-commutative modes of composition we can restrict ourselves, without loss of generality, to *planar* axiom links (Roorda 1991). Notice the crossing axiom links in Figure 7, for example.

Finally, we can sometimes apply eager evaluation of the auxiliary constructors, that is, we can do part of the label rewriting while we are still performing the axiom links. Grail detects automatically when this can be done safely. Tabulation is used to prevent many cases of recomputation.

All in all, the algorithm performs quite well given the PSPACE completeness of the decision problem for $\mathbf{NL}\Diamond_{\mathcal{R}}$.

# 6 Conclusion

We have given a (very) short description of the Grail theorem prover for the multimodal Lambek calculus and shown how the user can cooperate with or guide the theorem prover if this is desired. More details on Grail and the early failure mechanisms used in it can be found in (Moot 2001).

# References

Danos, V. (1990), La Logique Linéaire Appliquée à l'étude de Divers Processus de Normalisation (Principalement du $\lambda$-Calcul), PhD thesis, University of Paris VII.

Danos, V. & Regnier, L. (1989), 'The structure of multiplicatives', *Archive for Mathematical Logic* **28**, 181–203.

Girard, J.-Y. (1987), 'Linear logic', *Theoretical Computer Science* **50**, 1–102.

Kurtonina, N. (1995), Frames and Labels. A Modal Analysis of Categorial Inference, PhD thesis, OTS Utrecht, ILLC Amsterdam.

Lamarche, F. (1994), Proof nets for intuitionistic linear logic I: Essential nets, Technical report, Imperial College.

Lambek, J. (1961), On the calculus of syntactic types, *in* R. Jacobson, ed., 'Structure of Language and its Mathematical Aspects, Proceedings of the Symposia in Applied Mathematics', Vol. XII, American Mathematical Society, pp. 166–178.

Moortgat, M. (1997), Categorial type logics, *in* J. van Benthem & A. ter Meulen, eds, 'Handbook of Logic and Language', Elsevier/MIT Press, chapter 2.

Moot, R. (2001), Proof Nets for Linguistic Analysis, PhD thesis, Utrecht Institute of Linguistics OTS, Utrecht University. To appear.

Roorda, D. (1991), Resource Logics: A Proof-theoretical Study, PhD thesis, University of Amsterdam.