

# CHAPTER 9

---

## COMPLEXITY

TABLE 9.1 gives an overview of the complexity results for various fragments of  $\mathbf{NL}\diamond_{\mathcal{R}}$ . Even without contraction, weakening or similar structural rules where information is copied or deleted, the full logic  $\mathbf{NL}\diamond_{\mathcal{R}}$  is undecidable, as we will demonstrate in Section 9.3.

–	Undecidable	$\mathbf{NL}\diamond_{\mathcal{R}}$
⋮		
+	PSPACE	$\mathbf{NL}\diamond_{\mathcal{R}-}$
+	NP	<b>MILL1, LP</b>
+	?	<b>L</b>
+	P	
+	$n^6$	<b>NL,LTAG</b>
+	$n^3$	<b>AB</b>

Table 9.1: Complexity results for different fragments of  $\mathbf{NL}\diamond_{\mathcal{R}}$

In Section 9.2, however, we will show that if we disallow structural rules which remove unary connectives the resulting logic will be PSPACE complete. In Chapter 5 we already gave a number of examples of phenomena which could be described by the NP complete logic **MILL1** and related this logic to fragments of the multimodal Lambek calculus. Other NP complete

systems include **LP**, as shown by Kanovich (1991), and **L** extended with a commutative version of the right implication rule, as shown by Dörre (1996). Emms (1993a) shows the recognizing capacity of this logic is beyond that of context free grammars..

De Groote & Lamarche (2001) prove that, given a bracketed input, the decision problem for **NL** can be solved in  $O(n^6)$  time and in the next chapter I will present a fragment of  $\mathbf{NL} \diamond_{\mathcal{R}}$  with a restriction on the allowed formulas and fixed set of structural rules for which we can use the  $O(n^6)$  parsing algorithms which have been proposed for LTAGs.

Finally, there are simple **AB** grammars, precursors to the Lambek calculus introduced by Ajdukiewicz (1935) and Bar-Hillel (1964), which can be seen as the Lambek calculus with only the  $[L/], [L\setminus], [Ax]$  and  $[Cut]$  rules. **AB** grammars have been shown to be weakly equivalent to context free grammars by Bar-Hillel, Gaifman & Shamir (1964), and can therefore be compiled into context free grammars after which we can use any of the standard  $O(n^3)$  algorithms which exist for context free grammars.

Some problems still remain open: while it is relatively simple to show that the associative Lambek calculus is in NP, for example by the translation function of Section 5.1.4, it is still unknown if **L** is NP complete or if there exists a polynomial algorithm for deciding provability. Though Pentus (1995) (1997) has shown **L** grammars to be weakly equivalent to context free grammars, his proposed translation is exponential. Only some polynomial results for fragments of **L** have been found: Aarts (1994) shows that we can use polynomial parsing for the Lambek calculus with restricted formula complexity, whereas de Groote (1999b) gives an exponential algorithm based on proof nets and tabulation which performs in polynomial time for a subset of the complete logic, but so far the exact complexity for the full logic has remained elusive.

## 9.1 NP Complete Fragments

It seems quite reasonable to demand that we can determine the existence of a conversion sequence in polynomial time. In other words, we would like to restrict ourselves to packages of structural rules for which we have a polynomial time algorithm for finding a conversion sequence ending in a hypothesis tree, if such a conversion sequence exists.

Providing such an algorithm will immediately show the corresponding instance of  $\mathbf{NL} \diamond_{\mathcal{R}}$  is in NP, and, if the logic contains some form of commutativity, even NP complete.

In Section 4.5 we have already given an  $O(n^2)$  contraction algorithm for **MLL**, whereas in Section 7.8 we provided a  $O(n^2)$  contraction algorithm for **L**, though in the case of **L** this merely shows **L** to be in NP and does not imply NP completeness of the problem.

One possibility for investigating fragments of  $\mathbf{NL} \diamond_{\mathcal{R}}$  which are inherently in NP would be to provide specialized versions of the contraction criterion, along the lines of the contraction criterion for the Lambek calculus.

We will not investigate this possibility here, but in the next section we will explore a PSPACE complete formulation of  $NL \diamond_{\mathcal{R}}$ .

## 9.2 Restricted Structural Rules

In the previous section we saw how providing a polynomial conversion algorithm would give us a logic which is in NP, or even NP complete if it has some form of commutativity. Table 8.1, however, gives the set of structural rules as an input to the algorithm, so it would be nice if we could see, just from the form of the structural rules what the computational complexity of the logic defined by these structural rules would be. As we will see in the next section, it might be that the logic is undecidable. In this section we will present a way of identifying logics which are decidable in PSPACE by simple, local properties of their structural rules.

A natural restriction on the structural rules is to require that the left hand side of a structural conversion has at least as many unary connectives as the right hand side. We will call these postulates *non-expanding* as, looking at these postulates from the perspective of forward chaining proof search, the structural rules cannot increase the number of symbols during the course of a derivation.

**Definition 9.1** *Given an antecedent configuration  $\Xi$ , we define the length of  $\Xi$  as follows.*

$$\begin{aligned} \text{length}(\Delta_1 \circ_i \Delta_2) &= \text{length}(\Delta_1) + \text{length}(\Delta_2) + 2 \\ \text{length}(\langle \Delta \rangle^i) &= \text{length}(\Delta) + 1 \\ \text{length}(\Delta) &= 0 \end{aligned}$$

We can extend Definition 9.1 to include arbitrary  $n$ -ary configurations by noting that the general form is the following.

$$\text{length}(*(\Delta_1, \dots, \Delta_n)) = \text{length}(\Delta_1) + \dots + \text{length}(\Delta_n) + n$$

**Definition 9.2** *The logic  $NL \diamond_{\mathcal{R}-}$  is the logic  $NL \diamond_{\mathcal{R}}$  where for every structural rule  $R \in \mathcal{R}$*

$$\frac{\Gamma[\Xi'[\Delta_1, \dots, \Delta_n]] \vdash C}{\Gamma[\Xi[\Delta_{\pi_1}, \dots, \Delta_{\pi_n}]] \vdash C} [R]$$

*the following holds.*

$$\text{length}(\Xi[\Delta_{\pi_1}, \dots, \Delta_{\pi_n}]) \leq \text{length}(\Xi'[\Delta_1, \dots, \Delta_n])$$

*We call structural rules with this property non-expanding.*

Note that because every structural rule of  $\mathcal{R}$  is linear, Definition 3.4 requires  $\Xi'$  to be non-empty, or, equivalently, that  $\text{length}(\Xi') > 0$ .

The logic  $\mathbf{NL}_{\diamond_{\mathcal{R}}}$  still gives us access to a wide variety of structural postulates, including postulates like  $[K]$

$$\frac{\Gamma[\langle \Delta_1 \rangle^1 \circ_0 \langle \Delta_2 \rangle^1] \vdash C}{\Gamma[\langle \Delta_1 \circ_0 \Delta_2 \rangle^1] \vdash C} [K]$$

even though its inverse  $[K']$  is not allowed.

$$\frac{\Gamma[\langle \Delta_1 \circ_0 \Delta_2 \rangle^1] \vdash C}{\Gamma[\langle \Delta_1 \rangle^1 \circ_0 \langle \Delta_2 \rangle^1] \vdash C} [K']$$

All the structural rules we have seen so far satisfy this condition, with the exception of the  $[4]$  postulate.

$$\frac{\Gamma[\langle \Delta \rangle^1] \vdash C}{\Gamma[\langle \langle \Delta \rangle^1 \rangle^1] \vdash C} [4]$$

However, the linguistic applications of this postulate appear to be limited. Moortgat (1997) even shows we can simulate the  $[T]$  and  $[4]$  postulates in  $\mathbf{NL}_{\diamond}$  without using structural rules.

Of the postulates which have been proposed in the literature, only two postulates proposed by Moortgat (1996a), repeated below, violate this condition because the conclusion of both rules contains an extra unary structural connective.

$$\frac{\Gamma[(\Delta_1 \circ_1 \Delta_2) \circ_0 \Delta_3] \vdash C}{\Gamma[\Delta_1 \circ_1 \langle \Delta_2 \circ_0 \Delta_3 \rangle^1] \vdash C} [P1'] \quad \frac{\Gamma[\Delta_2 \circ_0 (\Delta_1 \circ_1 \Delta_3)] \vdash C}{\Gamma[\Delta_1 \circ_1 \langle \Delta_2 \circ_0 \Delta_3 \rangle^r] \vdash C} [P2']$$

We will prove that the languages generated by  $\mathbf{NL}_{\diamond_{\mathcal{R}}}$  are equivalent to the context sensitive languages, which are defined as follows.

**Definition 9.3** A type 1 or context sensitive grammar  $G$  is a tuple  $\langle \Sigma, N, S, R \rangle$ , where

$\Sigma$  is the set of terminal symbols,

$N$  is the set of nonterminal symbols,

$S$  is a designated member of  $N$  called the start symbol,

$R$  is the set of grammar rules.

We require that  $N$  and  $\Sigma$  are disjoint. The set  $R$  consists of rewrite rules  $\Gamma \rightarrow \Delta$ , where  $\Gamma$  and  $\Delta$  are lists of symbols from  $N \cup \Sigma$ . We restrict  $R$  to rules where  $\Gamma$  contains at least one nonterminal symbol and where  $\text{length}(\Gamma) \leq \text{length}(\Delta)$ . Specifically, this excludes rules of the form  $\Gamma \rightarrow \epsilon$ , where  $\epsilon$  is the empty list.

A type 1 or context sensitive language  $\mathcal{L}_G$  is the set of lists of terminal symbols obtained by taking a type 1 grammar  $G$  and computing the closure of the start symbol  $S$  under the operation of the rules  $R$ .

**Example 9.4** Context sensitive grammars are a quite expressive formalism. It is possible, for example, to formulate a grammar generating the following language.

$$\{a^n \mid n \text{ is prime}\}$$

A more simple example is the grammar which generates the following language.

$$\{a^n b^n c^n \mid n > 0\}$$

This grammar looks as follows:  $\langle \{a, b, c\}, \{S, B, C\}, S, R \rangle$  where  $R$  is the following set of rules.

$$\begin{aligned} S &\rightarrow_1 aSBC \\ S &\rightarrow_2 aBC \\ CB &\rightarrow_3 BC \\ aB &\rightarrow_4 ab \\ bB &\rightarrow_5 bb \\ bC &\rightarrow_6 bc \\ cC &\rightarrow_7 cc \end{aligned}$$

This grammar generates the language  $a^n b^n c^n$  for all  $n > 0$ . We can generate the string  $aabbcc$ , for example, as follows.

$$\begin{aligned} S &\rightarrow_1 \\ aSBC &\rightarrow_2 \\ aaBCBC &\rightarrow_3 \\ aaBBCC &\rightarrow_4 \\ aabBCC &\rightarrow_5 \\ aabbCC &\rightarrow_6 \\ aabbcC &\rightarrow_7 \\ aabbcc & \end{aligned}$$

To facilitate later proofs, we introduce the notion of lexicalization for phrase structure grammars. We formulate the definitions and lemma's in such a way that the don't use the context sensitive restriction on the rules of the grammar, so that we can reuse these results in Section 9.3, where we will apply the same definitions to unrestricted phrase structure grammars.

**Definition 9.5** We say a phrase structure grammar is lexicalized if every rule is of one of the following forms.

- (i)  $\Gamma \rightarrow \Delta$  where  $\Gamma$  and  $\Delta$  are lists of nonterminal symbols
- (ii)  $A \rightarrow \beta$  where  $A \in N$  and  $\beta \in \Sigma$ .

We call the rules of form (i) the grammar rules and the rules of form (ii) the lexicalization rules.

**Lemma 9.6** For lexicalized grammars we can restrict ourselves to derivations where all applications of grammar rules precede the applications of lexicalization rules.

**Proof** We prove the lemma by induction on the number of lexicalization rules which occur before grammar rules. Let  $\mathcal{D}$  be a derivation of a lexicalized grammar and  $A \rightarrow \beta$  be a lexicalization rule which occurs before some grammar rules, that is, we are in the following situation

$$S \rightarrow_a \Gamma[A] \rightarrow \Gamma[\beta] \rightarrow_b \Delta[\beta]$$

where at least one of the rules in  $\rightarrow_b$  is a grammar rule. Because of the restrictions on the rules, none of the rules in  $\rightarrow_b$  can rewrite  $\beta$ , so it is easy to see the following derivation is valid too.

$$S \rightarrow_a \Gamma[A] \rightarrow_b \Delta[A] \rightarrow \Delta[\beta]$$

Because this derivation has one less lexicalization rule occurring before grammar rules we can apply the induction hypothesis giving us a derivation of the required form.  $\square$

**Lemma 9.7** *For every phrase structure grammar  $G$  there is a lexicalized grammar  $G'$  which generates the same language.*

**Proof** Take any rule  $\rho$  of the form  $\Gamma \rightarrow \Delta$  from grammar  $G$  which is not one of the forms of Definition 9.5.

Let  $\beta_1, \dots, \beta_n$  be the terminal symbols occurring in  $\rho$ . Replace all occurrences of  $\beta_i$  in the grammar by a new nonterminal symbol  $N_i$ . This replacement does not introduce any new non-lexicalized rules: any rule previously corresponding to Definition 9.5.(ii) now corresponds to Definition 9.5.(i) and rules corresponding to Definition 9.5.(i) are unaffected. Finally, add a rule  $N_i \rightarrow \beta_i$  to the grammar for every  $i$ . All these new rules correspond to Definition 9.5.(ii) and rule  $\rho$  now corresponds to Definition 9.5.(i). Call this new grammar  $G'$ .

It is clear that in  $G'$  the nonterminal symbols  $N_1, \dots, N_n$  play the role of the terminal symbols  $\beta_1, \dots, \beta_n$ , that is, any derivation which generates a terminal symbol  $\beta_i$  by means of a rule in  $G$  which is not a lexicalization rule, generates a nonterminal symbol  $N_i$  in  $G'$ . A separate lexicalization rule then rewrites  $N_i$  to  $\beta_i$  in  $G'$ . Conversely, any rule in  $G'$  which produces a nonterminal symbol  $N_i$  which is not a nonterminal symbol of  $G$  produces a terminal symbol  $\beta_i$  in  $G$ .  $\square$

**Example 9.8** *The context sensitive grammar from Example 9.4 is not lexicalized. Only rule 3 is a valid grammar rule in a lexicalized context sensitive grammar. But replacing  $a$  by  $A$ ,  $b$  by  $D$  and  $c$  by  $E$  and adding the corresponding lexicalization rules produces a grammar in the form shown below, which is a lexicalized context sensitive grammar.*

$S \rightarrow_1 ASBC$   
 $S \rightarrow_2 ABC$   
 $CB \rightarrow_3 BC$   
 $AB \rightarrow_4 AD$   
 $DB \rightarrow_5 DD$   
 $DC \rightarrow_6 DE$   
 $EC \rightarrow_7 EE$   
 $A \rightarrow_8 a$   
 $D \rightarrow_9 b$   
 $E \rightarrow_{10} c$

The derivation of Example 9.4 proceeds as shown below.

$S \rightarrow_1$   
 $ASBC \rightarrow_2$   
 $AABCBC \rightarrow_3$   
 $AABBCC \rightarrow_4$   
 $AADBCC \rightarrow_5$   
 $AADDCC \rightarrow_6$   
 $AADDEC \rightarrow_7$   
 $AADDEE \rightarrow_8$   
 $aADDEE \rightarrow_8$   
 $aaDDEE \rightarrow_9$   
 $aabDEE \rightarrow_9$   
 $aabbEE \rightarrow_{10}$   
 $aabbcE \rightarrow_{10}$   
 $aabbc$

**Definition 9.9** From a lexicalized context sensitive grammar  $G$  we generate the corresponding multimodal Lambek calculus  $\mathcal{M}(G)$  as follows.  $\mathcal{M}(G)$  has one unary mode for every nonterminal of  $G$  and a single binary mode.

Every lexicalization rule  $A \rightarrow \beta$  corresponds to a lexical entry as follows.

$$\text{lex}(\beta) = a \setminus \square_A^\perp a$$

The goal formula of  $\mathcal{M}(G)$  is  $a \setminus \square_S^\perp a$  where  $S$  is the mode corresponding to the start symbol of  $G$ .

$\mathcal{M}(G)$  has a structural rule for every grammar rule  $A_1 \dots A_n \rightarrow_{R1} B_1 \dots B_m$  of  $G$ .

$$\frac{\Gamma[\langle \dots \langle \Delta \rangle^{B_m} \dots \rangle^{B_1}] \vdash C}{\Gamma[\langle \dots \langle \Delta \rangle^{A_n} \dots \rangle^{A_1}] \vdash C} [R1]$$

Because  $m > 0$  and  $n \leq m$ , this is a valid  $\mathbf{NL}\diamond_{\mathcal{R}-}$  rule.

Furthermore, the structural rule component of  $\mathcal{M}(G)$  contains one of the structural rules for associativity for the single binary mode

$$\frac{\Gamma[(\Delta_1 \circ \Delta_2) \circ \Delta_3] \vdash C}{\Gamma[\Delta_1 \circ (\Delta_2 \circ \Delta_3)] \vdash C} [\text{Ass2}]$$

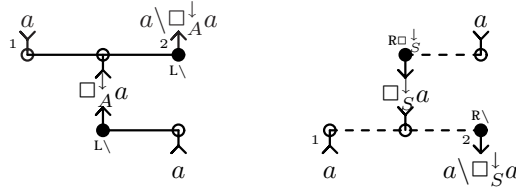


Figure 9.1: Proof structures for the lexicon and the goal formula

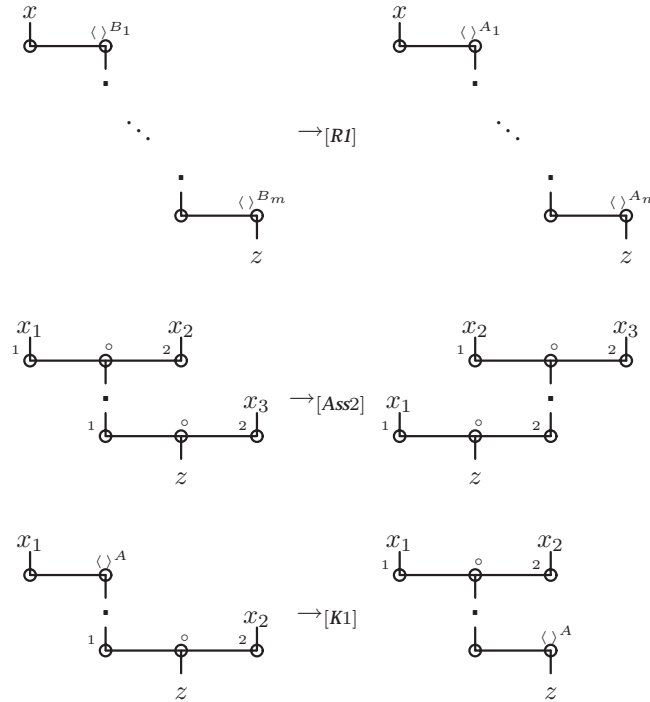


Figure 9.2: Structural conversions for  $\mathcal{M}(G)$

and the structural rule of [K1] for every mode  $A \in N$ .

$$\frac{\Gamma[\langle \Delta_1 \rangle^A \circ \Delta_2] \vdash C}{\Gamma[\langle \Delta_1 \circ \Delta_2 \rangle^A] \vdash C} \text{ [K1]}$$

Seeing this grammar from the proof structure perspective, all lexical entries are of the form shown in Figure 9.1 on the left, where  $A$  is a unary index of  $\mathcal{M}(G)$ . The goal proof structure is shown in Figure 9.1 on the right. Figure 9.2 gives a schematic representation of the structural conversions.

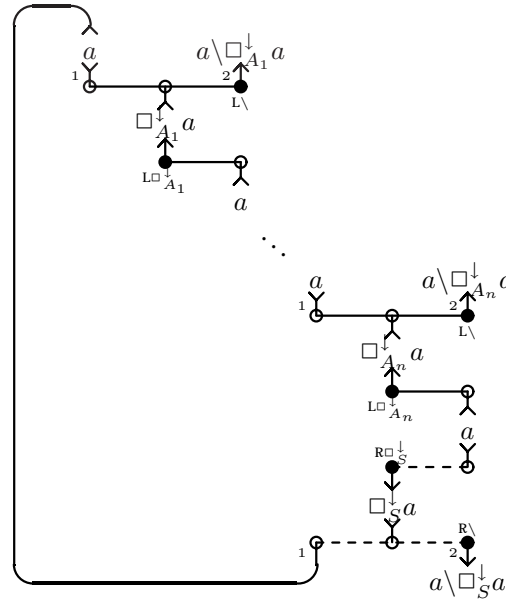


Figure 9.3: Proof structure for a derivation in  $\mathcal{M}(G)$

**Lemma 9.10** *Let  $\mathcal{M}(G)$  be a multimodal Lambek calculus which corresponds to a context sensitive grammar  $G$  according to Definition 9.9. For every sequence of hypotheses  $H_1, \dots, H_n$  which correspond to lexical entries of  $\mathcal{M}(G)$  only the proof structure  $\mathcal{S}$ , which is shown schematically in Figure 9.3, can convert to a hypothesis tree of  $H_1, \dots, H_n \vdash a \sqcup_S^\perp a$ .*

**Proof** Look at the structural rules of  $\mathcal{M}(G)$ : none of them change the order of the hypotheses, so we need to consider only those proof structures where the hypotheses are in the right order with respect to each other. All hypotheses induce proof structures as shown in Figure 9.1 on the left, whereas the goal formula induces the proof structure shown in Figure 9.1 on the right.

Suppose we connect the  $a$  hypothesis of the goal proof structure to any formula other than the  $a$  conclusion of the rightmost lexical proof structure. Linking it to its own conclusion produces a disconnected proof structure, whereas linking it to a different lexical proof structure will make the formula corresponding to that proof structure appear as the rightmost hypothesis. We follow this line of reasoning inductively for the next lexical proof structures until after the last lexical proof structure has been connected to its  $a$  conclusion, we connect its  $a$  hypothesis to the  $a$  conclusion of the goal proof structure, producing the proof structure pictured in Figure 9.3.  $\square$

**Definition 9.11** *We define a function  $f_1$  which translates a component of an abstract proof structure into a list of nonterminal symbols as follows. ‘||’ is the con-*

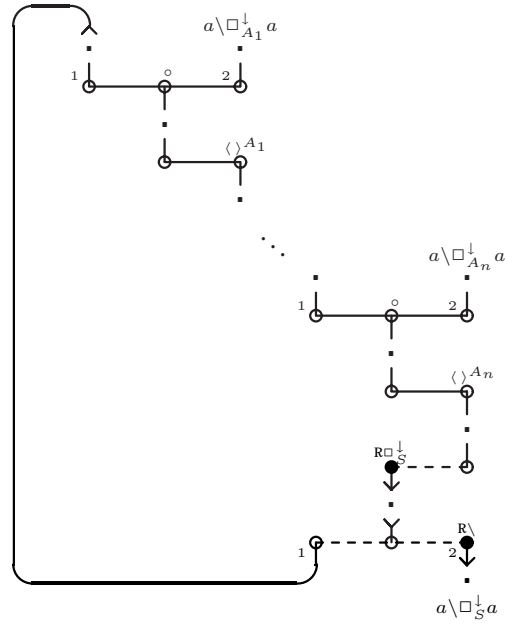


Figure 9.4: Abstract proof structure for a derivation in  $\mathcal{M}(G)$

*catenation operator.*

$$\begin{aligned} f_1(F) &= \epsilon \\ f_1(\langle \Gamma \rangle^i) &= f_1(\Gamma) \parallel i \\ f_1(\Gamma_1 \circ \Gamma_2) &= f_1(\Gamma_1) \parallel f_1(\Gamma_2) \end{aligned}$$

**Lemma 9.12** *Let  $\mathcal{M}(G)$  be a multimodal Lambek calculus which corresponds to a context sensitive grammar  $G$  according to Definition 9.9. For every conversion sequence  $\rho$  of a proof net for  $\mathcal{M}(G)$ , applying function  $f_1$  to the unique active component  $\mathcal{C}$  of the successive abstract proof structures of  $\rho$  before the two contractions of  $\rho$  and removing all identity transitions yields a derivation  $\mathcal{D}$  of  $G$ .*

**Proof** Because  $\rho$  ends in a hypothesis tree, we can apply Lemma 9.10, and we know  $\mathcal{S}$  has to be of the form shown in Figure 9.3 on the preceding page. Converting that proof structure into an abstract proof structure produces the result shown in Figure 9.4.

Because we have only one active component and a single active consecutive par link in it, we can apply Lemma 8.9 to ensure the two contractions can always be performed after any structural conversions in this component.

We will generate a derivation  $\mathcal{D}$  of  $\Gamma$  from  $\rho$  as follows. For the active component  $\mathcal{C}$  of the initial abstract proof structure  $f_1(\mathcal{C}) = A_1 \dots A_n$ . We extend this by applying the  $n$  appropriate lexicalization rules of  $G$  to produce a valid suffix of a derivation of  $w_1 \dots w_n$  in  $G$ .

We now proceed by induction on the length  $l$  of the sequence of the structural conversions  $\rho$ , simultaneously proving that any time we compute  $f_1(\Gamma_2)$  for the final clause of Definition 9.11 the result will be the empty list. For the abstract proof structure of Figure 9.4 this property holds, because the right branch of every  $[\circ]$  link is a leaf.

If  $l = 0$  then, because the consecutive ' $\square_S^\perp$ ' par link can be contracted  $\mathcal{C} = \langle a \circ \Gamma \rangle^S$ , and because we have proved by induction that  $f_1(\Gamma) = \epsilon$  this means  $f_1(\langle a \circ \Gamma \rangle^S)$  is  $S$ . Therefore, we have produced a valid derivation in  $G$ .

If  $l > 0$  we look at the last conversion.

If it is an [Ass2] conversion, we keep the suffix of the derivation in  $G$  the same.  $f_1(\Gamma_1 \circ (\Gamma_2 \circ \Gamma_3)) = f_1((\Gamma_1 \circ \Gamma_2) \circ \Gamma_3) = f_1(\Gamma_1) \| f_1(\Gamma_2) \| f_1(\Gamma_3)$  and, given that  $f_1(\Gamma_2 \circ \Gamma_3)$  is  $\epsilon$ ,  $f_1(\Gamma_2)$  and  $f_1(\Gamma_3)$  are both  $\epsilon$ .

If the last conversion is a [K1] conversions, we keep the suffix of the  $G$  derivation the same.  $f_1(\langle \Gamma_1 \rangle^A \circ \Gamma_2) = f_1(\Gamma_1) \| A \| f_1(\Gamma_2)$  whereas  $f_1(\langle \Gamma_1 \circ \Gamma_2 \rangle^A) = f_1(\Gamma_1) \| f_1(\Gamma_2) \| A$ , but since we know  $f_1(\Gamma_2) = \epsilon$  both are equivalent.

If the last conversion is a grammatical conversion, we prefix the corresponding grammatical rule to the derivation of  $G$ . The result is again a valid suffix of a derivation in  $G$ .  $\square$

**Lemma 9.13** *Given a lexicalized context sensitive grammar  $G$ , the multimodal Lambek calculus  $\mathcal{M}(G)$  corresponding to it according to Definition 9.9 generates the same language.*

**Proof** We need to show that the multimodal Lambek calculus  $\mathcal{M}(G)$  corresponding to  $G$  by Definition 9.9 generates the same language as  $G$ , that is that there exists a derivation  $\delta$  of string  $s$  in  $G$  iff there exists a derivation  $\rho$  of  $s$  in  $\mathcal{M}(G)$ .

$[\Rightarrow]$  Let  $\delta$  be a derivation in  $G$ . Lemma 9.6 ensures there is a derivation  $\delta'$  generating the same string where every grammar rule precedes every lexicalization rule. We construct the proof structure  $\mathcal{S}$  for the proof net derivation corresponding to  $\delta$  by looking at the lexicalization steps of  $\delta'$  and selecting the appropriate lexical entries corresponding to it according to the translation, then combining the lexical proof structures and the goal proof structure according to Lemma 9.10.

The abstract proof structure  $\mathcal{A}$  corresponding to  $\mathcal{S}$ , which looks as shown in Figure 9.4, can be converted as follows: for every grammatical rule of  $\delta'$  we apply the conversion corresponding to it according to Definition 9.9 to  $\mathcal{A}$ . We start by combining all  $[\langle \rangle^i]$  links using [K1] and [Ass2] rules alternately until we have a single sequence of  $[\langle \rangle^i]$  links which, from hypotheses to conclusion, corresponds to the sequence of terminals just after the last grammatical rule of  $\delta'$ . Now for every grammatical rule of  $\delta'$ , starting with the last one, we apply the corresponding structural rule. At every step, the sequence of  $[\langle \rangle^i]$  links will have modes  $B_1, \dots, B_m$  corresponding to the sequence of nonterminals of  $\delta'$ , until, before the first grammatical rule, it will contain a single

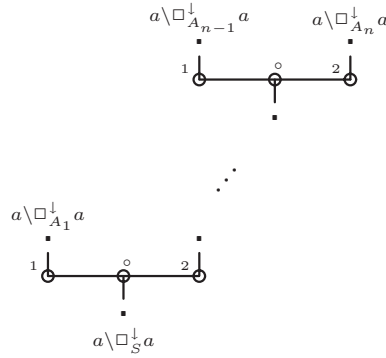


Figure 9.5: The final hypothesis tree of a  $\mathcal{M}(G)$  proof net

$[(\rangle^S]$  link, which we can contract using the  $[R\square_S^{\perp}]$  contraction. After the  $[R\setminus]$  contraction the hypothesis tree will look as shown in Figure 9.5.

$[\Leftarrow]$  This is a direct consequence of Lemma 9.12 which shows how to generate a derivation of  $G$  from a conversion sequence of a proof net in  $\mathcal{M}(G)$ .  $\square$

**Lemma 9.14** *A nondeterministic linear bounded Turing machine can encode sequent proof search for any non-expanding multimodal Lambek calculus.*

**Proof** First, we replace any word which has multiple lexical assignments  $A_1, \dots, A_m$  by a single lexical assignment  $A_1 \& \dots \& A_m$  and multiple possible goal formulas  $G_1, \dots, G_n$  by a single goal formula  $G_1 \oplus \dots \oplus G_n$ , adding the following logical ruled to the calculus.

$$\frac{\Gamma[A_i] \vdash C}{\Gamma[A_0 \& A_1] \vdash C} [L\&] \quad \frac{\Gamma \vdash C_i}{\Gamma \vdash C_0 \oplus C_1} [R\oplus]$$

Note that these are just the  $[L\&]$  and the  $[R\oplus]$  rules from linear logic adapted to  $\mathbf{NL}_{\diamond \mathcal{R}}$ . This addition to the logic is quite inessential and serves only as a way to generate the right amount of space on the input tape of the Turing machine in the case of multiple lexical assignments to a single word or multiple possible goal formulas.

The input tape of the Turing machine for the sequence of words  $w_1 \dots w_n$  consists of the lexical formulas  $f_1 \dots f_n$  corresponding to these words and the goal formula  $g$ , in Polish prefix notation, the different antecedent formulas separated by an arbitrary binary mode  $\circ_0$

$$\vdash \circ_0 f_1 \circ_0 \dots \circ_0 f_n g$$

Now look at the maximum amount of space required for forward chaining proof search for the sequent calculus compared to the size of the input

tape. Forward chaining proof search amounts to starting with the axioms and applying the sequent rules from premisses to conclusion performing all computations nondeterministically and halting with success when we have generated the formulas on the input tape in the right order and with failure when we have exhausted the search space.

Because of the subformula property, the sequents we have to consider only contain subformulas of the end-sequent. We also need a binary sequent separator symbol, for which we use ‘ $\bullet$ ’ to separate the different sequent proofs.

In Polish prefix notation, the sequent rules of  $[R\Diamond_j]$  and  $[R\bullet_i]$ , normally portrayed as follows

$$\frac{\Gamma \vdash C}{\langle \Gamma \rangle^j \vdash \Diamond_j C} [R\Diamond_j] \quad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma \circ_i \Delta \vdash A \bullet_i B} [R\bullet_i]$$

look like shown below.

$$\frac{\vdash \Gamma C}{\vdash \langle \rangle^j \Gamma \Diamond_j C} \quad \frac{\Delta \vdash \Gamma A \vdash \Delta B}{\vdash \circ_i \Gamma \Delta \bullet_i A B}$$

Counting the number of symbols in the premisses and the conclusion, we see that these rules, like the other tensor rules, increase the number of symbols, whereas the par rules keep the number of symbols constant. Furthermore, because all structural rules are non-expanding they may reduce the total amount of space needed, but they can never enlarge it.

We examine the tensor rules more closely. For the  $[R\Diamond_j]$  and the  $[L\Box_j^\perp]$  rule the conclusion has four symbols more than the premiss (we count  $\langle \rangle$  and  $\Box^\perp$  as one symbol), for the  $[R\bullet_i]$ ,  $[L/i]$  and  $[L\setminus_i]$  rule there are two more symbols in the conclusion. Of these extra symbols, two are included on the input tape, which means that only for the  $[R\Diamond_j]$  and the  $[L\Box_j^\perp]$  we have to add two extra spaces on the tape. This means that the maximum amount of space we need is the length of the input tape  $l$  plus two extra symbols for every positive occurrence of  $\Diamond_j$  and two extra symbols for every negative occurrence of  $\Box_j^\perp$ , which is strictly less than  $2l$ .  $\square$

**Lemma 9.15 (Koroda (1964))** *Any linear bounded Turing machine which does not generate the empty string is equivalent to a context sensitive grammar.*

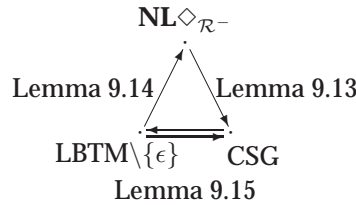


Figure 9.6: A summary of the previous lemma's

**Theorem 9.16** *The parsing problem for  $\mathbf{NL}\diamond_{\mathcal{R}^-}$  is equivalent to the parsing problem for context sensitive grammars.*

**Proof** The theorem is an immediate consequence of Lemmas 9.13, 9.14 and 9.15, as shown in Figure 9.6 on the page before. We can decide the derivability of any sequent of  $\mathbf{NL}\diamond_{\mathcal{R}^-}$  in nondeterministic linear space according to Lemma 9.14 and any problem which can be solved in nondeterministic linear space can be solved in  $\mathbf{NL}\diamond_{\mathcal{R}^-}$   $\square$

**Corollary 9.17** *The decision problem for  $\mathbf{NL}\diamond_{\mathcal{R}^-}$  is PSPACE complete.*

**Proof** Karp (1972) shows that the decision problem for context sensitive grammars, that is, given a description of a context sensitive grammar  $G$  and a string  $s$  we return 1 if  $s \in \mathcal{L}_G$  and 0 otherwise, is PSPACE complete.

Given Theorem 9.16 the PSPACE completeness of the decision problem for  $\mathbf{NL}\diamond_{\mathcal{R}^-}$  follows immediately. It is PSPACE hard because we can encode a context sensitive grammar in the structural rule component of  $\mathbf{NL}\diamond_{\mathcal{R}^-}$ . It is in PSPACE because we can translate any fragment of  $\mathbf{NL}\diamond_{\mathcal{R}^-}$ , including the structural rules, to a context sensitive grammar. Therefore it is PSPACE complete.  $\square$

### 9.3 $\mathbf{NL}\diamond_{\mathcal{R}}$

Carpenter (1995) showed that the multimodal Lambek calculus is undecidable if we allow structural rules which duplicate and erase material. A similar result is mentioned by Lincoln, Mitchell, Scedrov & Shankar (1990), where the relation between semi-Thue systems and cyclic linear logic with exponentials is sketched.

In this section I will prove that even if we restrict the structural rules to be linear according to Definition 3.4 the logic  $\mathbf{NL}\diamond_{\mathcal{R}}$  is still undecidable. We show this by using the structural rules for the unary connectives to encode the recognition problem for type 0 languages, which is known to be undecidable (Chomsky 1959). This result is mostly symmetric to the results of the previous section, since, by the single restriction on the form of the rules for a type 0 grammar, we will now give a translation encoding top down search for type 0 grammar opposed to the bottom up search of the previous translation.

One of the consequences of the results of this section is therefore that we could have also defined the logic  $\mathbf{NL}\diamond_{\mathcal{R}^+}$  which has only non-shrinking structural rules and which also generates the context sensitive languages.

**Definition 9.18** *A type 0 grammar  $G$  is a tuple  $\langle \Sigma, N, S, R \rangle$ , where*

*$\Sigma$  is the set of terminal symbols,*

*$N$  is the set of nonterminal symbols,*

$S$  is designated member of  $N$  called the start symbol,

$R$  is the set of grammar rules.

As usual, we require  $N$  and  $\Sigma$  to be disjoint. The set  $R$  consists of rewrite rules  $\Gamma \rightarrow \Delta$ , where  $\Gamma$  and  $\Delta$  are lists of symbols from  $N \cup \Sigma$ . The only restriction on the form of the rules is that  $\Gamma$  contains at least one nonterminal symbol.

A type 0 language  $\mathcal{L}_G$  is the set of lists of terminal symbols obtained by taking a type 0 grammar  $G$  and computing the closure of the start symbol  $S$  under the operation of the rules  $R$ .

Since Definition 9.5 and Lemma 9.7 were formulated for general phrase structure grammars, we can apply them here as well to talk about lexicalized type 0 grammars and know that any type 0 grammar  $G$  generates the same language as the corresponding lexicalized type 0 grammar  $G'$ .

Because our formulation of the Lambek calculus does not allow for empty antecedent derivations, we also need to restrict ourselves to type 0 grammars which do not derive the empty string. However, because determining whether a given type 0 grammar generates the empty string is known to be undecidable, we will propose a simple transformation on type 0 grammars which ensures the strings generated by the grammar are always non-empty.

**Definition 9.19** A type 0 grammar  $G$  is  $\epsilon$  free if  $\mathcal{L}_G$  does not include the empty string.

From a type 0 grammar  $G$  we can create the  $\epsilon$  free version  $G'$  by adding two new nonterminal symbols  $S'$  and  $F$  and a single new terminal symbol  $'.'$  to it.  $S'$  will be the start symbol of  $G'$  and the rules of  $G'$  are exactly those of  $G$  with the addition of the following two rules, where  $S$  is the start symbol of  $G$ .

$$\begin{aligned} S' &\rightarrow SF \\ F &\rightarrow . \end{aligned}$$

It is easy to see that  $S \rightarrow \Gamma$  in  $G$  iff  $S' \rightarrow \Gamma$  in  $G'$  because  $S'$ ,  $F$  and  $'.'$  don't appear in  $G$ .

Note that the  $\epsilon$  free version of a grammar  $G$  is defined in such a way that if  $G$  is lexicalized then  $G'$  is lexicalized as well.

**Definition 9.20** From a  $\epsilon$  free, lexicalized type 0 grammar  $G$  we generate a multimodal Lambek calculus  $\mathcal{M}_0(G)$  as in Definition 9.9.  $\mathcal{M}_0(G)$  has one unary mode for every nonterminal of  $G$  and a single binary mode.

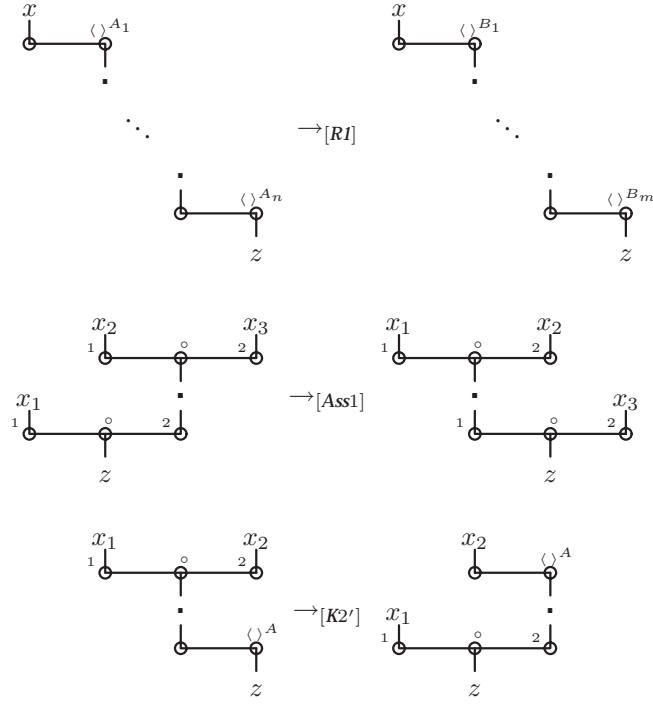
Every lexicalization rule  $A \rightarrow \beta$  corresponds to a lexical entry as follows.

$$\text{lex}(\beta) = \diamond_A a/a$$

The goal formula of  $\mathcal{M}_0(G)$  is  $\diamond_S a/a$

$\mathcal{M}_0(G)$  has a structural rule for every grammar rule  $A_1 \dots A_n \rightarrow_{R1} B_1 \dots B_m$  of  $G$ .



Figure 9.8: Structural conversions for  $\mathcal{M}_0(G)$ 

$$\begin{aligned}
 f_0(F) &= \beta \\
 f_0(\langle \Gamma \rangle^i) &= f_0(\Gamma) \parallel i \\
 f_0(\Gamma_1 \circ \Gamma_2) &= f_0(\Gamma_1) \parallel f_0(\Gamma_2)
 \end{aligned}$$

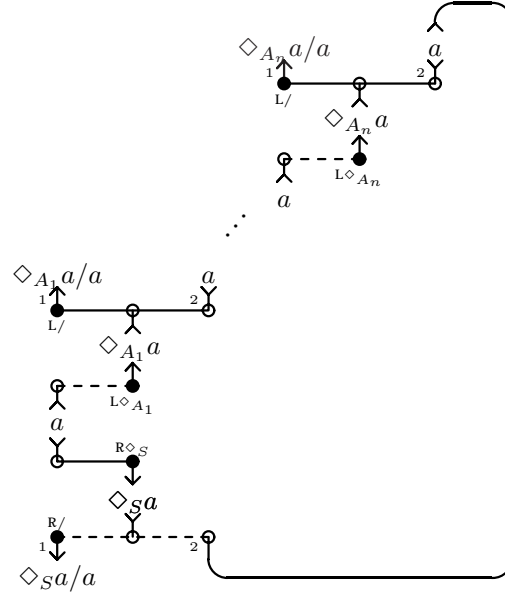
**Lemma 9.23** For every conversion sequence  $\rho$  for an abstract proof structure generated from  $\mathcal{M}_0(G)$ , applying function  $f_0$  of Definition 9.22 to the unique active component of the successive abstract proof structures of  $\rho$  and removing all identity transitions yields a derivation  $\mathcal{D}$  of  $G$ .

**Proof** We generate  $\mathcal{D}$  by induction on the length  $l$  of  $\rho$ , simultaneously proving that for every component whenever we compute  $f_0(\Gamma_1)$  for the final clause of Definition 9.22 the result will be a non-empty list of terminal symbols.

For the active component  $\mathcal{C}$  of the abstract proof structure shown in Figure 9.10,  $f_0(\mathcal{I}) = S$  which is a valid prefix for a derivation  $\mathcal{D}$ .

Given that the previous  $n$  steps of  $\rho$  produced a prefix of a derivation  $\mathcal{D}$  we extend this derivation as follows, depending on the next conversion  $\mathcal{C}_i \xrightarrow{c} \mathcal{C}_{i+1}$ .

If  $c$  is a grammatical conversion, then we extend  $\mathcal{D}$  with the corresponding grammatical rule of  $G$ .

Figure 9.9: Proof structure for a derivation in  $\mathcal{M}_0(G)$ 

If  $c$  is a  $[K2']$  conversion we need to show  $f_0(\langle \Gamma_1 \circ \Gamma_2 \rangle^A) = f_0(\Gamma_1 \circ \langle \Gamma_2 \rangle^A)$ . Because the concatenation operation is associative, both components translate to the same expression:  $f_0(\Gamma_1) \parallel f_0(\Gamma_2) \parallel A$ . We keep  $\mathcal{D}$  unchanged.

If  $c$  is an  $[Ass1]$  conversion, we do not extend  $\mathcal{D}$ . The associativity of the concatenation operation again preserves the translation of the two components.  $f_0(\Gamma_1 \circ (\Gamma_2 \circ \Gamma_3)) = f_0((\Gamma_1 \circ \Gamma_2) \circ \Gamma_3) = f_0(\Gamma_1) \parallel f_0(\Gamma_2) \parallel f_0(\Gamma_3)$ . Given that the first expression satisfies our invariant, that is, both  $f_0(\Gamma_1)$  and  $f_0(\Gamma_2)$  are non-empty lists of terminal symbols,  $f_0(\Gamma_1 \circ \Gamma_2)$  is the concatenation of these lists and is therefore also a non-empty list of terminal symbols.

If  $c$  is a  $[L_{\diamond A}]$  contraction, it will remove the  $[L_{\diamond A}]$  and  $[\langle \rangle^A]$  links, connecting the active component to the  $[\circ]$  link above it. This will replace the first nonterminal symbol of  $f_0(\mathcal{I})$  with the corresponding terminal symbol according to a lexicalization rule of  $G$ , with which we extend the derivation  $\mathcal{D}$ . By construction,  $f_0(\Gamma_1)$  for the new binary link produces a list consisting of a single terminal symbol.

The final conversion step is always the  $[R/]$  contraction. Because the shape of the  $[R/]$  redex requires the rest of the abstract proof structure to be on the left branch of the  $[\circ]$ . Since we have just shown that the  $f_0(\Gamma_1)$  produces a non-empty sequence of terminals for every left branch of a  $[\circ]$  link, this means we have successfully completed our derivation in  $G$ .  $\square$

**Lemma 9.24** *Given a  $\epsilon$  free, lexicalized type 0 grammar  $G$  the fragment  $\mathcal{M}_0(G)$  of  $NL_{\diamond \mathcal{R}}$  which corresponds to it according to Definition 9.20 generates the same*

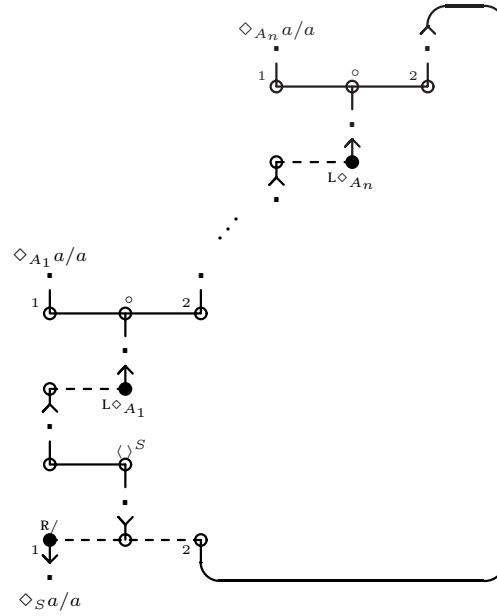


Figure 9.10: Abstract proof structure for a derivation in  $\mathcal{M}_0(G)$

language.

**Proof** We show that for every derivation of  $G$  producing sentence  $s$  we can construct a derivation in  $\mathcal{M}_0(G)$  producing the same sentence and vice versa.

[ $\Rightarrow$ ] Given a derivation  $\mathcal{D}$  of a grammar  $G$ , we look at the lexicalization rules of  $\mathcal{D}$  to produce the right lexical entries for the proof structure, which according to Lemma 9.21 has the form shown in Figure 9.9 and for which the corresponding abstract proof structure looks as shown in Figure 9.10. For every grammar rule of  $\mathcal{D}$ , we apply the corresponding structural conversion to the abstract proof structure until, after the last grammar rule the single active component of the abstract proof structure contains modes  $A_1, \dots, A_n$  from top to bottom. We then apply the  $[L \diamond_{A_1}]$  contraction, then using  $[Ass1]$  and  $[K2']$  conversions to move the successive unary modes up to the proper place for contraction, until, after the  $[L \diamond_{A_n}]$  we are in the position to perform the  $[R/]$  contraction, producing a hypothesis tree as required.

[ $\Leftarrow$ ] This is an immediate consequence of Lemma 9.23. □

**Theorem 9.25**  $NL \diamond_{\mathcal{R}}$  in Turing complete.

**Proof** Because Lemma 9.24 established the decision procedure for  $\text{NL}\diamond_{\mathcal{R}}$  is at least as hard as the parsing problem for type zero grammars, which is known to be Turing complete, we only have to give a procedure for enumerating derivations in  $\text{NL}\diamond_{\mathcal{R}}$ . We have already given many of those, see for example the proof net algorithm of Table 8.1. This establishes  $\text{NL}\diamond_{\mathcal{R}}$  is Turing complete.  $\square$

## 9.4 Conclusions

We have seen how  $\text{NL}\diamond_{\mathcal{R}}$ , with no other restriction on the structural rules than that they are all linear, is undecidable, but that  $\text{NL}\diamond_{\mathcal{R}^-}$  is PSPACE complete when given as its input a set structural rules which are non-expanding in addition to being linear. It appears this restriction gives us more than enough power to adequately describe syntactic phenomena.

An interesting open problem is if it is possible to give a characterization of packages of structural rules for which we can perform the conversions in polynomial time. This would give us a fragment for  $\text{NL}\diamond_{\mathcal{R}}$  which is in NP and it will be useful to compare the allowable structural rules for an NP fragment of  $\text{NL}\diamond_{\mathcal{R}}$  with the PSPACE fragment  $\text{NL}\diamond_{\mathcal{R}^-}$  to see what the trade-off would be.

Another interesting open question is how far we can extend the methods used for the known polynomial fragments of  $\text{NL}\diamond_{\mathcal{R}}$ , principally NL and formula-restricted versions of L, to more complex fragments.

Though PSPACE is a respectable complexity class for a logic, as a linguistic theory there are formalisms with much better complexity theoretic properties, though several others, such as HPSG (Pollard & Sag 1994), LFG (Kaplan & Bresnan 1982) or definite clause grammars (Tärnlund 1977) are only decidable in restricted cases. Also, PSPACE is only presented as an upper bound on the complexity of  $\text{NL}\diamond_{\mathcal{R}}$ , where we leave the structural rule component variable. Better complexity results may be obtained by using a fixed structural rule component.

In the next chapter, we will look at a polynomial grammar formalism, Lexicalized Tree Adjoining Grammars and relate it to a fragment of  $\text{NL}\diamond_{\mathcal{R}}$ , with a fixed set of structural rules and a restriction of the allowed formulas. Moreover, we will prove this correspondence is strong, allowing us to using the polynomial parsing strategies proposed for Lexicalized Tree Adjoining Grammars for this fragment.