Philippe de Groote
Glyn Morrill
Christian Retoré (Eds.)

Logical Aspects of Computational Linguistics

LNAI 2099

4th International Conference, LACL 2001
Le Croisic, France, June 2001
Proceedings

Springer

Philippe de Groote   Glyn Morrill
Christian Retoré (Eds.)

Logical Aspects of Computational Linguistics

4th International Conference, LACL 2001
Le Croisic, France, June 27-29, 2001
Proceedings

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saabrücken, Germany

Volume Editor

Philippe de Groote
INRIA Lorraine
615 rue du Jardin Botanique, B.P. 101
54602 Villers-lès-Nancy Cedex, France
E-mail: Philippe.deGroote@loria.fr

Glyn Morrill
Universitat Politècnica de Catalunya
Jordi Girona Salgado, 1-3, 08028 Barcelona, Spain
E-mail: morrill@lsi.upc.es

Christian Retoré
Institut de recherche en Informatique de Nantes (IRIN), Faculté des Sciences
2, rue de la Houssinière, B.P. 92208, 44322 Nantes Cedex 03, France
E-mail: retore@irisa.fr

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Logical aspects of computational linguistics : 4th international conference ;
proceedings / LACL 2001, LeCroisic, France, June 27 - 29, 2001. Philippe
de Groote ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong
Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2099 : Lecture notes in
artificial intelligence)
ISBN 3-540-42273-0

CR Subject Classification (1998): I.2, F.4.1

ISBN 3-540-42273-0 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

© Springer-Verlag Berlin Heidelberg 2001
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Boller Mediendesign
Printed on acid-free paper SPIN: 10839469 06/3142 5 4 3 2 1 0

Preface

This volume contains the proceedings of the 4th International Conference on Logical Aspects of Computational Linguistics, held June 27–29, 2001 in Le Croisic, France. The LACL conferences aim to provide a forum for the presentation and discussion of current research in all the formal and logical aspects of computational linguistics.

The program committee selected 16 papers from submissions of overall high quality. The papers cover a wide range of topics, including categorial grammars, dependency grammars, formal language theory, grammatical inference, hyperintensional semantics, minimalism, and type-logical semantics, by authors from Australia, Canada, Denmark, France, Germany, Italy, The Netherlands, Poland, Spain, Sweden, United Kingdom, and USA.

M. Moortgat (Universiteit Utrecht), G. K. Pullum (University of California, Santa Cruz), and M. Steedman (University of Edinburgh) presented invited talks, on "Structural Equations in Language Learning", "On the Distinction between Model-Theoretic and Generative-Enumerative Syntactic Frameworks", and "Reconciling Type-Logical and Combinatory Extensions of Categorial Grammar" respectively.

We would like to thank all the people who made this 4th LACL possible: the program committee, the external reviewers, the organization committee, and the LACL sponsors.

April 2001

Philippe de Groote
& Glyn Morrill

Organization

Program Committee

W. Buszkowski (Poznan)	M. Kanazawa (Tokyo)
R. Crouch, (Palo Alto)	G. Morrill, co-chair (Barcelona)
A. Dikovsky (Nantes)	R. Muskens (Tilburg)
M. Dymetman (Grenoble)	F. Pfenning (Pittsburgh)
C. Gardent (Nancy)	B. Rounds, (Ann Arbor)
Ph. de Groote, co-chair (Nancy)	E. Stabler (Los Angeles)

Organizing Committee

B. Daille (Nantes)	C. Piliere, publicity chair (Nancy)
A. Dikovsky (Nantes)	C. Retoré, chair (Rennes)
A. Foret (Rennes)	P. Sebillot (Rennes)
E. Lebret (Rennes)	

Additional Referees

J.-M. Andreoli	T. Holloway King	J. Marciniec
P. Blackburn	M. Kandulski	J.-Y. Marion
C. Brun	F. Lamarche	G. Perrier

此为试读，需要完整PDF请访问: www.ertongbook.com

Table of Contents

Structural Equations in Language Learning

Michael Moortgat

Utrecht Institute of Linguistics — OTS
Trans 10, 3512 JK Utrecht, The Netherlands
Michael.Moortgat@let.uu.nl

Abstract. In categorial systems with a fixed structural component, the learning problem comes down to finding the solution for a set of type-assignment equations. A hard-wired structural component is problematic if one want to address issues of structural variation. Our starting point is a type-logical architecture with separate modules for the logical and the structural components of the computational system. The logical component expresses *invariants* of grammatical composition; the structural component captures variation in the realization of the correspondence between form and meaning. Learning in this setting involves finding the solution to both the type-assignment equations and the structural equations of the language at hand. We develop a view on these two subtasks which pictures learning as a process moving through a two-stage cycle. In the first phase of the cycle, type assignments are computed statically from structures. In the second phase, the lexicon is enhanced with facilities for structural reasoning. These make it possible to dynamically relate structures during on-line computation, or to establish off-line lexical generalizations. We report on the initial experiments in [15] to apply this method in the context of the Spoken Dutch Corpus.
For the general type-logical background, we refer to [12]; §1 has a brief recap of some key features.

1 Constants and Variation

One can think of type-logical grammar as a functional programming language with some special purpose features to customize it for natural language processing tasks. Basic constructs are demonstrations of the form $\Gamma \vdash A$, stating that a structure Γ is a well-formed expression of type A. These statements are the outcome of a process of computation. Our programming language has a built-in vocabulary of logical constants to construct the type formulas over some set of atomic formulas in terms of the indexed unary and binary operations of (1a). Parallel to the formula language, we have the structure-building operations of (1b) with $(\cdot \circ_i \cdot)$ and $\langle\cdot\rangle^j$ as counterparts of $\bullet_i$ and $\Diamond_j$ respectively. The indices i and j are taken from given, finite sets I, J which we refer to as composition *modes*.

(1) *a.* $\mathsf{Typ} ::= \mathsf{Atom} \mid \Diamond_j\mathsf{Typ} \mid \Box_j\mathsf{Typ} \mid \mathsf{Typ} \bullet_i \mathsf{Typ} \mid \mathsf{Typ}/_i\mathsf{Typ} \mid \mathsf{Typ}\backslash_i\mathsf{Typ}$

b. $\mathsf{Struc} ::= \mathsf{Typ} \mid \langle\mathsf{Struc}\rangle^j \mid \mathsf{Struc} \circ_i \mathsf{Struc}$

© Springer-Verlag Berlin Heidelberg 2001

In presenting the rules of computation characterizing the derivability relation, we keep logical and structural aspects apart. The different composition modes all have the same *logical rules*. But we can key access to different *structural rules* by means of the mode distinctions.

Let us consider the logical component first. For each type-forming operation in (1a) there is a *constructor* rule (rule of use, assembly) and a *destructor* rule (rule of proof, disassembly). These rules can be presented in a number of equivalent ways: algebraically, in Gentzen or natural deduction format, or in a proof net presentation. The assembly/disassembly duality comes out particulary clearly in the algebraic presentation, where we have the *residuation laws* of (2). In the natural deduction format, these laws will turn up as Introduction/Elimination rules; in the Gentzen format as Left/Right introduction rules.

(2)
$$\Diamond_j A \vdash B \quad \text{iff} \quad A \vdash \Box_j B$$
$$A \vdash C/_i B \quad \text{iff} \quad A \bullet_i B \vdash C \quad \text{iff} \quad B \vdash A\backslash_i C$$

The composition of natural language *meaning* proceeds along the lines of the Curry-Howard interpretation of derivations, which reads off the meaning assembly from the logical inference steps that make up a computation. In this sense, the composition of meaning is *invariant* across languages — it is fully determined by the elimination/introduction rules for the grammatical constants. Languages show variation in the structural realization of the correspondence between form and meaning. Such variation is captured by the structural component of the computational system. Structural rules have the status of *non-logical axioms* (or postulates). The structural rules we consider in this paper are *linear transformations*:[1] they reassemble grammatical material, but they cannot duplicate or waste it. The familiar rules of Associativity and Commutativity in (3) can serve as illustrations. In a global form, these rules destroy essential grammatical information. But we will see in §2.2 how they can be tamed.

(3)
$$A \bullet B \vdash B \bullet A$$
$$(A \bullet B) \bullet C \dashv\vdash A \bullet (B \bullet C)$$

To obtain a type-logical grammar over a terminal vocabulary Σ, we have to specify a lexicon $\mathsf{Lex} \subseteq \Sigma \times \mathsf{Type}$, assigning each vocabulary item a finite number of types. A grammar, then, is a structure $G = (\mathsf{Lex}, \mathsf{Op})$, where Op is the union of the logical and the structural rules. Let $L(G, B)$ be the set of strings of type B generated by G, and let $\mathsf{Struc}(A_1, \ldots, A_n)$ be the set of structure trees with yield $A_1, \ldots, A_n$. For a string $\sigma = w_1 \cdot \ldots \cdot w_n \in \Sigma^+$, we say that $\sigma \in L(G, B)$ iff there are $A_1, \ldots, A_n$ and $\Gamma \in \mathsf{Struc}(A_1, \ldots, A_n)$ such that for $1 \leq i \leq n$, $\langle w_i, A_i \rangle \in \mathsf{Lex}$, and $\Gamma \vdash B$. To obtain $L(G)$, the language generated by the type-logical grammar G, we compute $L(G, B)$ for some fixed (finite set of) goal type(s)/start symbol(s) B.

[1] That is, we do not address multiple-use issues like parasitic gaps here, which might require (a controlled form of) Contraction.

2 Structural Reasoning in Learning

The modular treatment of logical and structural reasoning naturally suggests that we break up the learning problem in two subtasks. One task consists in finding appropriate categorization for the words in Σ as the elementary building blocks for meaning composition. This is essentially the problem of computing type-assignments as we know it from classical categorial learning theory. The second subtask addresses the question: What is the dynamic potential of words in syntax? Answering this question amounts to solving structural equations within a space set by universal grammar.

In tackling the second subtask, we will rely heavily on the unary constant $\Diamond$ and its residual $\Box$. As we have seen in §1, the binary product $\bullet$ captures the composition of grammatical parts, while the residual implications / and \ express incompleteness with respect to the composition relation. Extending the vocabulary with the unary constants $\Diamond, \Box$ substantially increases the analytical power of the categorial type language. This can be seen already in *the base logic* (i.e. the pure residuation logic, with empty structural module): the unary operators make it possible to refine type-assignments that would be overgenerating without the unary decoration. Moreover, in systems with a non-empty structural component, the unary operators can provide lexically anchored control over structural reasoning. We discuss these two aspects in turn.

In the base logic, the fundamental derivability pattern created by the unary operators is $\Diamond\Box A \vdash A \vdash \Box\Diamond A$. One can exploit this pattern to obtain the agreement configurations of (4).

$$(4)\quad \Diamond\Box A \bullet \begin{cases} \Diamond\Box A\backslash B \vdash B \\ A\backslash B \vdash B \\ \Box\Diamond A\backslash B \vdash B \end{cases} \quad A \bullet \begin{cases} \Diamond\Box A\backslash B \nvdash B \\ A\backslash B \vdash B \\ \Box\Diamond A\backslash B \vdash B \end{cases} \quad \Box\Diamond A \bullet \begin{cases} \Diamond\Box A\backslash B \nvdash B \\ A\backslash B \nvdash B \\ \Box\Diamond A\backslash B \vdash B \end{cases}$$

The treatment of polarity sensitive items in [2] illustrates this use of modal decoration. Consider the contrast between 'Nobody left yet' with the negative polarity item 'yet' and '*Somebody left yet'. The negative polarity trigger 'nobody' is assigned the type $s/(np\backslash\Box\Diamond s)$, whereas 'somebody' has the undecorated type $s/(np\backslash s)$. The negative polarity item 'yet' is typed as $\Box\Diamond s\backslash\Box\Diamond s$ — it requires a trigger such as 'nobody' to check the $\Box\Diamond$ decoration in its result subtype. In the base logic, we have $s/(np\backslash\Box\Diamond s) \vdash s/(np\backslash s)$, i.e. the $\Box\Diamond$ decoration on argument subtypes can be simplified away, allowing a derivation of e.g. 'Nobody left' where there is no polarity item to be checked. This strategy of unary decoration is extended in [3] to lexically enforce constraints on the scopal possibilities of generalized quantifier expressions such as discussed in [1].

For the use of unary type decoration to provide controlled access to structural reasoning, we can rely on the results of [11]. In that paper, we present embedding translations $\cdot^\natural$ from source logics $\mathcal{L}$ to target logics $\mathcal{L}'\Diamond$, in the sense that $A \vdash B$ is derivable in $\mathcal{L}$ iff $A^\natural \vdash B^\natural$ is derivable in $\mathcal{L}'\Diamond$. The translations $\cdot^\natural$ decorate the type assignments of the source logic with the unary operators $\Diamond, \Box$ in such a way that they licence access to restricted versions of the structural rules.

In the following sections, we use this modalization strategy to develop our two-stage view on the learning process. The first stage consists of learning from structures in the base logic. Because the base logic has no facilities for structural reasoning, the lexical ambiguity load in this phase soon becomes prohibitive. In the second stage, the lexicon is optimized by shifting to modalized $(\Diamond, \Box)$ type assignments. The modal decoration is designed in such a way that lexical ambiguity is reduced to derivational polymorphism. We will assume that the learning process cycles through these two stages. To carry out this program, a number of questions have to be answered:

- What kind of MODAL DECORATION do we envisage?
- What is the STRUCTURAL PACKAGE which delimits the space for variation?

We discuss these questions in §2.2. First, we address the problem of learning from structures in the base logic.

2.1 Solving Type Equations by Hypothetical Reasoning

The unification perspective on learning type assignments from structures is well understood — we refer the reader to the seminal work of [5], and to [9]. Here we present the problem of solving type assignment equations from a Logic Programming perspective in order to highlight the role of hypothetical reasoning in the process.

Consider the standard abstract interpreter for logic programs (see for example [16]). The resolution algorithm takes as input a program P and a goal G and initializes the resolvent to be the input goal G. While the resolvent is non-empty, the algorithm chooses a goal A from the resolvent and a matching program clause $A' \leftarrow B_1, \ldots, B_n$ $(n \geq 0)$ from P such that A and A' unify with mgu θ. A is removed from the resolvent and $B_1, \ldots, B_n$ added instead, with θ applied to the resolvent and to G. As output, the algorithm produces $G\theta$, if the resolvent is empty, or failure, if the empty clause cannot be derived.

[6] presents a variant on this refutation algorithm which does not return failure for an incomplete derivation, but instead extracts information from the non-empty resolvent which provides the *conditional answer* that would make the goal G derivable. In [6] the conditional answer approach is illustrated with the polymorphic type inference problem from lambda calculus. This illustration can be straightforwardly adapted to our categorial type inference problem. Writing $\Gamma \triangleleft \Delta$ for functor-argument structures with the functor as the left component and $\Gamma \triangleright \Delta$ for such structures with the functor as the right component, the 'program clauses' for categorial type assignment appear as (5).

(5) $\Gamma \triangleleft \Delta \vdash A$ *if* $\Gamma \vdash A/B$ *and* $\Delta \vdash B$ $\qquad$ $\Gamma \triangleright \Delta \vdash A$ *if* $\Gamma \vdash B$ *and* $\Delta \vdash B\backslash A$

In order to derive the empty clause, the program would need a lexicon of type assignment facts. In the absence of such a lexicon (or in the case of an incomplete lexicon), the conditional answer derivation returns the resolvent with the type

此为试读，需要完整PDF请访问：www.ertongbook.com

assignment goals that would be needed for a successful refutation. The conditional answer is optimized by *factoring*, i.e. by the contraction of unifiable type assignment goals. A sample run of the algorithm is given below.

input Alice ▷ dreams, Lewis ▷ dreams, (the ◁ girl) ▷ dreams, Alice ▷ (knows ◁ Lewis), Lewis ▷ (knows ◁ Alice), (the ◁ mathematician) ▷ (knows ◁ Alice), Alice ▷ (knows ◁ (the ◁ mathematician)), Alice ▷ (irritates ◁ (the ◁ mathematician)), (the ◁ mathematician) ▷ (irritates ◁ Alice), Alice ▷ (dreams ▷ (about ◁ Lewis)), Lewis ▷ (wrote ◁ (the ◁ book)), Lewis ▷ (knows ◁ (the ◁ girl)), Lewis ▷ (wrote ◁ (the ◁ (nice ◁ book))), (the ◁ girl) ▷ (knows ◁ (the ◁ book)), (the ◁ girl) ▷ (knows ◁ (the ◁ (book ▷ (which ◁ (irritates ◁ (the ◁ mathematician)))))), (the ◁ girl) ▷ (knows ◁ (the ◁ (book ▷ ((which ◁ (the ◁ mathematician)) ◁ wrote)))), ...

output The term assignments of (6). With the gloss $A = n$, $B = np$, $C = s$ for the type variables, these will look familiar enough.

(6)		
	Alice $\vdash B$	Lewis $\vdash B$
	dreams $\vdash B\backslash C$	the $\vdash B/A$
	girl $\vdash A$	mathematician $\vdash A$
	book $\vdash A$	nice $\vdash A/A$
	about $\vdash ((B\backslash C)\backslash(B\backslash C))/B$	irritates $\vdash (B\backslash C)/B$
	knows $\vdash (B\backslash C)/B$	wrote $\vdash (B\backslash C)/B$
	which $\vdash (A\backslash A)/(B\backslash C)$	which $\vdash ((A\backslash A)/((B\backslash C)/B))/B$

The interesting point of this run is the two type assignments for 'which': one for subject relativization (obtained from '... which irritates Alice'), the other for object relativization (from '... which Lewis wrote') — and, of course, many others for different structural contexts. Factoring (unification) cannot relate these assignments: this would require structural reasoning. To see that the learning algorithm is missing a generalization here, consider the meaning assembly for these two type assignments to the relative pronoun. The lambda program of (7a), expressing property intersection, would be adequate for the subject relativization type. To obtain appropriate meaning assembly for the object relativization assignment, one would need the lambda program of (7b).

(7)		
a.	which $\vdash (n\backslash n)/(np\backslash s)$	$\lambda x\lambda y\lambda z.(x\ z) \wedge (y\ z)$ $(= \mathbf{wh})$
b.	which $\vdash ((n\backslash n)/((np\backslash s)/np))/np$	$\lambda x'\lambda y'\lambda y\lambda z.((y'\ z)\ x') \wedge (y\ z)$

The point is that these two meaning programs are not unrelated. We can see this by analysing them from the perspective of **LP** (or Multiplicative Linear Logic) — a system which removes all structural obstacles to meaning composition in the sense that free restructuring and reordering under Associativity and Commutativity are available. In **LP**, the different type assignments are simply structural realizations of one and the same meaning. See the derivation of Figure 1, which produces the proof term of (8). Unfortunately, **LP** is of little use as a framework for natural language analysis: apart from the required meaning assembly of (8), there is a second derivation for the type transition of Figure 1 producing the proof term $\lambda y_1.\lambda x_2.(\mathbf{wh}\ (x_2\ y_1))$, which gets the thematic roles of subject and object wrong. Can we find a way of controlling structural reasoning, so that we can do with a single type assignment to the relative pronoun, while keeping the thematic structure intact?

$$\dfrac{\dfrac{\dfrac{[\mathrm{p}_2 \vdash n]^4 \quad \dfrac{\dfrac{\text{which}}{(n\backslash n)/(np\backslash s)} \quad \dfrac{\dfrac{\dfrac{\dfrac{[\mathrm{p}_1 \vdash np]^2 \quad \dfrac{[\mathrm{r}_1 \vdash (np\backslash s)/np]^3 \quad [\mathrm{r}_0 \vdash np]^1}{\mathrm{r}_1 \circ \mathrm{r}_0 \vdash np\backslash s}[/E]}{\mathrm{p}_1 \circ (\mathrm{r}_1 \circ \mathrm{r}_0) \vdash s}[\backslash E]}{(\mathrm{p}_1 \circ \mathrm{r}_1) \circ \mathrm{r}_0 \vdash s}[\mathrm{Ass}]}{\mathrm{r}_0 \circ (\mathrm{p}_1 \circ \mathrm{r}_1) \vdash s}[\mathrm{Comm}]}{\mathrm{p}_1 \circ \mathrm{r}_1 \vdash np\backslash s}[\backslash I]^1}{\text{which} \circ (\mathrm{p}_1 \circ \mathrm{r}_1) \vdash n\backslash n}[/E]}{\mathrm{p}_2 \circ (\text{which} \circ (\mathrm{p}_1 \circ \mathrm{r}_1)) \vdash n}[\backslash E]}{\mathrm{p}_2 \circ ((\text{which} \circ \mathrm{p}_1) \circ \mathrm{r}_1) \vdash n}[\mathrm{Ass}]}{\dfrac{\dfrac{(\text{which} \circ \mathrm{p}_1) \circ \mathrm{r}_1 \vdash n\backslash n}{\text{which} \circ \mathrm{p}_1 \vdash (n\backslash n)/((np\backslash s)/np)}[/I]^3}{\text{which} \vdash ((n\backslash n)/((np\backslash s)/np))/np}[/I]^2}[\backslash I]^4$$

Fig. 1. Relating subject and object relativization in **LP**.

(8) $$\mathbf{wh} \vdash_{\mathbf{LP}} \lambda y_1.\lambda x_2.(\mathbf{wh}\ \lambda z_0.((x_2\ z_0)\ y_1)) \quad =_\beta \quad \lambda y_1.\lambda x_2.\lambda z_3.\lambda x_4.(((x_2\ x_4)\ y_1) \wedge (z_3\ x_4))$$

2.2 Modal Decorations for Solving Structural Equations

To answer this question, we turn to the second phase of the learning process. Let $\mathsf{type}(w)$ be the set of types which the algorithm of §2.1 associates with a word w in the base logic lexicon: $\mathsf{type}(w) = \{A \mid \langle w, A\rangle \in \mathsf{Lex}\}$. The type assignments found in §2.1 are built up in terms of the binary connectives / and \: they do not exploit the full type language, and they do not appeal to structural reasoning. In the second phase of the learning cycle, these limitations are lifted. We translate the question at the end of the previous section as follows: Can we find a modal decoration $\cdot^\natural$ and an associated structural package $\mathcal{R}$ which would allow the learner to identify a $B \in \mathsf{type}(w)$ such that $B^\natural \vdash A$ for all $A \in \mathsf{type}(w)$? Or, if we opt for a weaker package $\mathcal{R}$ that makes unique type-assignment unattainable, can we at least reduce the cardinality of $\mathsf{type}(w)$ by removing some derivable type assignments from the type-set? In what follows, we consider various options for $\cdot^\natural$ and for $\mathcal{R}$.

Consider first the decorations $\lfloor\cdot\rfloor, \lceil\cdot\rceil : \mathbf{B} \mapsto \mathbf{B}\Diamond$ of (9) for *input* and *output* polarities respectively.[2]

(9)
$$\begin{array}{rclrcl} \lfloor p \rfloor &=& p & \lceil p \rceil &=& p \\ \lfloor A \bullet B \rfloor &=& \Diamond\Box\lfloor A\rfloor \bullet \Diamond\Box\lfloor B \rfloor & \lceil A \bullet B \rceil &=& \lceil B\rceil \bullet \lceil A \rceil \\ \lfloor A/B \rfloor &=& \Diamond\Box\lfloor A\rfloor / \lceil B \rceil & \lceil A/B \rceil &=& \lceil A\rceil / \Diamond\Box\lfloor B \rfloor \\ \lfloor B\backslash A \rfloor &=& \lceil B\rceil \backslash \Diamond\Box\lfloor A \rfloor & \lceil B\backslash A \rceil &=& \Diamond\Box\lfloor B\rfloor \backslash \lceil A \rceil \end{array}$$

The effect of $\lfloor\cdot\rfloor, \lceil\cdot\rceil$ is to prefix all input subformulas with $\Diamond\Box$. In the absence of structural rules, this modal marking would indeed be a pure embellishment, in the sense that $\mathbf{B} \vdash \bullet\Gamma \Rightarrow A$ iff $\mathbf{B}\Diamond \vdash \lfloor \bullet\Gamma \rfloor \Rightarrow \lceil A \rceil$. But we are interested in the situation where the $\Diamond\Box$ decoration gives access to structural reasoning. As a crude first attempt, consider the postulate package (10) which would be the modal analogue of (3), i.e. it allows full restructuring and reordering under $\Diamond$ control. In the Associativity postulates, one of the factors A_i $(1 \leq i \leq 3)$ is of the form $\Diamond A'$, with the rule label indexed accordingly.

(10)
$$\begin{array}{cl} A_1 \bullet (A_2 \bullet A_3) \vdash (A_1 \bullet A_2) \bullet A_3 & [A_i] \\ (A_1 \bullet A_2) \bullet A_3 \vdash A_1 \bullet (A_2 \bullet A_3) & [A_i^{-1}] \\ & \\ B \bullet \Diamond A \vdash \Diamond A \bullet B & [C] \\ \Diamond B \bullet A \vdash A \bullet \Diamond B & [C^{-1}] \end{array}$$

With the package (10), we again have the embedding

$$\mathbf{LP} \vdash \bullet\Gamma \Rightarrow A \quad \text{iff} \quad \mathbf{B}\Diamond + (10) \vdash \lfloor \bullet\Gamma \rfloor \Rightarrow \lceil A \rceil$$

Consider again the type assignments we computed in (6) for the relative pronoun:

$$\begin{array}{rl} \mathsf{type}(\text{which}) = \{ & (n\backslash n)/(np\backslash s), \\ & ((n\backslash n)/((np\backslash s)/np))/np, \\ & \ldots \} \end{array}$$

Calculating $\lfloor\cdot\rfloor$ for the first of these, we obtain (11)

(11)
$$\begin{array}{rl} \lfloor (n\backslash n)/(np\backslash s) \rfloor &= \Diamond\Box\lfloor n\backslash n\rfloor / \lceil np\backslash s \rceil \\ &= \Diamond\Box(\lceil n\rceil \backslash \Diamond\Box\lfloor n \rfloor)/(\Diamond\Box\lfloor np\rfloor\backslash\lceil s\rceil) \\ &= \Diamond\Box(n\backslash\Diamond\Box n)/(\Diamond\Box np\backslash s) \end{array}$$

which indeed gives the type transformation

$$\Diamond\Box(n\backslash\Diamond\Box n)/(\Diamond\Box np\backslash s) \vdash ((n\backslash n)/((np\backslash s)/np))/np$$

In Figure 2, we give an example derived from the modalized type assignment to the relative pronoun. We concentrate on the subderivation that realizes non-peripheral extraction via $\Diamond$ controlled structural reasoning. The modal decoration implements the 'key and lock' strategy of [14]. For a constituent of type

[2] $\mathbf{B}$ is the base logic for the binary connectives: the pure residuation logic for $/, \bullet, \backslash$, with no structural postulates at all. $\mathbf{B}\Diamond$ is the extended system with the unary connectives $\Diamond, \Box$.

$\Diamond\Box A$, the $\Diamond$ component provides access to the structural postulates in (10). At the point where such a marked constituent has found the structural position where it can be used by the logical rules, the $\Diamond$ key unlocks the $\Box$ lock: the control feature is cancelled through the basic law $\Diamond\Box A \vdash A$.

$$\dfrac{[\mathrm{r}_1 \vdash \Diamond\Box np]^4 \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Lewis}}{np} \qquad \dfrac{\dfrac{\dfrac{\text{dedicated}}{((np\backslash s)/pp)/np} \quad \dfrac{[\mathrm{p}_2 \vdash \Box np]^5}{\langle \mathrm{p}_2\rangle \vdash np}\,[\Box E]}{\text{dedicated} \circ \langle \mathrm{p}_2\rangle \vdash (np\backslash s)/pp}\,[/E] \quad \dfrac{\dfrac{\text{to}}{pp/np} \quad \dfrac{\text{Alice}}{np}}{\text{to} \circ \text{Alice} \vdash pp}\,[/E]}{(\text{dedicated} \circ \langle \mathrm{p}_2\rangle) \circ (\text{to} \circ \text{Alice}) \vdash np\backslash s}\,[/E]}{\text{Lewis} \circ ((\text{dedicated} \circ \langle \mathrm{p}_2\rangle) \circ (\text{to} \circ \text{Alice})) \vdash s}\,[\backslash E]}{\text{Lewis} \circ (\text{dedicated} \circ (\langle \mathrm{p}_2\rangle \circ (\text{to} \circ \text{Alice}))) \vdash s}\,[A_2]}{\text{Lewis} \circ (\text{dedicated} \circ ((\text{to} \circ \text{Alice}) \circ \langle \mathrm{p}_2\rangle)) \vdash s}\,[C]}{\text{Lewis} \circ ((\text{dedicated} \circ (\text{to} \circ \text{Alice})) \circ \langle \mathrm{p}_2\rangle) \vdash s}\,[A_3^{-1}]}{(\text{Lewis} \circ (\text{dedicated} \circ (\text{to} \circ \text{Alice}))) \circ \langle \mathrm{p}_2\rangle \vdash s}\,[A_3^{-1}]\ \dagger}{\langle \mathrm{p}_2\rangle \circ (\text{Lewis} \circ (\text{dedicated} \circ (\text{to} \circ \text{Alice}))) \vdash s}\,[C^{-1}]}{\dfrac{\mathrm{r}_1 \circ (\text{Lewis} \circ (\text{dedicated} \circ (\text{to} \circ \text{Alice}))) \vdash s}{\text{Lewis} \circ (\text{dedicated} \circ (\text{to} \circ \text{Alice})) \vdash \Diamond\Box np\backslash s}\,[\backslash I]^4}\,[\Diamond E]^5$$

Fig. 2. Non-peripheral extraction under $\Diamond$ control: '(the book which) Lewis dedicated to Alice'. The ($\dagger$) sign marks the entry point for an alternative derivation, driven from an assignment $(n\backslash n)/(s/\Diamond\Box np)$ for the relative pronoun. See the discussion in §2.3.

2.3 Calibration

The situation we have obtained is a crude first approximation in two respects. First, the *modal decoration* is overly rich in the sense that every input subformula is given a chance to engage in structural reasoning. Second, the structural package (10) is not much better that the global structural reasoning of §1 in that it allows full reordering and restructuring, this time under $\Diamond$ control. The task here is to find the proper trade-off between the degree of lexical ambiguity one is prepared to tolerate, and the expressivity of the structural package. We discuss these two considerations in turn.

STRUCTURAL REASONING Consider first the structural component. The package in (12) seems to have a pleasant balance between expressivity and structural constraint. We refer the reader to the discussion of extraction asymmetries between head-initial and head-final languages in [14], Dutch verb-raising in [13], and the analysis of French cliticization in [10], all of which are based essentially on the structural features of (12). In this section, we discuss the postulates in their schematic form — further fine-tuning in terms of mode distinctions for the $\bullet$ and $\Diamond$ operations is straightforward and will be taken into consideration in §3.

(12)
$$\begin{array}{ll} \Diamond A \bullet (B \bullet C) \dashv\vdash (\Diamond A \bullet B) \bullet C & (Pl1) \\ \Diamond A \bullet (B \bullet C) \dashv\vdash B \bullet (\Diamond A \bullet C) & (Pl2) \\ & \\ (A \bullet B) \bullet \Diamond C \dashv\vdash (A \bullet \Diamond C) \bullet B & (Pr2) \\ (A \bullet B) \bullet \Diamond C \dashv\vdash A \bullet (B \bullet \Diamond C) & (Pr1) \end{array}$$

The postulates can be read in two directions. In the $\vdash$ direction, they have the effect of *revealing* a $\Diamond$ marked constituent, by promoting it from an embedded position to a position where it is visible for the logical rules: the immediate left or right daughter of the structural root node.[3] In the $\dashv$ direction, they *hide* a marked constituent, pushing it from a visible position to an embedded position. Apart from the $\dashv\vdash$ asymmetry, the postulates preserve the left-right asymmetry of the primitive operations / and \: the *Pl* postulates have a bias for left branches; for the *Pr* postulates only right branches are accessible.

We highlight some properties of this package.

Linearity The postulates rearrange a structural configuration; they cannot duplicate or waste grammatical material.

Control The postulates operate under $\Diamond$ control. Because the logic doesn't allow the control features to enter a derivation out of the blue, this means they have to be lexically anchored.

Locality The window for structural reasoning is strictly local: postulates can only see two products in construction with each other (with one of the factors bearing the licensing $\Diamond$ feature).

Recursion Non-local effects of structural reasoning arise through recursion.

In comparison with universal package (10), the postulates of (12) represent a move towards *more specific* forms of structural reasoning. One can see this in the deconstruction of *Pr*2 (similarly, *Pl*2) as the compilation of a sequence of structural inferences in (10). The postulate *C* of (10) is removed from (12) as an independent structural inference; instead, a restricted use of it is *encapsulated* in *Pr*2 (or *Pl*2).

$$(A \bullet B) \bullet \Diamond C \vdash (A \bullet \Diamond C) \bullet B \qquad (Pr2)$$

$$\text{Combinator:} \quad Pr2 = \gamma^{-1}(\gamma(A_2) \circ C) \circ A_3^{-1}$$

$$(A \bullet B) \bullet \Diamond C \overset{A_3^{-1}}{\vdash} A \bullet (B \bullet \Diamond C) \overset{C}{\vdash} A \bullet (\Diamond C \bullet B) \overset{A_2}{\vdash} (A \bullet \Diamond C) \bullet B$$

The careful reader may have noticed that the package (12) is too weak to allow the derivation of the extraction example in Figure 2. The modalized type

[3] The reader should keep in mind that, as a result of the cut rule, it is the pattern to the *left* of $\vdash$ that shows up in the conclusion of the natural deduction inferences we have given.

assignment for the relative pronoun in (11) has $(\Diamond\Box np\backslash s)$ as the subtype for the relative clause body: the $\Diamond\Box np$ hypothesis is withdrawn to the left. But this means that complement positions on right branches are inaccessible for the $\Diamond\Box np$ gap hypothesis, if we want to stay within the limits of (12). Accessing a right branch position from the launching point of $\Diamond\Box np$ would require the extra postulate $Pl3$ (and by symmetry $Pr3$), establishing communication between the left- and right-biased options. Again, these are forms of structural reasoning encapsulating a controlled amount of C.

(13)
$$\begin{array}{lll} \Diamond A \bullet (B \bullet C) \dashv\vdash B \bullet (C \bullet \Diamond A) & (Pl3)? \\ (A \bullet B) \bullet \Diamond C \dashv\vdash (\Diamond C \bullet A) \bullet B & (Pr3)? \end{array}$$

$$\dfrac{\dfrac{[\mathrm{q}_1 \vdash \Diamond\Box np]^1 \qquad \dfrac{\dfrac{\dfrac{\dfrac{\text{Lewis}}{np} \qquad \dfrac{\dfrac{\dfrac{\text{dedicated}}{((np\backslash s)/pp)/np} \qquad \dfrac{[\mathrm{s}_1 \vdash \Box np]^2}{\langle \mathrm{s}_1\rangle \vdash np}[\Box E]}{\text{dedicated} \circ \langle \mathrm{s}_1\rangle \vdash (np\backslash s)/pp}[/E] \qquad \dfrac{\dfrac{\text{to}}{pp/np} \qquad \dfrac{\text{Alice}}{np}}{\text{to} \circ \text{Alice} \vdash pp}[/E]}{(\text{dedicated} \circ \langle \mathrm{s}_1\rangle) \circ (\text{to} \circ \text{Alice}) \vdash np\backslash s}[/E]}{\text{Lewis} \circ ((\text{dedicated} \circ \langle \mathrm{s}_1\rangle) \circ (\text{to} \circ \text{Alice})) \vdash s}[\backslash E]}{\text{Lewis} \circ ((\text{dedicated} \circ (\text{to} \circ \text{Alice})) \circ \langle \mathrm{s}_1\rangle) \vdash s}[Pr2]}{(\text{Lewis} \circ (\text{dedicated} \circ (\text{to} \circ \text{Alice}))) \circ \langle \mathrm{s}_1\rangle \vdash s}[Pr1]}{(\text{Lewis} \circ (\text{dedicated} \circ (\text{to} \circ \text{Alice}))) \circ \mathrm{q}_1 \vdash s}[\Diamond E]^2}{\text{Lewis} \circ (\text{dedicated} \circ (\text{to} \circ \text{Alice})) \vdash s/\Diamond\Box np}[/I]^1$$

Fig. 3. Non-peripheral extraction in terms of the package (12). Compare with the derivation in Figure 2.

There is a lexical alternative to strengthening the postulate package which is obtained from a directional variant of the type assignment to the relative pronoun, with $(s/\Diamond\Box np)$ as the subtype for the relative clause body. Under this alternative, the lexicon assigns two types to 'which': one for subject relativization, one for non-subject cases. See the derivation in Figure 3. Notice the trade-off here between increasing the size of the lexicon (storage) versus simplification of the on-line computation (the structural package). Different learners could make different choices with respect to this trade-off: we do not want to assume that the solution for the lexicon and the structural module has to be unique. Individual solutions can count as equally adequate as long as they associate the same forms with the same meaning. As we have noticed in §1, meaning composition is fully determined by the logical introduction/elimination rules for the type-logical constants modulo directionality (i.e. / and \ are indentified).

The two alternatives above make different choices with respect to the distribution of grammatical complexity over the lexicon and the structural module. For an example of alternative solutions that are essentially of the same complexity, we refer to the analysis of Dutch verb raising (VR) in [13], where a leftwing

此为试读，需要完整PDF请访问：www.ertongbook.com