Switch Graphs for Parsing Type Logical Grammars*

Bob Carpenter
Alias-i
181 North 11th Street, #401
Brooklyn, NY, 11211
carp@colloquial.com

Glyn Morrill
Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
E-08034 Barcelona
morrill@lsi.upc.edu

Abstract

Parsing in type logical grammars amounts to theorem proving in a substructural logic. This paper takes the proof net presentation of Lambek's associative calculus as a case study. It introduces switch graphs for online maintenance of the Danos-Regnier acyclicity condition on proof nets. Early detection of Danos-Regnier acyclicity violations supports early failure in shift-reduce parsers. Normalized switch graphs represent the combinatorial potential of a set of analyses derived from lexical and structural ambiguities. Packing these subanalyses and memoizing the results leads directly to a dynamic programming algorithm for Lambek grammars.

1 Introduction

Following Montague (1970), we take the goal of a theory of grammar to be that of assigning semantic terms to linguistic expressions. Type logical grammar is a paradigm for developing grammatical theories based on a strong notion of typing for natural language expressions. Specifically, each linguistic expression is assigned a syntactic type and a semantic term. For instance, the expression "John read the book" of English might be assigned a syntactic type S and the semantic term **read**(**the**(**book**))(**j**), the expression "book that John read" the term **that**(λx.**read**(x)(**j**))(**book**) and type CN, and "person that read the book" the type CN and term **that**(λy.**read**(**the**(**book**))(y))(**person**).

Supported by CICYT project TIC2002–04019–C03–01.

2 Lambek's Associative Calculus

Lambek's associative calculus **L** (Lambek 1958) contains three connectives: concatenation, left division, and right division. Logically, concatenation is conjunction and the divisions are directed implications. Algebraically, concatenation is a free semigroup product and the divisions its left and right residuals. Viewed as a purely syntactic formalism, **L** assigns syntactic types to linguistic expressions modeled as sequences of tokens. From a stipulated lexical assignment of expressions to syntactic types, further assignments of expressions to types are derived through purely logical inference, with the logic representing a sound and complete axiomatization and inference system over the algebraic structure (Pentus 1995).

L appears near the bottom of a hierarchy of substructural logics obtained by dropping structural rules: Lambek proofs are valid as multiplicative intuitionistic linear proofs (restoring permutation) which are valid as conjuntive and implicative relevance proofs (restoring contraction) which are valid as conjuntive and implicative intuitionistic proofs (restoring weakening). In type logical grammars, lexical entries are associated with syntactic types and intuitionistic (in fact probably relevant) proofs as semantic representations, notated as terms of the simply typed λ-calculus with product, under the Curry-Howard correspondence. The semantics of a

derived expression is the result of substituting the lexical semantics into the reading of the derivation as an intuitionistic proof.

2.1 Syntactic and Semantic Types

The set of *syntactic types* is defined recursively on the basis of a set SynAtom of *atomic syntactic types*. The full set SynTyp of syntactic types is the least set containing the atomic syntactic types SynAtom and closed under the formation of products (SynTyp$\cdot$SynTyp), left divisions (SynTyp$\backslash$SynTyp), and right divisions (SynTyp/SynTyp). The two division, or "slash", types, A/B, read *A over B*, and $B\backslash A$, read *B under A*, refine the semantic function types by providing a directionality of the argument with respect to the function. A linguistic expression assigned to type A/B combines with an expression of type B on its right side to produce an expression of type A. An expression of type $B\backslash A$ combines with an expression of syntactic type B on its left to produce an expression of type A. The product syntactic type $A\cdot B$ is assigned to the concatenation of an expression of type A to an expression of type B. The distinguishing feature of Lambek calculus with respect to the earlier categorial grammar of Bar-Hillel is that as well as the familar cancelation (modus ponens) rules, it admits also a form of the deduction theorem: if the result of concatenating an expression e to each B results in an expression of type A, then it follows that e is assigned to syntactic type A/B.

Semantic representations in Lambek type logical grammar are simply typed λ-terms with product. We assume a set SemAtom of *atomic semantic types*, which generate the usual *function types* $\sigma \to \tau$ and *product types* $\sigma \times \tau$. Terms are grounded on an infinite set of distinct *variables* Var_σ, along with a set of distinct *contants* Con_σ for each type σ. We assume the usual *λ-terms* consisting of variables, constants, *function applications* $\alpha(\beta)$, *function abstractions* $\lambda x.\alpha$, *pairs* $\langle\alpha,\beta\rangle$ and *projections* from pairs $\pi_1\delta$ and $\pi_2\delta$ onto the first and second element of the pair respectively. We say that a term α is *closed* if and only if it contains no free variables.

A *type map* consists of a mapping $\mathsf{typ} : \mathsf{SynAtom} \to \mathsf{SemTyp}$. That is, each atomic syntactic type $A \in \mathsf{AtomCat}$ is assigned to a (not necessarily atomic) semantic type $\mathsf{typ}(A) \in \mathsf{SemTyp}$. Semantic types are assigned to complex syntactic types as follows:

$$\mathsf{typ}(A \cdot B) = \mathsf{typ}(A) \times \mathsf{typ}(B) \quad \text{[Product]}$$

$$\mathsf{typ}(A/B) = \mathsf{typ}(B) \to \mathsf{typ}(A) \quad \text{[Right Division]}$$

$$\mathsf{typ}(B\backslash A) = \mathsf{typ}(B) \to \mathsf{typ}(A) \quad \text{[Left Division]}$$

We will often write $\alpha : A$ where α is a λ-term of type $\mathsf{typ}(A)$.

2.2 Linguistic Expressions and the Lexicon

In the Lambek calculus, linguistic expressions are modeled by sequences of atomic symbols. These atomic symbols are drawn from a finite set Tok of *tokens*. The full set of linguistic *expressions* Tok^* is the set of sequences of tokens. For the sake of this short version of the paper we admit the empty sequence; we will address its exclusion (as in the original definition of **L**) in a longer version.

The compositional assignment of semantic terms to linguistic expressions is grounded by a finite set of assignments of terms and types to expressions. A *lexicon* is a finite relation $\mathsf{Lex} \subseteq \mathsf{Tok}^* \times \mathsf{Term} \times \mathsf{SynTyp}$, where all $\langle w, \alpha, A\rangle \in \mathsf{Lex}$ are such that the semantic term α is of the appropriate type for the syntactic type A. We assume that the only terms used in the lexicon are *relevant*, in the sense of relevance logic, in not containing vacuous abstractions. Note that the set of atomic semantic types, atomic syntactic types and the semantic type mapping are assumed as part of the definition of a lexicon. Type logical grammar is an example of a fully lexicalized grammar formalism in that the lexicon is the only locus of language-specific information.

2.3 Proof Nets

A *sequent* $\Gamma \Rightarrow \alpha : A$ is formed from an *antecedent* Γ consisting of a (possibly empty) sequence of λ-term and syntactic type pairs, and a *consequent* pair $\alpha : A$, where the terms are of the appropritate type for the types. Following Roorda (1991), we define theoremhood with Girard-style proof nets (Girard 1987), a geometric alternative to Lambek's Gentzen-style calculus (Lambek 1958).

Proof nets form a graph over nodes labeled by polar types, where a *polar type* is the combination of a syntactic type and one of two *polarities*, *input* (negative) and *output* (positive). We write $A^\bullet$ for the *input polar type*, which corresponds to antecedent types and is thus logicaly negative. We write A° for

the *output polar type*, which is logically positive and corresponds to a consequent type. A *literal* is a polar type with an atomic syntactic type. Where A is an atomic syntactic type, the literals $A^\bullet$ and A° are said to be *complementary*.

Each polar type defines an ordered binary tree rooted at that polar type, known as a *polar tree*. For a literal, the polar tree is a single node labeled by that literal. For polar types with complex syntactic types, the polar tree is rooted at the polar type and unfolded upwards based on connective and polarity according to the solid lines in Figure 1, which includes also other annotation. Examples for some linguistically motivated types are shown in Figure 2.

The solid edges of the graphs are the edges of the *logical links*. Each unfolding is labeled with a multiplicative linear logic connective, either *multiplicative conjunction* ($\otimes$) or *multiplicative disjunction* ($\wp$). This derives from the logical interpretation of the polar type trees as formula trees in multiplicative linear logic. Unfolding the Lambek connectives to their linear counterparts, $(A/B)^\bullet$ and $(B\backslash A)^\bullet$ unfold to $A^\bullet \wp B^\circ$; $(A/B)^\circ$ and $(B\backslash A)^\circ$ unfold to $A^\circ \otimes B^\bullet$; $(A \cdot B)^\bullet$ unfolds to $A^\bullet \otimes B^\bullet$; and $(A \cdot B)^\circ$ unfolds to $A^\circ \wp B^\circ$. The type unfoldings correspond to the classical equivalences between $(\phi \rightarrow \psi)$ and $(\neg\phi \vee \psi)$, between $\neg(\phi \rightarrow \psi)$ and $(\phi \wedge \neg\psi)$, and between $\neg(\phi \wedge \psi)$ and $(\neg\phi \vee \neg\phi)$. For atomic syntactic types A, $A^\bullet$ becomes simply A, whereas A° becomes its linear *negation* $A^\perp$; this is the sense in which polar atomic types correspond to logical literals. The non-commutatitive nature of the Lambek calculus is reflected in the ordering of the subtrees in the unfoldings; for commutative logics, the proof trees are not ordered.

The *proof frame* for a syntactic sequent $C_1, \ldots, C_n \Rightarrow C_0$ is the ordered sequence of polar trees rooted at $C_0^\circ, C_1^\bullet, \ldots, C_n^\bullet$. We convert sequents to frames in this order, with the output polar tree first. In general, what follows applies to any cyclic reordering of these polar trees. Note that the antecedent types $C_1, \ldots C_n$ have input (negative) polarity inputs and the consequent type C_0 has output (positive) polarity. All of our proof frames are *intuitionistic* in that they have a single output conclusion, i.e. a unique polar tree rooted at an output type.

A *partial proof structure* consists of a proof frame with a set of *axiom links* linking pairs of complementary literals with at most one link per literal. Axiom links in both directions are shown in Figure 3. A *proof structure* is a proof structure in which all literals are connected to complementary literals by axiom links.

Proof nets are proof structures meeting certain conditions. A proof structure is *planar* if and only if its axiom links can be drawn in the half-plane without crossing lines; this condition enforces the lack of commutativity of the Lambek calculus. The final condition on proof structures involves switching. A *switching* of a proof structure is a subgraph that arises from the proof structure by removing exactly one edge from each disjunctive ($\wp$) link. A proof structure is said to be *Danos-Regnier (DR-) acyclic* if and only if each of its switchings is acyclic (Danos and Regnier 1989).[1] A *proof net* is a planar DR-acyclic proof structure. A *theorem* is any sequent forming the proof frame of a proof net.

Consider the three proof nets in Figure 4. The first example has no logical links, and corresponds to the simplest sequent derivation $S \Rightarrow S$. The second example represents a determiner, noun and intransitive verb sequence. Both of these examples are acyclic, as must be every proof net with no logical $\wp$-links. The third example corresponds to the type-raising sequent $N \Rightarrow S/(N\backslash S)$. Unlike the other examples, this proof net involves a $\wp$-link and is cyclic. But both of its switchings are acyclic, so it satisfies the Danos-Regnier acyclicity condition.

2.4 Essential Nets and Semantic Trips

A term is said to be *pure* if and only if it contains no constants. The *linear terms* are closed, pure λ-terms that bind each variable exactly once. Each proof net in the Lambek calculus corresponds to a linear (i.e. binding each variable exactly once) λ-term via the Curry-Howard correspondence. This term abstracts over variables standing in for the semantics of the inputs in the antecedent of the sequent and has a body that is determined by the consequent of the sequent. For instance, the λ-term $\lambda x.\lambda P.P(x)$ corresponds to the syntactic sequent $x : N$, P :

[1] The full Danos-Regnier condition is that every switching be acyclic and connected. Fadda and Morrill (2005) show that for the intuitionistic case (i.e. single output conclusion, as for **L**), DR-acyclicity entails the connectedness of every switching.

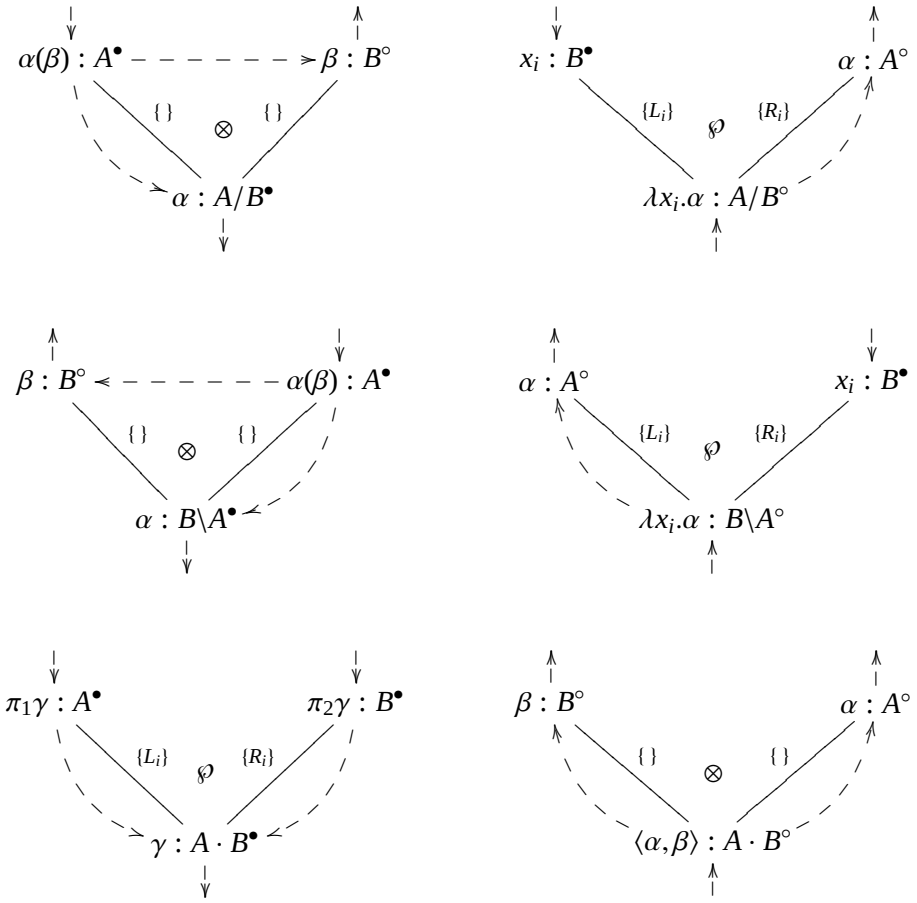

Figure 1: Logical Links with Switch Paths (solid) and Semantic Trip (dashed)

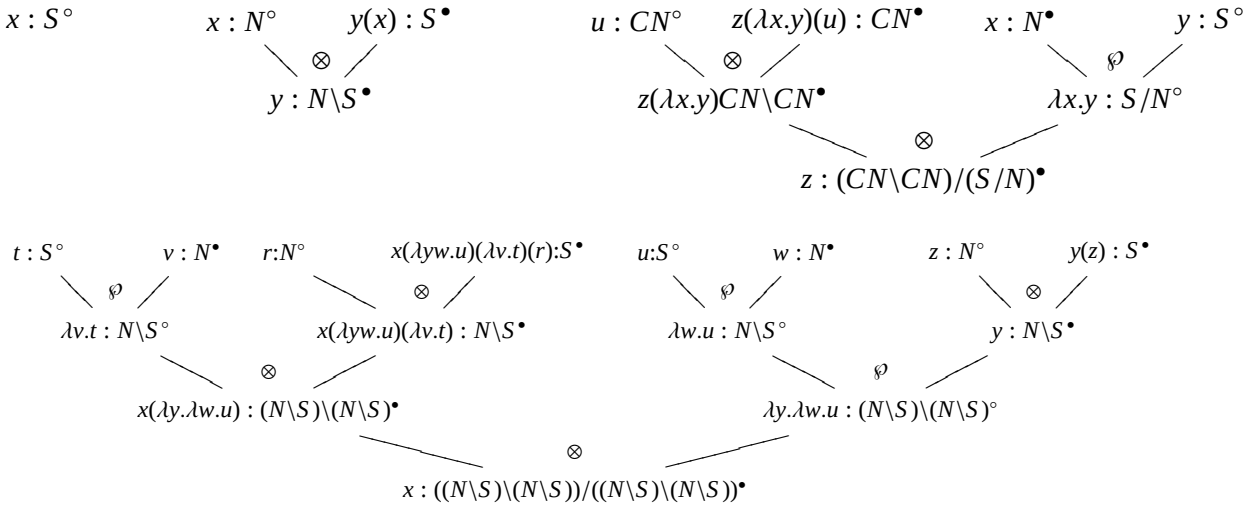

Figure 2: Examples of Polar Type Trees

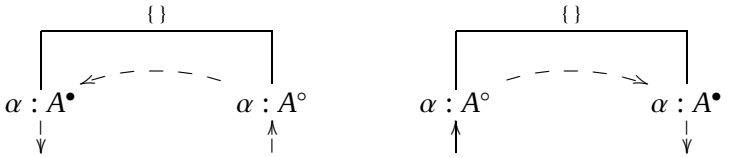

Figure 3: Axiom Links with Switch Paths and Semantic Trip

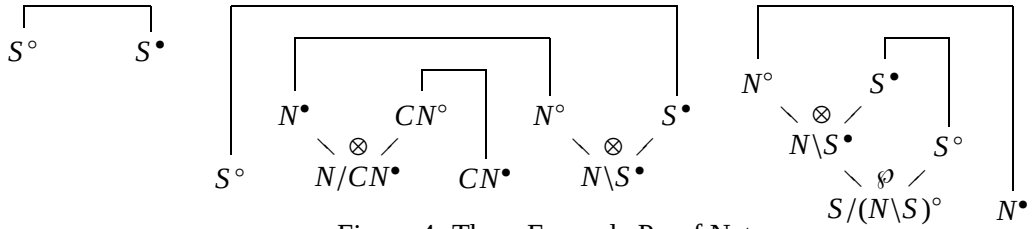

Figure 4: Three Example Proof Nets

$N\backslash S \Rightarrow P(x) : S$ and $\lambda x.\lambda P.P(x)$ corresponds to the sequent $x : N \Rightarrow \lambda P.P(x) : S/(N\backslash S)$. The λ-term induced by the Curry-Howard correspondence can be determined by a unification problem over a proof net (Roorda 1991). Different proof nets for the same theorem correspond to different interpretations through the Curry-Howard correspondence. The *essential net* of a proof structure is the directed graph rooted at the root node of the output polar type tree whose edges are shown as dashed lines in Figures 1 and 3 (LaMarche 1994). Each output division type introduces a fresh variable on its input subtype (its argument), as indicated by the labels x_i in Figure 1. The essential nets for the examples in Figure 4 are shown in Figure 5.

Terms are computed for the polar type trees by assigning terms to the roots of the polar inputs. The tree is then unfolded unifying in substitutions as it goes, as illustrated in the example polar type trees in Figure 2. The direction of axiom links in the essential net provide the substitutions necessary to solve the unification problem of λ-terms in the proof net established by equating the two halves of each axiom linked complementary pair of literals. A traversal of an essential net carrying out the substitutions specified by axiom links constitutes a *semantic trip* the end result of which is the Curry-Howard λ-term for the Lambek calculus theorem derived by the proof net. All λ-terms derived from a semantic trip with variables or constants assigned to input root polar types will be in β-η long form. The essential net directly corresponds to the tree of the semantic term derived by the Curry-Howard correspondence.

The well-formedness of a set of axiom linkings over a polar tree may be expressed in terms of the essential net. Among the conditions are that an essential net must be acyclic and planar. In addition, essential nets must be connected in two ways. First, there must be a path from the root of the single output polar tree to the root of each of the input polar trees. Second, there must be a path from each output daughter of an output division to the input daughter. That is, when A/B° is unfolded to $B^\bullet A^\circ$, there must be a path from A° to $B^\bullet$. These conditions express the definition of linear semantic terms dictated through the logic by the Curry-Howard correspondence. The first condition requires each variable (or term) corresponding to the root of an input polar tree to occur in the output term, whereas the second condition requires that variables only occur within their proper scopes so that they are bound. The essential nets presented in Figure 5 adhere to these conditions and produce well-typed linear λ-terms. The example presented in Figure 6 shows a set of axiom links that does not form a proof net; it violates the condition on variable binding, as is seen from the lack of path from the N° daughter to the $N^\bullet$ daughter of the N/N° node. The major drawback to using these conditions directly in parsing is that they are existential in the sense of requring the existence of a certain kind of path, and thus difficult to refute online during parsing. In comparison, the Danos-

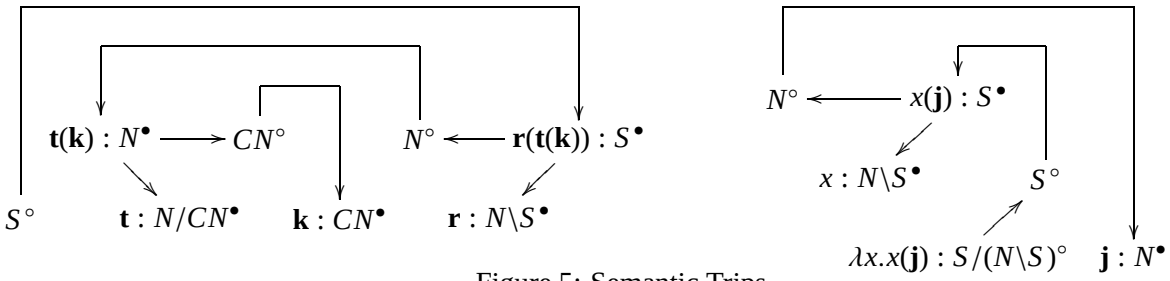

Figure 5: Semantic Trips

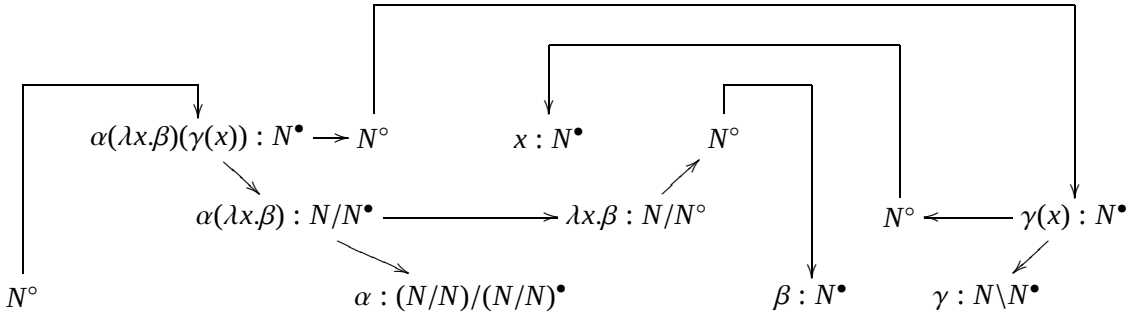

Figure 6: Unbound Variable in Illegal Semantic Trip

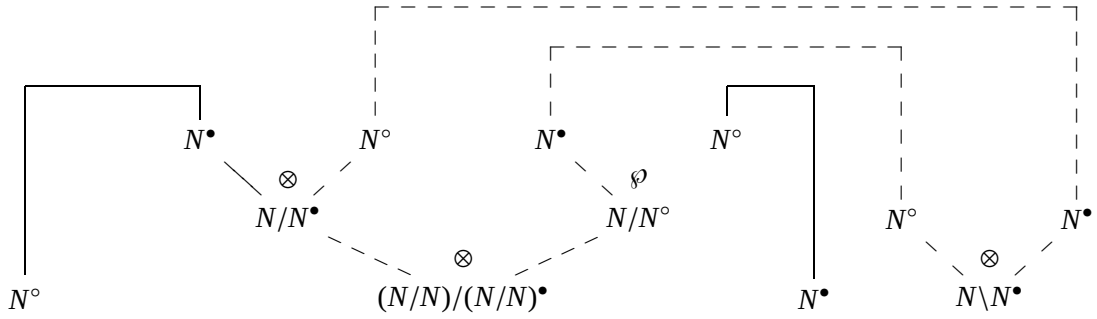

Figure 7: Switching with Path Violating Danos-Regnier Acyclicity

Regnier acyclicity condition is violated by the attempt to close off the binding of the variable. The path vilolating DR acyclicity is shown in Figure 7, with the path given in dashed lines and the switching taking the right daughter of N/N° as the arc to remove.

3 Parsing with Switch Graphs

The planar connection of all literals into a proof structure is straightforward to implement. Axiom links are simply added in such a way that planarity is maintained until a complete linkage is found. In our shift-reduce-style parser, planarity is maintained by a stack in the usual way (Morrill 2000). For dynamic programming, we combine switch graphs in the cells in a Cocke-Kasami-Younger (CKY) parser (Morrill 1996). The main challenge is enforcing DR-acyclicity, and this is the main focus of the rest of the paper. We introduce switch graphs, which not only maintain DR-acyclicity, but also lead the way to a normal form for well-formed subsequence fragments of a partial proof structure. This normal form underlies the packing of ambiguities in subderivations in exactly the same way as usual in dynamic programming parsing.

3.1 Switch Graphs

Switch graphs are based on the observation that a proof structure is DR-acyclic if and only if every cycle contains both edges of a $\wp$-link. If a cycle contains both edges of a $\wp$-link, then any switching removes the cycle. Thus if every cycle in a proof structure contains both edges of a $\wp$-link, every switching is acyclic.

The *(initial) switch graph* of a partial proof structure is defined as the undirected graph underlying the partial proof structure with edges labeled with sets of $\wp$-edge identifiers as indicated in Figures 1 and 3. Each edge in a logical $\wp$-link is labeled with the singleton set containing an identifier of the link itself, either L_i for the left link of $\wp$-link i or R_i for the right link of $\wp$-link i. Edges of axiom links and logical $\otimes$-links are labeled with the empty set.

The *closure* of a switch graph is computed by iterating the following operation: if there is an edge $n_1 - n_2$ labeled with set X_1 and an edge edge $n_2 - n_3$ labeled with set X_2 such that $X_1 \cup X_2$ does not contain both edges of a $\wp$-link, add an edge $n_1 - n_3$ labeled with $X_1 \cup X_2$. An edge $n-m$ labeled by X is *subsumed* by an edge between the same nodes $n-m$ labeled by Y if $Y \subseteq X$. The *normal switch graph* of a partial proof structure is derived by closing its the initial switch graph, removing edges that are subsumed by other edges, and restricting to the literal nodes not connected by an axiom link. These normal switch graphs define a unique representation of the combinatorial possibilities of a span of polar trees and their associated links in a partial proof structure. That is, any partial proof structure substructure that leads to the same normal switch graph may be substituted in any proof net while maintaining well-formedness.

The fundamental insight explored in this paper is that two literals may be connected by an axiom link in a partial proof structure without violating DR-acyclicity if and only if they are not connected in the normal switch graph for the partial proof structure. The normal switch graph arising from the addition of an axiom link is easily computed. It is just the closure generated by adding the new axiom link, with the two literals being linked removed.

3.2 Shift-Reduce Parsing

In this section, we present the search space for a shift-reduce-style parsing algorithm based on switch graphs. The states in the search consist of a *global stack* of literals, a *lexical stack* of literals, the remaining tokens to be processed, and the set of links among nodes on the stacks in the switch graph. The shift-reduce *search space* is characterized by an initial state and state *transitions*. These are shown in schematic form in Figure 8. The *initial state* contains the output type's literals and switch graph. A *lexical* transition is from a state with an empty lexical stack to one containing the lexical literals of the next token; the lexical entry's switch graph merges with the current one. A *shift* transition pops a literal from the lexical stack and pushes it onto the global stack. A *reduce* transition adds an axiom link between the top of the global stack and lexical stack if they are complementary and are not connected in the switch graph; the resulting switch graph results from adding the axiom link and normalizing. The stack discipline insures that all partial proof structures considered are planar.

Figure 10 displays as rows the shift-reduce search

Stack	Lex	Sw-Gr	Op
A°		$\mathsf{gr}(A^\circ)$	start(A)
S		**G**	
S	$A^\bullet$	$\mathbf{G} \oplus \mathsf{gr}(A^\bullet)$	lex(w, A)

Stack	Lex	Sw-Gr	Op
$A_i\mathbf{S}$	$\overline{A}_j\mathbf{L}$	**G**	
S	**L**	$(\mathbf{G} \oplus i{=}j) - \{i, j\}$	reduce(i, j)
$A\mathbf{S}$	$B\mathbf{L}$	**G**	
$BA\mathbf{S}$	**L**	**G**	shift(B)

Figure 8: Shift-Reduce Parsing Schematic

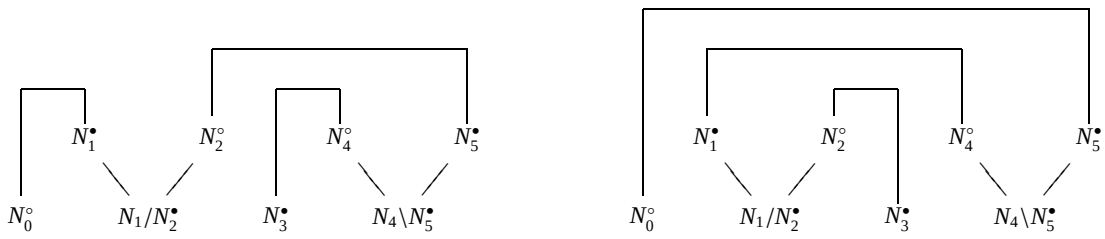

Figure 9: Modifier Attachment Ambiguity Proof Nets

Stack	Lex	Tok	Sw-Gr	Ax	Op
N_0°					start
N_0°	$N_1^\bullet N_2^\circ$	w_1	1-2{}		lex
–	N_2°			0=1	reduce
N_2°					shift
N_2°	$N_3^\bullet$	w_2			lex
$N_3^\bullet N_2^\circ$					shift
$N_3^\bullet N_2^\circ$	$N_4^\circ N_5^\bullet$	w_3	4-5{}		lex
N_2°	$N_5^\bullet$			3=4	reduce
				2=5	reduce

Stack	Lex	Tok	Sw-Gr	Ax	Op
N_0°					start
N_0°	$N_1^\bullet N_2^\circ$	w_1	1-2{}		lex
$N_1^\bullet N_0^\circ$	N_2°		1-2{}		shift
$N_2^\circ N_1^\bullet N_0^\circ$			1-2{}		shift
$N_2^\circ N_1^\bullet N_0^\circ$	$N_3^\bullet$	w_2	1-2{}		lex
$N_1^\bullet N_0^\circ$				2=3	reduce
$N_1^\bullet N_0^\circ$	$N_4^\circ N_5^\bullet$	w_3	4-5{}		lex
N_0°	$N_5^\bullet$			1=4	reduce
				0=5	reduce

Figure 10: Modifier Attachment Ambiguity Shift-Reduce Search States

states corresponding to the two valid proof nets shown in Figure 9. The subscripts on syntactic types in the diagram is only so that they can be indexed in the rows of the table describing the search states. The initial state in both searches is created from the output type's literal. The third column of the diagrams indicate the token consumed at each lexical entry. The switch graphs are shown for the rows for which they're active. Because there are no $\wp$-links, all sets of edges are empty. The fifth column shows the axiom linking made at each reduce step. The history of these decisions and lexical insertion choices determines the final proof net. Finally, the sixth column shows the operation used to derive the result. Note that reduction is from the top of the lexical stack to the top of the global stack and is only allowed if the nodes to be linked are not connected in the switch graph. This is why $N_1^\bullet$ cannot reduce with N_2° in the second diagram in Figure 10; the second shift is mandatory at this point. Note that as active nodes are removed, as in the first diagram reduction step linking 0=2, the switch graph contracts to just the unlinked nodes. After the reduction, only $N_2^\bullet$ is unlinked, so there can be no switch graph links. The link between node 4 and 5 is similarly removed almost as soon as it's introduced in the second reduction step. In the second diagram, the switch graph links persist as lexical literals are pushed onto the stack.

Shift-reduce parses stand in one-to-one correspondence with proof nets. The shift and reduce operations may be read directly from a proof net by working left to right through the literals. Between literals, the horizontal axiom links represent literals on the stack. Literals in the current lexical syntactic type represent the lexical stack. Literals that are shifted to the global stack eventually reduce by axiom linking with a literal to the right; literals that are reduced from the lexical stack axiom link to their left with a literal on the global stack.

3.3 Memoized Parsing

Using switch graphs, we reduce associative Lambek calculus parsing to an infinite binary phrase-structure grammar, where the non-terminals are normalized switch graphs. The phrase structure schemes are shown in Steedman notation in Figure 11. Lexical entries for syntactic type A are derived from the input polar tree rooted at $A^\bullet$. This polar tree yields a switch graph, which is always a valid lexical entry in the phrase structure grammar. Any result of axiom linking adjacent complementary pairs of literals in the polar tree that maintains switch-graph acyclicity is also permitted. For instance, allowing empty left-hand sides of sequents, the input type $A/(B/B)^\bullet$ would produce the literals $A_1^\bullet B_2^\bullet B_3^\circ$ with links 1-2 : $\{L_3\}$, 1-3 : $\{R_3\}$. This could be reduced by taking the axiom link 2=3, to produce the single node switch graph $A_1^\bullet$. In contrast, $(B/B)/A^\bullet$ produces the switch graph $B_1^\bullet B_2^\circ A_3^\circ$ with links 1-2, 1-3, and 2-3. Thus the complementary B literals may not be linked.

Given a pair of normal switch graphs, the binary rule scheme provides a finite set of derived switch graphs. One or more complementary literals may be axiom linked in a nested fashion at the borders of both switch graphs. These sequences are marked as Δ and $\overline{\Delta}$ and their positions are given relative to the other literals in the switch graph in Figure 11. Unlinked combinations are not necessary because the graph must eventually be connected. This scheme is non-deterministic in choice of Δ. For instance, an adverb input $(N_1\backslash S_2)/(N_4\backslash S_3)^\bullet$ produces the literals $N_1^\circ S_2^\bullet S_3^\circ N_4^\bullet$ and connections 1-2, 1-3:$\{L_4\}$, 1-4:$\{R_4\}$, 2-3:$\{L_4\}$, and 2-4:$\{R_4\}$. When it combines with a verb phrase input $N_5\backslash S_6^\bullet$ with literals $N_5^\circ S_6^\bullet$ and connections 5-6, then either the nominals may be linked (4=5), or the nominals and sentential literals may be linked (4=5, 3=6). The result of the single linking is $N_1^\circ S_2^\bullet S_3^\circ S_6^\bullet$ with connections 1-2, 1-3:$\{L_4\}$, 1-6:$\{R_4\}$, 2-3:$\{L_4\}$, and 2-6:$\{R_4\}$. The result of the double linking is simply $N_1^\circ S_6^\bullet$ with connection 1-6, or in other words, a verb phrase.

The dynamic programming equality condition is that two analyses are considered equal if they lead to the same normalized switch graphs. This equality is only considered up to the renaming of nodes and edges. Backpointers to derivations allow semantic readings to be packed in the form of lexical choices and axiom linkings. For instance, consider the two parses in Figure 12.

With a finite set of lexical entries, bottom-up memoized parsing schemes will terminate. We illustrate two derivations of a simple subject-verb-object construction in Figure 13. This is a so-called *spurious ambiguity* because the two derivations produce

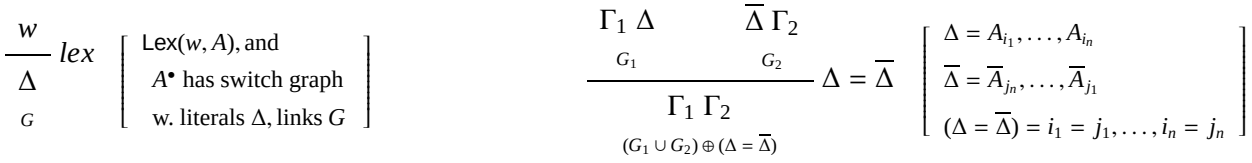

Figure 11: Phrase-Structure Schemes over Switch Graphs

Figure 12: Modifier Attachment Ambiguity Packing

the same semantic term. They are not spurious globally because the alternative linkings are required for adverbial modification and object relativization respectively. The ambiguity in the phrase structure grammar results from the associativity of the combination of axiom linkings. The two derivations do not propagate their ambiguity under the dynamic programming scheme precisely because they produce equivalent results. Nevertheless, a worthwhile optimization is to restrict the structure of combinations of linkings in the phrase-structure schemes to correspond to an unambiguous left-most linking strategy; this corresponds to the way in which other associative operators are parsed in programming language. For instance, $x+y+z$ will be assumed to be $x+(y+z)$ if + is defined to be right associative.

An unambiguous right-associative context-free grammar for linkings M over literals A and their complements $\overline{A}$ is:

$$M \rightarrow \; A\,\overline{A} \;\mid\; A\,M\,\overline{A} \;\mid\; A\,\overline{A}\,M \;\mid\; A\,M\,\overline{A}\,M$$

An example of packing for subject/object scope ambiguities is shown in Figure 14. The derivations in Figure 14 produce different semantic interpretations; one of these is subject-wide scope and the other object-wide scope. Unsurprisingly, the memoizing parser does not solve P = NP in the affirmitive (Pentus 2003). The size of the switch graphs on the intermediate structures is not bounded, nor is the number of alternative switch-paths between literals. It remains an open question as to whether the switch graph parser could be bounded for a fixed lexicon (Pentus 1997).

3.4 Empty Antecedents and Subtending

Lambek's calculus required the antecedent Γ in a sequent $\Gamma \Rightarrow \alpha : A$ to be non-empty. Proof nets derive theorems ($\Rightarrow CN/CN$) and $((CN/CN)/(CN/CN) \Rightarrow CN/CN)$, as shown in Figure 15. These derivations both allow the construction of an output, namely the identity term $\lambda x.x$ and modifier syntactic type CN/CN, out of no input.

A literal A is said to *subtend* a complementary literal $\overline{A}$ if they are the leftmost and rightmost descendants of a $\wp$-link. In both of the examples in Figure 15, the output adjective CN/CN° unfolds to the sequence of literals $CN^{\bullet}CN^{\circ}$ in which the input $CN^{\bullet}$ subtends the output CN°. If literals that stand in a subtending relation are never linked, the set of theorems is restricted to those derivable in Lambek's original calculus.

Consider the proof net in Figure 16. An analysis in which S_8° linked to $S_{11}^{\bullet}$ and $N_9^{\bullet}$ linked to N_{10}° is not ruled out by Danos-Regnier acyclicity. It is ruled out by the subtending condition because S_8° subtends $S_{11}^{\bullet}$, being the leftmost and right most daughters of the $\wp$-node $(N_{10}\backslash S_{11})\backslash(N_9\backslash S_8)^{\circ}$. Further note that there are no cycles violating DR acyclicity; each of the sixteen switchings is acyclic.

4 Conclusion

We have introduced switch graphs for shift-reduce and CKY-style parsing of grammars in the associative Lambek calculus. Switch graphs encode

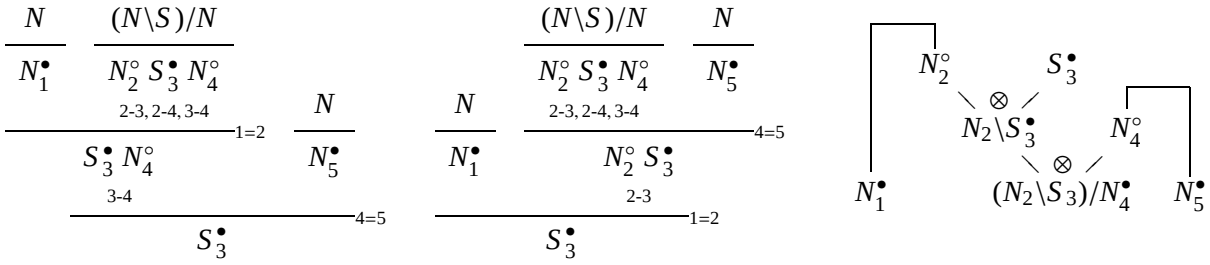

Figure 13: Left vs. Right Attachment: Packing Locally Spurious Attachment Ambiguity

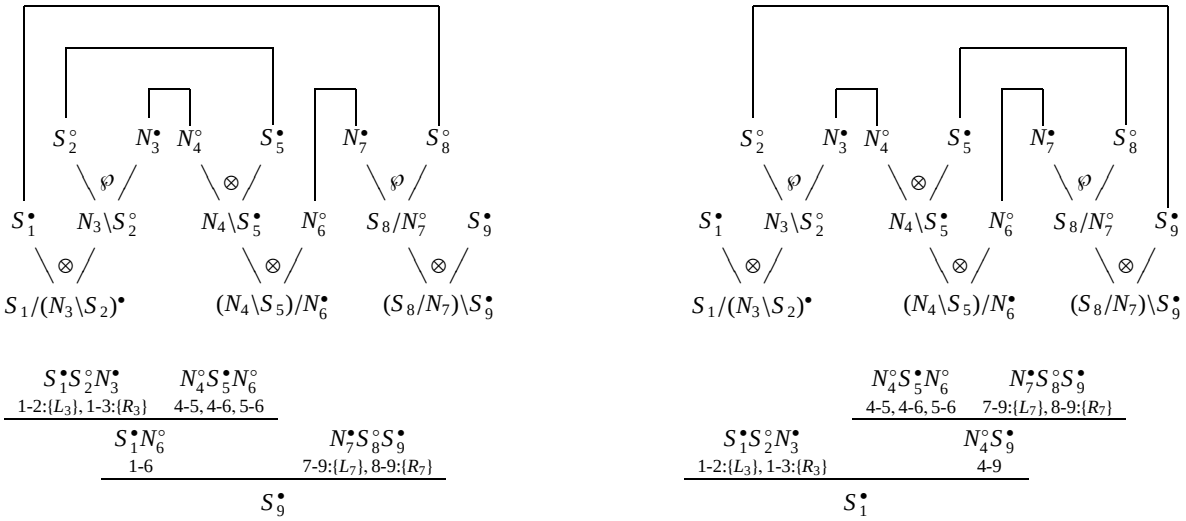

Figure 14: Scope Ambiguity: Partial Proof Structure Fragments with Phrase Structure

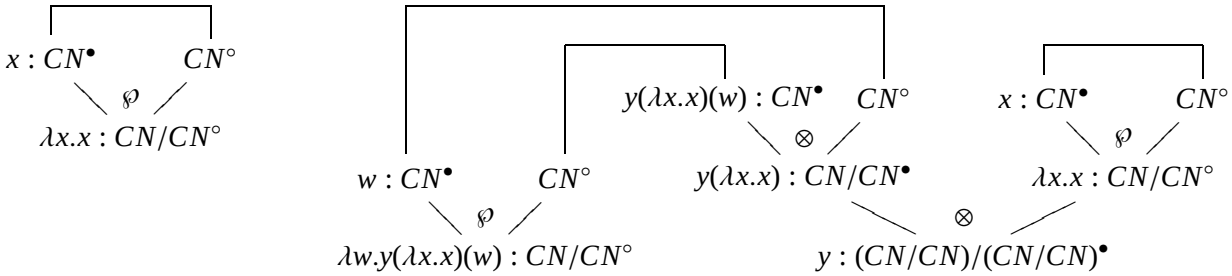

Figure 15: Subtending Examples

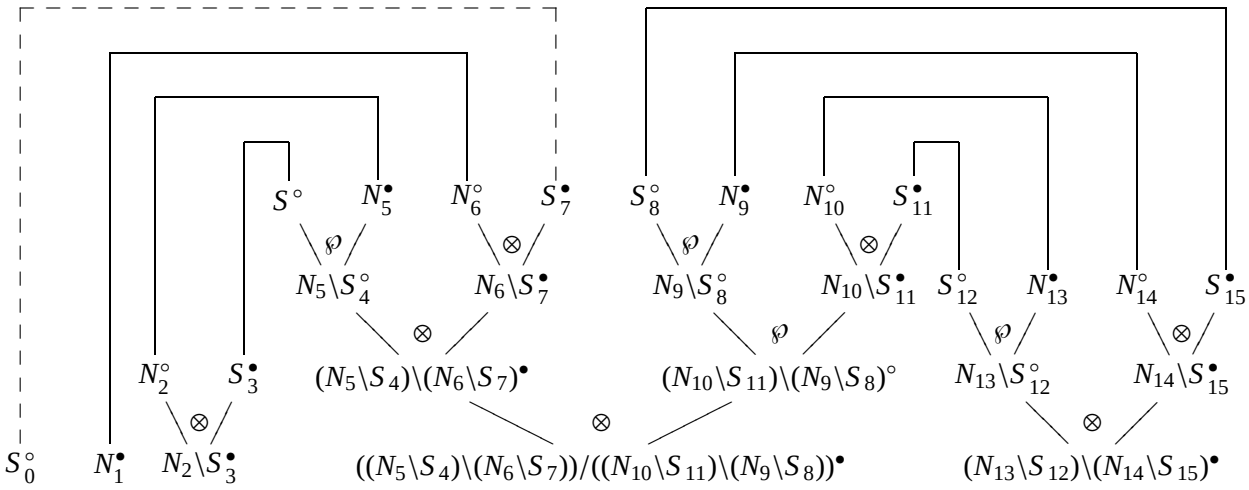

Figure 16: Higher-Order Example: Adverbial Intensifier

the axiom-linking possibilities of a sequence of unlinked literals deriving from underlying polar trees. We introduced two parsers based on switch graphs. The shift-reduce parsers are memory efficient and parses correspond uniquely to (cut free) proof nets. They can be made more efficient by bounding stack size. The memoizing parsers are able to pack attachment and scope distinctions that lead to different λ-terms but have the same combinatory possibilities.

References

D. Bechet. 2003. Incremental parsing of lambek calculus using proof-net interfaces. In *Proc. of the 8th International Workshop on Parsing Technologies*.

V. Danos and L. Regnier. 1989. The structure of multiplicatives. *Arch. Math. Logic*, 28:181–203.

P. de Groote and C. Retoré. 1996. On the semantic readings of proof nets. In *Proc. of Formal Grammar*, pages 57–70.

M. Faddo and G. Morrill. 2005. The Lambek calculus with brackets. In P. Scott, C. Casadio, and R. Seely, editors, *Language and Grammar: Studies in Math. Ling. and Nat. Lang.* CSLI Press, Stanford.

J.-Y. Girard. 1987. Linear logic. *Theoret. Comput. Sci.*, 50:1–102.

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

J. Lambek. 1958. The mathematics of sentence structure. *Amer. Math. Mon.*, 65:154–170.

R. Montague. 1970. Universal grammar. *Theoria*, 36:373–398.

G. Morrill. 1996. Memoisation of categorial proof nets: parallelism in categorial processing. Technical Report LSI-96-24-R, Dept. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya.

G. Morrill. 2000. Incremental processing and acceptability. *Comput. Ling.*, 26(3):319–338.

M. Pentus. 1995. Models for the Lambek calculus. *Annals of Pure and Applied Logic*, 75(1–2):179–213.

M. Pentus. 1997. Product-free Lambek calculus and context-free grammars. *Journal of Symbolic Logic*, 62(2):648–660.

M. Pentus. 2003. Lambek calculus is NP-complete. Technical Report TR-203005, CUNY Graduate Center.

D. Roorda. 1991. *Resource logics: Proof-theoretical investigations*. Ph.D. thesis, Universiteit van Amsterdam.