Negative Polarity Licensing at the Syntax-Semantics Interface

John Fry
Stanford University and Xerox PARC
Dept. of Linguistics
Stanford University
Stanford, CA 94305-2150, USA
fry@csli.stanford.edu

Abstract

Recent work on the syntax-semantics interface (see e.g. (Dalrymple et al., 1994)) uses a fragment of linear logic as a 'glue language' for assembling meanings compositionally. This paper presents a glue language account of how negative polarity items (e.g. *ever, any*) get licensed within the scope of negative or downward-entailing contexts (Ladusaw, 1979), e.g. *Nobody ever left.* This treatment of licensing operates precisely at the syntax-semantics interface, since it is carried out entirely within the interface glue language (linear logic). In addition to the account of negative polarity licensing, we show in detail how linear-logic proof nets (Girard, 1987; Gallier, 1992) can be used for efficient meaning deduction within this 'glue language' framework.

1 Background

A recent strain of research on the interface between syntax and semantics, starting with (Dalrymple et al., 1993), uses a fragment of linear logic as a 'glue language' for assembling the meaning of a sentence compositionally. In this approach, meaning assembly is guided not by a syntactic constituent tree but rather by the flatter functional structure (the LFG *f-structure*) of the sentence.

As a brief review of this approach, consider sentence (1):

(1) Everyone left.

$$f{:}\begin{bmatrix}\text{PRED} & \text{'LEAVE'} \\ \text{SUBJ} & g{:}\begin{bmatrix}\text{PRED} & \text{'EVERYONE'}\end{bmatrix}\end{bmatrix}$$

Each word in the sentence is associated with a 'meaning constructor' template, specified in the lexicon; these meaning constructors are then instantiated with values from the f-structure. For sentence (1), this produces two premises of the linear logic glue language:

everyone: $(g_\sigma \rightsquigarrow_e x \multimap H \rightsquigarrow_t S(x)) \multimap H \rightsquigarrow_t every(person, S)$

left: $g_\sigma \rightsquigarrow_e X \multimap f_\sigma \rightsquigarrow_t leave(X)$

In the **everyone** premise the higher-order variable S ranges over the possible scope meanings of the quantifier, with lower-case x acting as a traditional first-order variable "placeholder" within the scope. H ranges over LFG structures corresponding to the meaning of the entire generalized quantifier.[1]

A meaning for (1) can be derived by applying the linear version of modus ponens, during which (unlike classical logic) the first premise **everyone** "consumes" the second premise **left**. This deduction, along with the substitutions $H \mapsto f_\sigma$, $X \mapsto x$ and $S \mapsto \lambda x.leave(x)$, produces the final meaning $f_\sigma \rightsquigarrow_t every(person, \lambda x.leave(x))$, which is in this simple case the only reading for the sentence.

One advantage of this deductive style of meaning assembly is that it provides an elegant account of quantifier scoping: each possible scope has a corresponding proof, obviating the need for quantifier storage.

2 Meaning deduction via proof nets

A *proof net* (Girard, 1987) is an undirected, connected graph whose node labels are propositions. A

[1] Here we have simplified the notation of Dalrymple *et al.* somewhat, for example by stripping away the universal quantifier operators from the variables. In this regard, note that the lower-case variables stand for arbitrary constants rather than particular terms, and generally are given limited scope within the antecedent of the premise. Upper-case variables are Prolog-like variables that become instantiated to specific terms within the proof, and generally their scope is the entire premise.

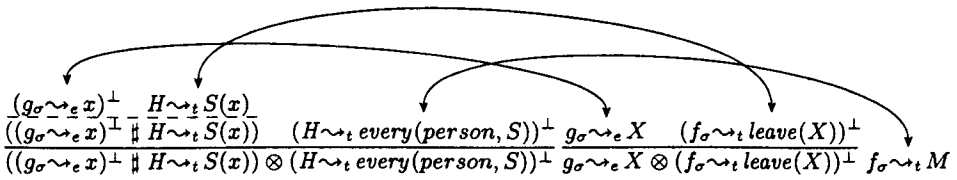


Figure 1: Proof net for *Everyone left*.

theorem of multiplicative linear logic corresponds to only one proof net; thus the manipulation of proof nets is more efficient than sequent deduction, in which the same theorem might have different proofs corresponding to different orderings of the inference steps. A further advantage of proof nets for our purposes is that an invalid meaning deduction, e.g. one corresponding to some spurious scope reading of a particular sentence, can be illustrated by exhibiting its defective graph which demonstrates visually why no proof exists for it. Proof net techniques have also been exploited within the categorial grammar community, for example for reasons of efficiency (Morrill, 1996) and in order to give logical descriptions of certain syntactic phenomena (Lecomte and Retoré, 1995).

In this section we construct a proof net from the premises for sentence (1), showing how to apply higher-order unification to the meaning terms in the process. We then review the $O(n^2)$ algorithm of Gallier (1992) for propositional (multiplicative) linear logic which checks whether a given proof net is valid, i.e. corresponds to a proof. The complete process for assembling a meaning from its premises will be shown in four steps: (1) rewrite the premises in a normalized form, (2) assemble the premises into a graph, (3) connect together the positive ("producer") and negative ("consumer") meaning terms, unifying them in the process, and (4) test whether the resulting graph encodes a proof.

2.1 Step 1: set up the sequent

Since our goal is to derive, from the premises of sentence (1), a meaning M for the f-structure f of the entire sentence, what we seek is a proof of the form

$$\mathbf{everyone} \otimes \mathbf{left} \vdash f_\sigma \rightsquigarrow_t M.$$

Glue language semantics has so far been restricted to the *multiplicative* fragment of linear logic, which uses only the multiplicative conjunction operator $\otimes$ (*tensor*) and the linear implication operator $\multimap$. The same fragment is obtained by replacing $\multimap$ with the operators $\parr$ and $^\perp$, where $\parr$ (*par*) is the multiplicative 'or'[2] and $^\perp$ is linear negation and $(A \multimap B) \equiv (A^\perp \parr B)$. Using the version without $\multimap$, we normalize two sided sequents of the form $A_1, \ldots, A_m \vdash B_1, \ldots, B_n$ into right-sided sequents of the form $\vdash A_1^\perp, \ldots, A_m^\perp, B_1, \ldots, B_n$. (In sequent representations of this style, the comma represents $\otimes$ on the left side of the sequent and $\parr$ on the right side.) In our new format, then, the proof takes the form

$$\vdash \mathbf{everyone}^\perp, \mathbf{left}^\perp, f_\sigma \rightsquigarrow_t M.$$

The proof net further requires that sequents be in negation normal form, in which negation is applied only to atomic terms.[3] Moving the negations inward (the usual double-negation and 'de Morgan' properties hold), and displaying the full premises, we obtain the normalized sequent

$$\begin{array}{ll} \vdash & ((g_\sigma \rightsquigarrow_e x)^\perp \parr H \rightsquigarrow_t S(x)) \\ & \otimes (H \rightsquigarrow_t every(person, S))^\perp, \\ & g_\sigma \rightsquigarrow_e X \otimes (f_\sigma \rightsquigarrow_t leave(X))^\perp, \\ & f_\sigma \rightsquigarrow_t M. \end{array}$$

2.2 Step 2: create the graph

The next step is to create a graph whose nodes consist of all the terms which occur in the sequent. That is, a node is created for each literal C and for each negated literal $C^\perp$; a node is created for each compound term $A \otimes B$ or $A \parr B$; and nodes are also created for its subterms A and B. Then, for each node of the form $A \parr B$, we draw a soft edge in the form of a horizontal dashed line connecting it to nodes A and B. For each node of the form $A \otimes B$, we draw a hard edge (solid line) connecting it to nodes A and B. For the example at hand, this produces the graph in Figure 1 (ignoring the curved edges at the top).

[2] This notation is Gallier's (1992).

[3] Note that we refer to noncompound terms as 'literal' or 'atomic' terms because they are atomic from the point of view of the glue language, even though these terms are in fact of the form $S \rightsquigarrow_\tau M$, where S is an expression over LFG structures and M is a type-τ expression in the meaning language.

2.3 Step 3: connect the literals

The final step in assembling the proof net is to connect together the literal nodes at the top of the graph. It is at this stage that unification is applied to the variables in order to assign them the values they will assume in the final meaning. Each different way of connecting the literals and instantiating their variables corresponds to a different reading for the sentence.

For each literal, we draw an edge connecting it to a matching literal of opposite sign; i.e. each literal A is connected to a literal $B^{\perp}$ where A unifies with B. Every literal in the graph must be connected in this way. If for some literal A there exists no matching literal B of opposite sign then the graph does not encode a proof and the algorithm fails.

In this process the unifications apply to whole expressions of the form $S \rightsquigarrow_{\tau} M$, including both variables over LFG structures and variables over meaning terms. For the meaning terms, this requires a limited higher-order unification scheme that produces the unifier $\lambda x.p(x)$ from a second-order term T and a first-order term $p(x)$. As noted by Dalrymple *et al.* (to appear), all the apparatus that is required for their simple intensional meaning language falls within the decidable $l\lambda$ fragment of Miller (1990), and therefore can be implemented as an extension of a first-order unification scheme such as that of Prolog.

For the example at hand, there is only one way to connect the literals (and hence at most one reading for the sentence), as shown in Figure 1. At this stage, the unifications would bind the variables in Figure 1 as follows: $X \mapsto x$, $H \mapsto f_{\sigma}$, $S \mapsto \lambda x.leave(x)$, $M \mapsto every(person, \lambda x.leave(x))$.

2.4 Step 4: test the graph for validity

Finally, we apply Gallier's (1992) algorithm to the connected graph in order to check that it corresponds to a proof. This algorithm recursively decomposes the graph from the bottom up while checking for cycles. Here we present the algorithm informally; for proofs of its correctness and $O(n^2)$ time complexity see (Gallier, 1992).

Base case: If the graph consists of a single link between literals A and $A^{\perp}$, the algorithm succeeds and the graph corresponds to a proof.

Recursive case 1: Begin the decomposition by deleting the bottom-level par nodes. If there is some terminal node $A \parr B$ connected to higher nodes A and B, delete $A \parr B$. This of course eliminates the dashed edge from $A \parr B$ to A and to B, but does not remove nodes A and B. Then run the algorithm on the resulting smaller (possibly unconnected) graph.

Recursive case 2: Otherwise, if no terminal par node is available, find a terminal tensor node to delete. This case is more complicated because not every way of deleting a tensor node necessarily leads to success, even for a valid proof net. Just choose some terminal tensor node $A \otimes B$. If deleting that node results in a single, connected (i.e. cyclic) graph, then that node was not a valid splitting tensor and a different one must be chosen instead, or else halt with failure if none is available. Otherwise, delete $A \otimes B$, which leaves nodes A and B belonging to two unconnected graphs $G1$ and $G2$. Then run the algorithm on $G1$ and $G2$.

This process will be demonstrated in the examples which follow.

3 A glue language treatment of NPI licensing

Ladusaw (1979) established what is now a well-known generalization in semantics, namely that negative polarity lexical items (NPI's, e.g. *any*, *ever*) are licensed within the scope of downward-entailing operators (e.g. *no*, *few*). For example, the NPI *ever* occurs felicitously in a context like *No one ever left* but not in **John ever left.*[4] Ladusaw showed that the status of a lexical item as a NPI or licenser depends on its meaning; i.e. on semantic rather than syntactic or lexical properties. On the other hand, the requirement that NPI's be licensed in order to appear felicitously in a sentence is a constraint on surface syntactic form. So the domain of NPI licensing is really the *interface* between syntax and semantics, where meanings are composed under syntactic guidance.

This section gives an implementation of NPI licensing at the syntax-semantics interface using glue language. No separate proof or interpretation apparatus is required, only modification of the relevant meaning constructors specified in the lexicon.

3.1 Meaning constructors for NPI's

There is a resource-based interpretation of the NPI licensing problem: the negative or decreasing licensing operator must make available a resource, call it ℓ, which will license the NPI's, if any, within its scope. If no such resource is made available the NPI's are unlicensed and the sentence is rejected.

[4] Here we consider only 'rightward' licensing (within the scope of the quantifier), but this approach applies equally well to 'leftward' licensing (within the restriction).

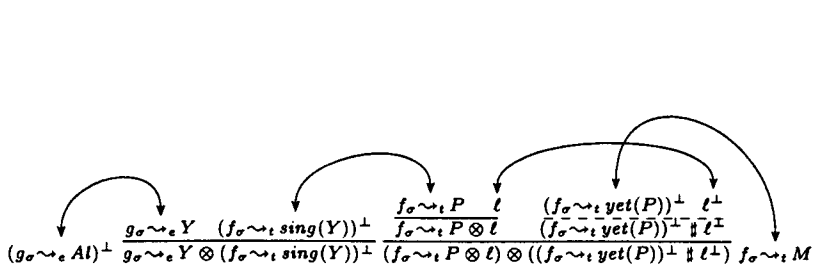


Figure 2: Invalid proof net of $*Al\ sang\ yet$.

The NPI's must be made to require the ℓ resource. The way one implements such a requirement in linear logic is to put the required resource on the left side of the implication operator $\multimap$. This is precisely our approach. However, since the NPI is just 'borrowing' the license, not consuming it (after all, more than one NPI may be licensed, as in *No one ever saw anyone*), we also add the resource to the right hand side of the implication. That is, for a meaning constructor of the form $A \multimap B$, we can make a corresponding NPI meaning constructor of the form

$$(A \otimes \ell) \multimap (B \otimes \ell).$$

For example, the meaning constructor proposed in (Dalrymple et al., 1993) for the sentential modifier *obviously* is

$$\textbf{obviously:} \quad f_\sigma \rightsquigarrow_t P \multimap f_\sigma \rightsquigarrow_t obviously(P).$$

Under this analysis of sentential modification, NPI adverbs such as *yet* or *ever* would take the same form, but with the licensing apparatus added:

$$\textbf{ever:} \quad (f_\sigma \rightsquigarrow_t P \otimes \ell) \multimap (f_\sigma \rightsquigarrow_t ever(P) \otimes \ell).$$

This technique can be readily applied to the other categories of NPI as well. In the case of the NPI quantifier phrase *anyone*[5] the licensing apparatus is added to the earlier template for *everyone* to produce the meaning constructor

$$\textbf{anyone:} \quad (g_\sigma \rightsquigarrow_e x \multimap H \rightsquigarrow_t S(x) \otimes \ell) \multimap (H \rightsquigarrow_t any(person, S) \otimes \ell).$$

The only function of the $\ell \multimap \ell$ pattern inside an NPI is to consume the resource ℓ and then produce it again. However, for this to happen, the resource ℓ will have to be generated by some licenser whose scope includes the NPI, as we show below. If no outside ℓ resource is made available, then the extraneous, unconsumed ℓ material in the NPI guarantees that no proof will be generated. In proof net terms, the output ℓ cannot feed back into the input ℓ without producing a cycle.

We now demonstrate how the deduction is blocked for a sentence containing an unlicensed NPI such as (2).

(2) $*$Al sang yet.

$$f: \begin{bmatrix} \text{PRED} & \text{'SING'} \\ \text{SUBJ} & g:\begin{bmatrix} \text{PRED} & \text{'AL'} \end{bmatrix} \\ \text{MODS} & \{\begin{bmatrix} \text{PRED} & \text{'YET'} \end{bmatrix}\} \end{bmatrix}$$

The relevant premises are

Al:	$g_\sigma \rightsquigarrow_e Al$
sang:	$g_\sigma \rightsquigarrow_e Y \multimap f_\sigma \rightsquigarrow_t sing(Y)$
yet:	$(f_\sigma \rightsquigarrow_t P \otimes \ell) \multimap (f_\sigma \rightsquigarrow_t yet(P) \otimes \ell)$

The graph of (2), shown in Figure 2, does not encode a proof. The reason is shown in Figure 3. At this point in the algorithm, we have deleted the leftmost terminal tensor node. However, the only remaining terminal tensor node cannot be deleted, since doing so would produce a single connected subgraph; the cycle is in the edge from ℓ to $\ell^\perp$. At this point the algorithm fails and no meaning is derived.

3.2 Meaning constructors for NPI licensers

It is clear from the proposal so far that lexical items which license NPI's must make available a ℓ resource within their scope which can be consumed by the NPI. However, that is not enough; a licenser can still occur inside a sentence without an NPI, as in e.g. *No one left*. The resource accounting of linear logic requires that we 'clean up' by consuming any excess ℓ resources in order for the meaning deduction to go through.

Fortunately, we can solve this problem within the licenser's meaning constructor itself. For a lexical category whose meaning constructor is of the form $A \multimap B$, we assign to the NPI licensers of that category the meaning constructor

$$(\ell \multimap (A \otimes \ell)) \multimap B.$$

By its logical structure, being embedded inside another implication, the inner implication here serves

[5] *Any* also has another, so-called 'free choice' interpretation (as in e.g. *Anyone will do*) (Ladusaw, 1979; Kadmon and Landman, 1993), which we ignore here.

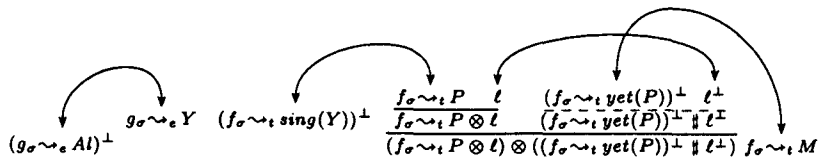


Figure 3: Point of failure. Bottom tensor node cannot be deleted.

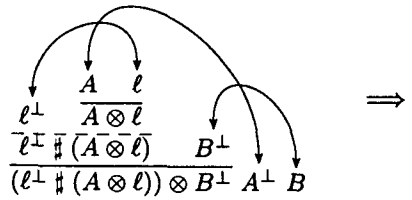
to introduce 'hypothetical' material. All of the NPI licensing occurs within the hypothetical (left) side of the outermost implication. Since the ℓ resource is made available to the NPI only within this hypothetical, it is guaranteed that the NPI is assembled within, and therefore falls under, the scope of the licenser. Furthermore, the formula is 'self cleaning', in that the ℓ resource, even if not used by an NPI, does not survive the hypothetical and so cannot affect the meaning of the licenser in some other way. That is, the licensing constructor $(\ell \multimap (A \otimes \ell)) \multimap B$ can derive all of the same meanings as the nonlicensing version $A \multimap B$.

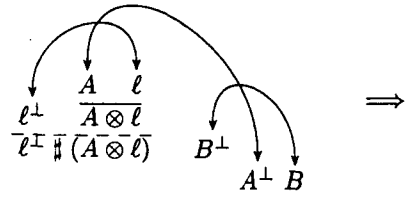
Fact 1 $(\ell \multimap (A \otimes \ell)) \multimap B \vdash A \multimap B$

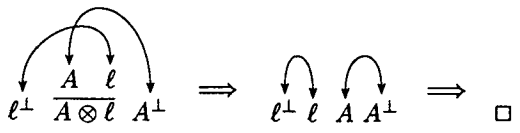
Proof We construct the proof net of the equivalent right-sided sequent

$$\vdash (\ell^{\perp} \parr (A \otimes \ell)) \otimes B^{\perp}, A^{\perp}, B$$

and then test that it is valid.







This self-cleaning property means that a licensing resource ℓ is exactly that—a license. Within the scope of the licenser, the ℓ is available to be used once, several times (in a "chain" of NPI's which pass it along), or not at all, as required.[6]

A simple example is provided by the NPI-licensing adverb *rarely*. We modify our sentential adverb template to create a meaning constructor for *rarely* which licenses an NPI within the sentence it modifies.

rarely: $(\ell \multimap (f_{\sigma} \rightsquigarrow_t P \otimes \ell)) \multimap f_{\sigma} \rightsquigarrow_t rarely(P)$

The case of licensing quantifier phrases such as *nobody* and *few students* follows the same pattern. For example, *nobody* takes the form

nobody: $((g_{\sigma} \rightsquigarrow_e x \otimes \ell) \multimap (H \rightsquigarrow_t S(x) \otimes \ell)) \multimap H \rightsquigarrow_t no(person, S).$

We can now derive a meaning for sentence (3), in which *nobody* and *anyone* play the roles of licenser and NPI, respectively.

(3) Nobody saw anyone.

$$f: \begin{bmatrix} \text{PRED} & \text{'SEE'} \\ \text{SUBJ} & g: [\text{PRED} \quad \text{'NOBODY'}] \\ \text{OBJ} & h: [\text{PRED} \quad \text{'ANYONE'}] \end{bmatrix}$$

Normally, a sentence with two quantifiers would generate two different scope readings—in this case, (4) and (5).

(4) $f_{\sigma} \rightsquigarrow_t no(person, \lambda x.any(person, \lambda y.see(x, y)))$

(5) $f_{\sigma} \rightsquigarrow_t any(person, \lambda y.no(person, \lambda x.see(x, y)))$

However, Ladusaw's generalization is that NPI's are licensed *within the scope* of their licensers. In fact, the semantics of *any* prevent it from taking wide scope in such a case (Kadmon and Landman, 1993; Ladusaw, 1979, p. 96-101). Our analysis, then, should derive (4) but block (5).

[6]This multiple-use effect can be achieved more directly using the exponential operator !; however this unnecessary step would take us outside of the multiplicative fragment of linear logic and preclude the proof net techniques described earlier.

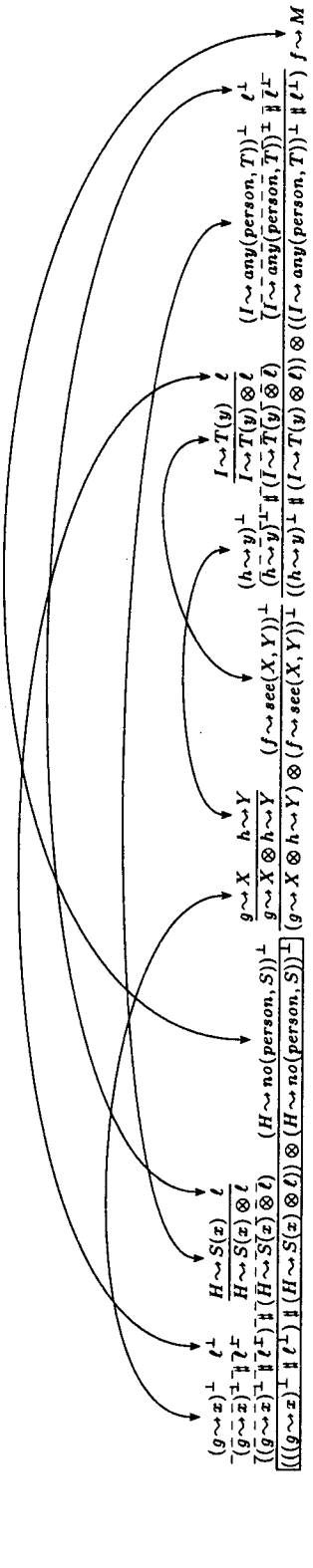


Figure 4: Valid proof net for *Nobody saw anyone* where *no* has wide scope (reading (4)). The initial splitting tensor is shown boxed (bottom left).

The premises are

nobody:	$((g_\sigma \rightsquigarrow_e x \otimes \ell) \multimap (H \rightsquigarrow_t S(x) \otimes \ell)) \multimap H \rightsquigarrow_t no(person, S)$
saw:	$(g_\sigma \rightsquigarrow_e X \otimes h_\sigma \rightsquigarrow_e Y) \multimap f_\sigma \rightsquigarrow_t see(X, Y)$
anyone:	$(h_\sigma \rightsquigarrow_e y \multimap I \rightsquigarrow_t T(y) \otimes \ell) \multimap (I \rightsquigarrow_t any(person, T) \otimes \ell)$

The proof net for reading (4) is shown in Figure 4.[7] As required, the net in Figure 4, corresponding to wide scope for *no*, is valid. The first step in the proof of Figure 4 is to delete the only available splitting tensor, which is boxed in the figure. A second way of linking the positive and negative literals in Figure 4 produces a net which corresponds to (5), the spurious reading in which *any* has wide scope. In that graph, however, all three of the available terminal tensor nodes produce a single, connected (cyclic) graph if deleted, so decomposition cannot even begin and the algorithm fails. Once again, it is the licensing resources which are enforcing the desired constraint.

4 Categorial grammar approaches

The ℓ atom used here is somewhat analogous to the (negative) lexical 'monotonicity markers' proposed by Sánchez Valencia (1991; 1995) and Dowty (1994) for categorial grammar. In these approaches, categories of the form A/B are marked with monotonicity properties, i.e. as A^+/B^+, A^+/B^-, A^-/B^+, or A^-/B^-, and similarly for left-leaning categories of the form $A\backslash B$. Then monotonicity constraints can be enforced using category assignments like the following from (Dowty, 1994):

no:	$\left\{ \begin{array}{l} (S^+/VP^-)/CN^- \\ (S^-/VP^+)/CN^+ \end{array} \right\}$
any:	$(S^-/VP^-)/CN^-$
ever:	VP^-/VP^-

Sánchez Valencia and Dowty, however, are less concerned with the distribution of NPI's than they are with using monotonicity properties to characterize valid inference patterns, an issue which we have ignored here. Hence their work emphasizes *logical* polarity, where an odd number of negative marks indicates negative polarity, and an even number of negatives cancel each other to produce positive polarity. For example, the category of **no** above "flips" the polarity of its argument. By contrast, our system, like Ladusaw's (1979) original proposal, is what Dowty (1994, p. 134-137) would call "intuitionistic":

[7] The subscripts have been stripped from the formulas in order to save space in the diagram.

since multiple negative contexts do not cancel each other out, we permit doubly-licensed NPI's as in *Nobody rarely sees anyone*. To handle such cases, while at the same time accounting for monotonic inference properties, Dowty (1994) proposes a double-marking framework whereby categories like A^{-}_{-}/B^{+}_{-} are marked for both logical polarity and syntactic polarity.

5 Conclusion

We have elaborated on and extended slightly the 'glue language' approach to semantics of Dalrymple *et al.* It was shown how linear logic proof nets can be used for efficient natural-language meaning deductions in this framework. We then presented a glue language treatment of negative polarity licensing which ensures that NPI's are licensed within the semantic scope of their licensers, following (Ladusaw, 1979). This system uses no new global rules or features, nor ambiguous lexical entries, but only the addition of ℓ's to the relevant items within the lexicon. The licensing takes place precisely at the syntax-semantics interface, since it is implemented entirely in the interface glue language. Finally, we noted briefly some similarities and differences between this system and categorial grammar 'monotonicity marking' approaches.

6 Acknowledgements

I'm grateful to Mary Dalrymple, John Lamping and Stanley Peters for very helpful discussions of this material. Vineet Gupta, Martin Kay, Fernando Pereira and four anonymous reviewers also provided helpful comments on several points. All remaining errors are naturally my own.

References

Mary Dalrymple, John Lamping, and Vijay Saraswat. 1993. LFG semantics via constraints. In *Proceedings of the 6th Meeting of the European Association for Computational Linguistics*, University of Utrecht, April.

Mary Dalrymple, John Lamping, Fernando Pereira, and Vijay Saraswat. 1994. A deductive account of quantification in LFG. In Makoto Kanazawa, Christopher J. Piñón, and Henriette de Swart, editors, *Quantifiers, Deduction, and Context.* CSLI Publications, Stanford, CA.

Mary Dalrymple, John Lamping, Fernando Pereira, and Vijay Saraswat. To appear. Quantifiers, anaphora, and intensionality. *Journal of Logic, Language and Information.*

David Dowty. 1994. The role of negative polarity and concord marking in natural language reasoning. In Mandy Harvey and Lynn Santelmann, editors, *Proceedings of SALT IV*, pages 114–144, Ithaca, NY. Cornell University.

Jean Gallier. 1992. Constructive logics. Part II: Linear logic and proof nets. MS, Department of Computer and Information Science, University of Pennsylvania.

Jean-Yves Girard. 1987. Linear logic. *Theoretical Computer Science*, 50.

Nirit Kadmon and Fred Landman. 1993. Any. *Linguistics and Philosophy 16*, pages 353–422.

William A. Ladusaw. 1979. *Polarity Sensitivity as Inherent Scope Relations.* Ph.D. thesis, University of Texas, Austin. Reprinted in Jorge Hankamer, editor, *Outstanding Dissertations in Linguistics.* Garland, 1980.

Alain Lecomte and Christian Retoré. 1995. Pomset logic as an alternative categorial grammar. In Glyn V. Morrill and Richard T. Oehrle, editors, *Formal Grammar.* Proceedings of the Conference of the European Summer School in Logic, Language, and Information, Barcelona.

Dale A. Miller. 1990. A logic programming language with lambda abstraction, function variables and simple unification. In Peter Schroeder-Heister, editor, *Extensions of Logic Programming*, Lecture Notes in Artificial Intelligence. Springer-Verlag.

Glyn V. Morrill. 1996. Memoisation of categorial proof nets: parallelism in categorial processing. In V. Michele Abrusci and Claudia Casadio, editors, *Proceedings of the Roma Workshop on Proofs and Linguistic Categories*, Rome.

Victor Sánchez Valencia. 1991. *Studies on Natural Logic and Categorial Grammar.* Ph.D. thesis, University of Amsterdam.

Victor Sánchez Valencia. 1995. Parsing-driven inference: natural logic. *Linguistic Analysis*, 25(3-4):258–285.