Efficient Linear Logic Meaning Assembly

Vineet Gupta
Caelum Research Corporation
NASA Ames Research Center
Moffett Field CA 94035
vgupta@ptolemy.arc.nasa.gov

John Lamping
Xerox PARC
3333 Coyote Hill Road
Palo Alto CA 94304 USA
lamping@parc.xerox.com

1 Introduction

The "glue" approach to semantic composition in Lexical-Functional Grammar uses linear logic to assemble meanings from syntactic analyses (Dalrymple et al., 1993). It has been computationally feasible in practice (Dalrymple et al., 1997b). Yet deduction in linear logic is known to be intractable. Even the propositional tensor fragment is NP complete(Kanovich, 1992). In this paper, we investigate what has made the glue approach computationally feasible and show how to exploit that to efficiently deduce underspecified representations.

In the next section, we identify a restricted pattern of use of linear logic in the glue analyses we are aware of, including those in (Crouch and Genabith, 1997; Dalrymple et al., 1996; Dalrymple et al., 1995). And we show why that fragment is computationally feasible. In other words, while the glue approach could be used to express computationally intractable analyses, actual analyses have adhered to a pattern of use of linear logic that is tractable.

The rest of the paper shows how this pattern of use can be exploited to efficiently capture all possible deductions. We present a conservative extension of linear logic that allows a reformulation of the semantic contributions to better exploit this pattern, almost turning them into Horn clauses. We present a deduction algorithm for this formulation that yields a compact description of the possible deductions. And finally, we show how that description of deductions can be turned into a compact underspecified description of the possible meanings.

Throughout the paper we will use the illustrative sentence "every gray cat left". It has functional structure

$$(1)\quad f{:}\left[\begin{array}{ll} \text{PRED} & \text{`LEAVE'} \\ \text{SUBJ} & g{:}\left[\begin{array}{ll} \text{PRED} & \text{`CAT'} \\ \text{SPEC} & \text{`EVERY'} \\ \text{MODS} & \{\left[\begin{array}{ll}\text{PRED} & \text{`GRAY'}\end{array}\right]\} \end{array}\right] \end{array}\right]$$

and semantic contributions

$$\begin{array}{rl}
\textbf{leave} & :\forall x.\, g_\sigma \rightsquigarrow x \multimap f_\sigma \rightsquigarrow leave(x) \\
\textbf{cat} & :\forall x.\, (g_\sigma\ \text{VAR}) \rightsquigarrow x \multimap (g_\sigma\ \text{RESTR}) \rightsquigarrow cat(x) \\
\textbf{gray} & :\forall P.\, [\forall x.\, (g_\sigma\ \text{VAR}) \rightsquigarrow x \multimap (g_\sigma\ \text{RESTR}) \rightsquigarrow P(x)] \\
& \quad \multimap [\forall x.\, (g_\sigma\ \text{VAR}) \rightsquigarrow x \\
& \qquad \multimap (g_\sigma\ \text{RESTR}) \rightsquigarrow gray(P)(x)] \\
\textbf{every} & :\forall H, R, S. \\
& \quad [\forall x.\, (g_\sigma\ \text{VAR}) \rightsquigarrow x \multimap (g_\sigma\ \text{RESTR}) \rightsquigarrow R(x)] \\
& \quad \otimes [\forall x.\, g_\sigma \rightsquigarrow x \multimap H \rightsquigarrow S(x)] \\
& \quad \multimap H \rightsquigarrow every(R, S)
\end{array}$$

For our purposes, it is more convenient to follow (Dalrymple et al., 1997a) and separate the two parts of the semantic contributions: use a lambda term to capture the meaning formulas, and a type to capture the connections to the f-structure. In this form, the contributions are

$$\begin{array}{rl}
\textbf{leave}: & \lambda x.leave(x) : g_\sigma \multimap f_\sigma \\
\textbf{cat}: & \lambda x.cat(x) : (g_\sigma\ \text{VAR}) \multimap (g_\sigma\ \text{RESTR}) \\
\textbf{gray}: & \lambda P.\lambda x.gray(P)(x) : \\
& \quad ((g_\sigma\ \text{VAR}) \multimap (g_\sigma\ \text{RESTR})) \\
& \qquad \multimap (g_\sigma\ \text{VAR}) \multimap (g_\sigma\ \text{RESTR}) \\
\textbf{every}: & \lambda R.\lambda S.every(R, S) : \\
& \quad \forall H.\, (((g_\sigma\ \text{VAR}) \multimap (g_\sigma\ \text{RESTR})) \\
& \qquad \otimes (g_\sigma \multimap H)) \\
& \qquad \multimap H
\end{array}$$

With this separation, the possible derivations are determined solely by the "types", the connections to the f-structure. The meaning is assembled by applying the lambda terms in accordance with a proof of a type for the sentence. We give the formal system behind this approach, $\mathcal{C}$ in Figure 1 — this is a different presentation of the system given in (Dalrymple et al., 1997a), adding the two standard rules for tensor, using pairing for meanings. For the types, the system merely consists of the linear logic rules for the glue fragment.

We give the proof for our example in Figure 2, where we have written the types only, and have omitted the trivial proofs at the top of the tree. The meaning *every(gray(cat),left)* may be assembled by putting the meanings back in according to the rules of $\mathcal{C}$ and η-reduction.

$$\begin{array}{c} M : A \vdash_{\mathcal{C}} M' : A \\ \text{where } M \equiv_{\alpha,\eta} M' \end{array} \qquad \frac{\Gamma, P, Q, \Delta \vdash_{\mathcal{C}} R}{\Gamma, Q, P, \Delta \vdash_{\mathcal{C}} R}$$

$$\frac{\Gamma, M : A[B/X] \vdash_{\mathcal{C}} R}{\Gamma, M : \forall X.A \vdash_{\mathcal{C}} R} \qquad \frac{\Gamma \vdash_{\mathcal{C}} M : A[Y/X]}{\Gamma \vdash_{\mathcal{C}} M : \forall X.A}(Y \text{ new})$$

$$\frac{\Gamma \vdash_{\mathcal{C}} N : A \qquad \Delta, M[N/x] : B \vdash_{\mathcal{C}} R}{\Gamma, \Delta, \lambda x.M : A \multimap B \vdash_{\mathcal{C}} R}$$

$$\frac{\Gamma, y : A \vdash_{\mathcal{C}} M[y/x] : B}{\Gamma \vdash_{\mathcal{C}} \lambda x.M : A \multimap B}(y \text{ new})$$

$$\frac{\Gamma, M : A, N : B \vdash R}{\Gamma, \langle M, N\rangle : A \otimes B \vdash R} \qquad \frac{\Gamma \vdash M : A \qquad \Delta \vdash N : B}{\Gamma, \Delta \vdash \langle M, N\rangle : A \otimes B}$$

Figure 1: The system $\mathcal{C}$. M, N are meanings, and x, y are meaning variables. A, B are types, and X, Y are type variables. P, Q, R are formulas of the kind $M : A$. Γ, Δ are multisets of formulas.

2 Skeleton references and modifier references

The terms that describe atomic types, terms like g_σ and $(g_\sigma\ \textsc{var})$, are *semantic structure references*, the *type* atoms that connect the semantic assembly to the syntax. There is a pattern to how they occur in glue analyses, which reflects their function in the semantics.

Consider a particular type atom in the example, such as g_σ. It occurs once positively in the contribution of "every" and once negatively in the contribution of "leave". A slightly more complicated example, the type $(g_\sigma\ \textsc{restr})$ occurs once positively in the contribution of "cat", once negatively in the contribution of "every", and once each positively and negatively in the contribution of "gray".

The pattern is that every type atom occurs once positively in one contribution, once negatively in one contribution, and once each positively and negatively in zero or more other contributions. (To make this generalization hold, we add a negative occurrence or "consumer" of f_σ, the final meaning of the sentence.) This pattern holds in all the glue analyses we know of, with one exception that we will treat shortly. We call the independent occurrences the *skeleton* occurrences, and the occurrences that occur paired in a contribution *modifier* occurrences.

The pattern reflects the functions of the lexical entries in LFG. For the type that corresponds to a particular f-structure, the idea is that, the entry corresponding to the head makes a positive skeleton contribution, the entry that subcategorizes for the f-structure makes a negative skeleton contribution, and modifiers on the f-structure make both positive and negative modifier contributions.

Here are the contributions for the example sentence again, with the occurrences classified. Each occurrence is marked positive or negative, and the skeleton occurrences are underlined.

$$\begin{array}{rl} \textbf{leave}: & \underline{g_\sigma}^{-} \multimap \underline{f_\sigma}^{+} \\ \textbf{cat}: & \underline{(g_\sigma\ \textsc{var})}^{-} \multimap \underline{(g_\sigma\ \textsc{restr})}^{+} \\ \textbf{gray}: & ((g_\sigma\ \textsc{var})^{+} \multimap (g_\sigma\ \textsc{restr})^{-}) \\ & \quad \multimap (g_\sigma\ \textsc{var})^{-} \multimap (g_\sigma\ \textsc{restr})^{+} \\ \textbf{every}: & \forall H.\ (((\underline{(g_\sigma\ \textsc{var})}^{+} \multimap \underline{(g_\sigma\ \textsc{restr})}^{-}) \\ & \quad \otimes (\underline{g_\sigma}^{+} \multimap H^{-})) \\ & \quad \multimap H^{+} \end{array}$$

This pattern explains the empirical tractability of glue inference. In the general case of multiplicative linear logic, there can be complex combinatorics in matching up positive and negative occurrences of literals, which leads to NP-completeness (Kanovich, 1992). But in the glue fragment, on the other hand, the only combinatorial question is the relative ordering of modifiers. In the common case, each of those orderings is legal and gives rise to a different meaning. So the combinatorics of inference tends to be proportional to the degree of semantic ambiguity. The complexity per possible reading is thus roughly linear in the size of the utterance.

But, this simple combinatoric structure suggests a better way to exploit the pattern. Rather than have inference explore all the combinatorics of different modifier orders, we can get a single underspecified representation that captures all possible orders, without having to

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{cat} \vdash (g_\sigma\ \textsc{var}) \multimap (g_\sigma\ \textsc{restr}) \qquad (g_\sigma\ \textsc{var}) \multimap (g_\sigma\ \textsc{restr}) \vdash (g_\sigma\ \textsc{var}) \multimap (g_\sigma\ \textsc{restr})}{\mathbf{cat}, ((g_\sigma\ \textsc{var}) \multimap (g_\sigma\ \textsc{restr})) \multimap (g_\sigma\ \textsc{var}) \multimap (g_\sigma\ \textsc{restr}) \vdash (g_\sigma\ \textsc{var}) \multimap (g_\sigma\ \textsc{restr})}}{\mathbf{gray}, \mathbf{cat} \vdash (g_\sigma\ \textsc{var}) \multimap (g_\sigma\ \textsc{restr}) \qquad \mathbf{leave} \vdash g_\sigma \multimap f_\sigma}}{\mathbf{gray}, \mathbf{cat}, \mathbf{leave} \vdash ((g_\sigma\ \textsc{var}) \multimap (g_\sigma\ \textsc{restr})) \otimes (g_\sigma \multimap f_\sigma) \qquad f_\sigma \vdash f_\sigma}}{\mathbf{gray}, \mathbf{cat}, \mathbf{leave}, (((g_\sigma\ \textsc{var}) \multimap (g_\sigma\ \textsc{restr})) \otimes (g_\sigma \multimap f_\sigma)) \multimap f_\sigma \vdash f_\sigma}}{\mathbf{every}, \mathbf{gray}, \mathbf{cat}, \mathbf{leave} \vdash f_\sigma}
$$

Figure 2: Proof of "Every gray cat left", omitting the lambda terms

explore them.

The idea is to do a preliminary deduction involving just the skeleton, ignoring the modifier occurrences. This will be completely deterministic and linear in the total length of the formulas. Once we have this skeletal deduction, we know that the sentence is well-formed and has a meaning, since modifier occurrences essentially occur as instances of the identity axiom and do not contribute to the type of the sentence. Then the system can determine the meaning terms, and describe how the modifiers can be attached to get the final meaning term. That is the goal of the rest of the paper.

3 Conversion toward horn clauses

The first hurdle is that the distinction between skeleton and modifier applies to atomic types, not to entire contributions. The contribution of "every", for example, has skeleton contributions for g_σ, $(g_\sigma\ \textsc{var})$, and $(g_\sigma\ \textsc{restr})$, but modifier contributions for H. Furthermore, the nested implication structure allows no nice way to disentangle the two kinds of occurrences. When a deduction interacts with the skeletal g_σ in the hypothetical it also brings in the modifier H.

If the problematic hypothetical could be converted to Horn clauses, then we could get a better separation of the two types of occurrences. We can approximate this by going to an indexed linear logic, a conservative extension of the system of Figure 1, similar to Hepple's system(Hepple, 1996).

To handle nested implications, we introduce the type constructor $A\{B\}$, which indicates an A whose derivation made use of B. This is similar to Hepple's use of indices, except that we indicate dependence on types, rather than on indices. This is sufficient in our application, since each such type has a unique positive skeletal occurrence.

We can eliminate problematic nested implications by translating them into this construct, in accordance with the following rule:

For a nested hypothetical at top level that has a mix of skeleton and modifier types:

$$M : (A \multimap B) \multimap C$$

replace it with

$$x : A, \quad M : (B\{A\} \multimap C)$$

where x is a new variable, and reduce complex dependency formulas as follows:

1. Replace $A\{B \multimap C\}$ with $A\{C\{B\}\}$.
2. Replace $(A \multimap B)\{C\}$ with $A \multimap B\{C\}$.

The semantics of the new type constructors is captured by the additional proof rule:

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma, x : A \vdash \lambda x.M : B\{A\}}$$

The translation is sound with respect to this rule:

Theorem 1 *If Γ is a set of sentences in the unextended system of Figure 1, A is a sentence in that system, and Γ' results from Γ by applying the above conversion rules, then $\Gamma \vdash A$ in the system of Figure 1 iff $\Gamma' \vdash A$ in the extended system.*

The analysis of pronouns present a different problem, which we discuss in section 5. For all other glue analyses we know of, these conversions are sufficient to separate items that mix interaction and modification into statements of the form $\mathcal{S}$, $\mathcal{M}$, or $\mathcal{S} \multimap \mathcal{M}$, where $\mathcal{S}$ is pure skeleton and $\mathcal{M}$ is pure modifier. Furthermore, $\mathcal{M}$ will be of the form $A \multimap A$, where A may be a formula, not just an atom. In other words, the type of the modifier will be an identity axiom. The modifier will consume some meaning and produce a modified meaning of the same type.

In our example, the contribution of "every", can be transformed by two applications of the nested hypothetical rule to

$$\begin{array}{l} \textbf{every} : \lambda R.\lambda S.\mathit{every}(R,S) : \\ \qquad \forall H.\, (g_\sigma \text{ RESTR})\{(g_\sigma \text{ VAR})\} \\ \qquad\quad \multimap H\{g_\sigma\} \multimap H \\ x : (g_\sigma \text{ VAR}) \\ y : g_\sigma \end{array}$$

Here, the last two sentences are pure skeleton, producing $(g_\sigma \text{ VAR})$ and g_σ, respectively. The first is of the form $\mathcal{S} \multimap \mathcal{M}$, consuming $(g_\sigma \text{ RESTR})$, to produce a pure modifier.

While the rule for nested hypotheticals could be generalized to eliminate all nested implications, as Hepple does, that is not our goal, because that does remove the combinatorial combination of different modifier orders. We use the rule only to segregate skeleton atoms from modifier atoms. Since we want modifiers to end up looking like the identity axiom, we leave them in the $A \multimap A$ form, even if A contains further implications. For example, we would not apply the nested hypothetical rule to simplify the entry for **gray** any further, since it is already in the form $A \multimap A$.

Handling intensional verbs requires a more precise definition of skeleton and modifier. The type part of an intensional verb contribution looks like $(\forall F.(h_\sigma \multimap F) \multimap F) \multimap g_\sigma \multimap f_\sigma$ (Dalrymple et al., 1996).

First, we have to deal with the small technical problem that the $\forall F$ gets in the way of the nested hypothetical translation rule. This is easily resolved by introducing a skolem constant, S, turning the type into $((h_\sigma \multimap S) \multimap S) \multimap g_\sigma \multimap f_\sigma$. Now, the nested hypothetical rule can be applied to yield $(h_\sigma \multimap S)$ and $S\{S\{h_\sigma\}\} \multimap g_\sigma \multimap f_\sigma$.

But now we have the interesting question of whether the occurrences of the skolem constant, S, are skeleton or modifier. If we observe how S resources get produced and consumed in a deduction involving the intensional verb, we find that $(h_\sigma \multimap S)$ produces an S, which may be modified by quantifiers, and then gets consumed by $S\{S\{h_\sigma\}\} \multimap g_\sigma \multimap f_\sigma$. So unlike a modifier, which takes an existing resource from the environment and puts it back, the intentional verb places the initial resource into the environment, allows modifiers to act on it, and then takes it out. In other words, the intensional verb is acting like a combination of a skeleton producer and a skeleton consumer.

So just because an atom occurs twice in a contribution doesn't make the contribution a modifier. It is a modifier if its atoms must interact with the outside, rather than with each other. Roughly, paired modifier atoms function as $f \multimap f$, rather than as $f \otimes f^\perp$, as do the S atoms of intensional verbs.

Stated precisely:

Definition 2 *Assume two occurrences of the same type atom occur in a single contribution. Convert the formula to a normal form consisting of just $\otimes$, $⅋$, and $\perp$ on atoms by converting subformulas $A \multimap B$ to the equivalent $A^\perp ⅋ B$, and then using DeMorgan's laws to push all $\perp$'s down to atoms. Now, if the occurrences of the same type atom occur with opposite polarity and the connective between the two subexpressions in which they occur is $⅋$, then the occurrences are modifiers. All other occurrences are skeleton.*

For the glue analyses we are aware of, this definition identifies exactly one positive and one negative skeleton occurrence of each type among all the contributions for a sentence.

4 Efficient deduction of underspecified representation

In the converted form, the skeleton deductions can be done independently of the modifier deductions. Furthermore, the skeleton deductions are completely trivial, they require just a linear time algorithm: since each type occurs once positively and once negatively, the algorithm just resolves the matching positive and negative skeleton occurrences. The result is several deductions starting from the contributions, that collectively use all of the contributions. One of the deductions produces a meaning for f_σ, for the whole f-structure. The others produce pure modifiers — these are of the form $A \multimap A$. For

Lexical contributions in indexed logic:

$$
\begin{aligned}
\textbf{leave}:&\quad \lambda x.leave(x) : g_\sigma \multimap f_\sigma\\
\textbf{cat}:&\quad \lambda x.cat(x) : (g_\sigma\ \text{VAR}) \multimap (g_\sigma\ \text{RESTR})\\
\textbf{gray}:&\quad \lambda P.\lambda x.gray(P)(x) : ((g_\sigma\ \text{VAR}) \multimap (g_\sigma\ \text{RESTR})) \multimap (g_\sigma\ \text{VAR}) \multimap (g_\sigma\ \text{RESTR})\\
\textbf{every}_1:&\quad \lambda R.\lambda S.every(R,S) : \forall H.\ (g_\sigma\ \text{RESTR})\{(g_\sigma\ \text{VAR})\} \multimap H\{g_\sigma\} \multimap H\\
\textbf{every}_2:&\quad x : (g_\sigma\ \text{VAR})\\
\textbf{every}_3:&\quad y : g_\sigma
\end{aligned}
$$

The following can now be proved using the extended system:

$$
\begin{gathered}
\textbf{gray} \vdash \lambda P.\lambda x.gray(P)(x) : ((g_\sigma\ \text{VAR}) \multimap (g_\sigma\ \text{RESTR})) \multimap (g_\sigma\ \text{VAR}) \multimap (g_\sigma\ \text{RESTR})\\
\textbf{every}_2, \textbf{cat}, \textbf{every}_1 \vdash \lambda S.every(\lambda x.cat(x), S) : \forall H.\ H\{g_\sigma\} \multimap H\\
\textbf{every}_3, \textbf{leave} \vdash leave(y) : f_\sigma
\end{gathered}
$$

Figure 3: Skeleton deductions for "Every gray cat left".

the example sentence, the results are shown in Figure 3.

These skeleton deductions provide a compact representation of all possible complete proofs. Complete proofs can be read off from the skeleton proofs by interpolating the deduced modifiers into the skeleton deduction. One way to think about interpolating the modifiers is in terms of proof nets. A modifier is interpolated by disconnecting the arcs of the proof net that connect the type or types it modifies, and reconnecting them through the modifier. Quantifiers, which turn into modifiers of type $\forall F.F \multimap F$, can choose which type they modify.

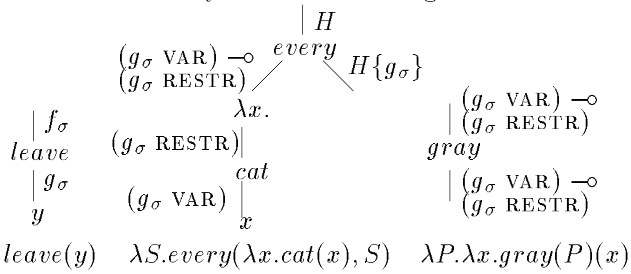
Not all interpolations of modifiers are legal, however. For example, a quantifier must outscope its noun phrase. The indices of the modifier record these limitations. In the case of the modifier resulting from "every cat", $\forall H.H\{g_\sigma\} \multimap H$, it records that it must outscope "every cat" in the $\{g_\sigma\}$. The indices determine a partial order of what modifiers must outscope other modifiers or skeleton terms.

In this particular example, there is no choice about where modifiers will act or what their relative order is. In general, however, there will be choices, as in the sentence "someone likes every cat", analyzed in Figure 4.

To summarize so far, the skeleton proofs provide a compact representation of all possible deductions. Particular deductions are read off by interpolating modifiers into the proofs, subject to the constraints. But we are usually more interested in all possible meanings than in all possible deductions. Fortunately, we can extract a compact representation of all possible meanings from the skeleton proofs.

We do this by treating the meanings of the skeleton deductions as trees, with their arcs annotated with the types that correspond to the types of values that flow along the arcs. Just as modifiers were interpolated into the proof net links, now modifiers are interpolated into the links of the meaning trees. Constraints on what modifiers must outscope become constraints on what tree nodes a modifier must dominate.

Returning to our original example, the skeleton deductions yield the following three trees:



Notice that higher order arguments are reflected as structured types, like $(g_\sigma\ \text{VAR}) \multimap (g_\sigma\ \text{RESTR})$. These trees are a compact description of the possible meanings, in this case the one possible meaning. We believe it will be possible to translate this representation into a UDRS representation(Reyle, 1993), or other similar representations for ambiguous sentences.

We can also use the trees directly as an underspecified representation. To read out a particular meaning, we just interpolate modifiers into the arcs they modify. Dependencies on a

The functional structure of "Someone likes every cat".

$$f{:}\left[\begin{array}{ll} \textsc{pred} & \text{`like'} \\ \textsc{subj} & h{:}\left[\begin{array}{ll}\textsc{pred} & \text{`someone'}\end{array}\right] \\ \textsc{obj} & g{:}\left[\begin{array}{ll}\textsc{pred} & \text{`cat'} \\ \textsc{spec} & \text{`every'}\end{array}\right] \end{array}\right]$$

The lexical entries after conversion to indexed form:

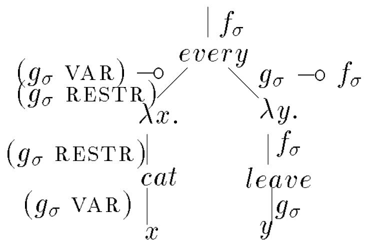
$$\begin{array}{rl} \mathbf{like}: & \lambda x.\lambda y.like(x,y) : (h_\sigma \otimes g_\sigma) \multimap f_\sigma \\ \mathbf{cat}: & \lambda x.cat(x) : (g_\sigma\ \textsc{var}) \multimap (g_\sigma\ \textsc{restr}) \\ \mathbf{someone_1}: & z : h_\sigma \\ \mathbf{someone_2}: & \lambda S.some(person,\ S) : \forall H.\ H\{h_\sigma\} \multimap H \\ \mathbf{every_1}: & \lambda R.\lambda S.every(R,S) : \forall H.\ (g_\sigma\ \textsc{restr})\{(g_\sigma\ \textsc{var})\} \multimap H\{g_\sigma\} \multimap H \\ \mathbf{every_2}: & x : (g_\sigma\ \textsc{var}) \\ \mathbf{every_3}: & y : g_\sigma \end{array}$$

From these we can prove:

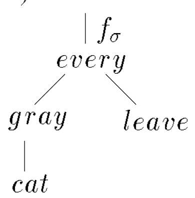
$$\mathbf{someone_1}, \mathbf{every_3}, \mathbf{like} \vdash like(z,y) : f_\sigma$$
$$\mathbf{someone_2} \vdash \lambda S.some(person,\ S) : \forall H.\ H\{h_\sigma\} \multimap H$$
$$\mathbf{every_2}, \mathbf{cat}, \mathbf{every_1} \vdash \lambda S.every(cat, S) : \forall H.\ H\{g_\sigma\} \multimap H$$

Figure 4: Skeleton deductions for "Someone likes every cat"

modifier's type indicate that a lambda abstraction is also needed. So, when "every cat" modifies the sentence meaning, its antecedent, instantiated to $f_\sigma\{g_\sigma\}$ indicates that it lambda abstracts over the variable annotated with g_σ and replaces the term annotated f_σ. So the result is:



Similarly "gray" can modify this by splicing it into the line labeled $(g_\sigma\ \textsc{var}) \multimap (g_\sigma\ \textsc{restr})$ to yield (after η-reduction, and removing labels on the arcs).

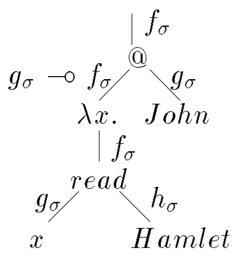


This gets us the expected meaning $every(gray(cat), leave)$.

In some cases, the link called for by a higher order modifier is not directly present in the tree, and we need to do λ-abstraction to support it. Consider the sentence "John read Hamlet quickly". We get the following two trees from the skeleton deductions:

$read(John, Hamlet)$ $\qquad$ $\lambda P.\lambda x.quickly(P)(x)$

There is no link labeled $g_\sigma \multimap f_\sigma$ to be modified. The left tree however may be converted by λ-abstraction to the following tree, which has a required link. The @ symbol represents λ application of the right subtree to the left.



Now *quickly* can be interpolated into the link labeled $g_\sigma \multimap f_\sigma$ to get the desired meaning $quickly(read(Hamlet), John)$, after η-reduction. The cases where λ-abstraction is required can be detected by scanning the modifiers and noting whether the links to be modified are present in the skeleton trees. If not, λ-abstraction can introduce them into the underspecified representation. Furthermore, the introduction is unavoidable, as the link will be present in any final meaning.

5 Anaphora

As mentioned earlier, anaphoric pronouns present a different challenge to separating skeleton and modifier. Their analysis yields types like $f_\sigma \multimap (f_\sigma \otimes g_\sigma)$ where g_σ is skeleton and f_σ is modifier. We sketch how to separate them.

We introduce another type constructor $(B)A$, informally indicating that A has not been fully used, but is also used to get B.

This lets us break apart an implication whose right hand side is a product in accordance with the following rule:

For an implication that occurs at top level, and has a product on the right hand side that mixes skeleton and modifier types:

$$\lambda x.\langle M, N\rangle : A \multimap (B \otimes C)$$

replace it with

$$\lambda x.M : (C)A \multimap B, \quad N : C$$

The semantics of this constructor is captured by the two rules:

$$\frac{M_1 : A_1, \ldots, M_n : A_n \vdash M : A}{M_1 : (B)A_1, \ldots, M_n : (B)A_n \vdash M : (B)A}$$

$$\frac{\Gamma,\ M_1 : (B)A,\ M_2 : B \vdash N : C}{\Gamma',\ M_1' : A,\ M_2' : B \vdash N' : C}$$

where the primed terms are obtained by replacing free x's with what was applied to the $\lambda x.$ in the deduction of $(B)A$

With these rules, we get the analogue of Theorem 1 for the conversion rule. In doing the skeleton deduction we don't worry about the $(B)A$ constructor, but we introduce constraints on modifier positioning that require that a hypothetical dependency can't be satisfied by a deduction that uses only part of the resource it requires.

6 Acknowledgements

We would like to thank Mary Dalrymple, John Fry, Stefan Kaufmann, and Hadar Shemtov for discussions of these ideas and for comments on this paper.

References

Richard Crouch and Josef van Genabith. 1997. How to glue a donkey to an f-structure, or porting a dynamic meaning representation into LFG's linear logic based glue-language semantics. Paper to be presented at the Second International Workshop on Computational Semantics, Tilburg, The Netherlands, January 1997.

Mary Dalrymple, John Lamping, and Vijay Saraswat. 1993. LFG semantics via constraints. In *Proceedings of the Sixth Meeting of the European ACL*, pages 97–105, University of Utrecht. European Chapter of the Association for Computational Linguistics.

Mary Dalrymple, John Lamping, Fernando C. N. Pereira, and Vijay Saraswat. 1995. Linear logic for meaning assembly. In *Proceedings of CLNLP*, Edinburgh.

Mary Dalrymple, John Lamping, Fernando C. N. Pereira, and Vijay Saraswat. 1996. Intensional verbs without type-raising or lexical ambiguity. In Jerry Seligman and Dag Westerståhl, editors, *Logic, Language and Computation*, pages 167–182. CSLI Publications, Stanford University.

Mary Dalrymple, Vineet Gupta, John Lamping, and Vijay Saraswat. 1997a. Relating resource-based semantics to categorial semantics. In *Proceedings of the Fifth Meeting on Mathematics of Language (MOL5)*, Schloss Dagstuhl, Saarbrücken, Germany.

Mary Dalrymple, John Lamping, Fernando C. N. Pereira, and Vijay Saraswat. 1997b. Quantifiers, anaphora, and intensionality. *Journal of Logic, Language, and Information*, 6(3):219–273.

Mark Hepple. 1996. A compilation-chart method for linear categorical deduction. In *Proceedings of COLING-96*, Copenhagen.

Max I. Kanovich. 1992. Horn programming in linear logic is NP-complete. In *Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 200–210, Los Alamitos, California. IEEE Computer Society Press.

Uwe Reyle. 1993. Dealing with ambiguities by underspecification: Construction, representation, and deduction. *Journal of Semantics*, 10:123–179.