Context Change and Underspecification in Glue Language Semantics*

Dick Crouch
Speech Research Unit,
DRA Malvern, UK
crouch@signal.dra.hmg.gb

Josef van Genabith
School of Computer Applications
Dublin City University, Eire
Josef.Van.Genabith@CompApp.DCU.IE

August 1996

1 Introduction

Context Update

The interpretation of a natural language utterance typically both depends on the context of utterance, and has the effect of updating that context. A common response to this context-sensitivity has been to view the meaning of a sentence as an update function on contexts. The meaning can be derived in a strictly compositional way on the basis of the syntactic structure of the sentence. But depending on the context to which the update function is applied, the truth-conditional significance of the sentence may vary.

This paper is a preliminary exploration of an alternative way of approaching context update. Instead of composing the meaning of a sentence out of its parts in a context-independent way, and having the meaning update context, the composition itself draws on and updates context. The result is a context-independent meaning that represents the truth-conditional import of the sentence as uttered in its particular context.

We start with the use by Mary Dalrymple and others [Dalrymple *et al* 1993a, 1993b, 1995a, 1995b] of a fragment of linear logic as a 'glue language' for piecing together the meanings of individual words and constituents within a deductive setting. In computer science, linear logic has come to be seen as a useful tool for modelling update and change. It is therefore natural to wonder whether linear logic can be used to model contextual update. As we will shortly see, Dalrymple *et al*'s glue language treatment already models the update of *meaning assignments* to constituents. We intend to use basically the same mechanism to additionally update *context assignments*, and provide and E-type analysis of certain anaphors.

Underspecification

Another common, though to some extent orthogonal, reaction to the context-sensitivity of natural language interpretation has been a concern with underspecification. In the absence of context, sentences (e.g. like *They won*) can be highly ambiguous. It is computationally infeasible to generate all possible interpretations and use context purely as a filter to weed

*This is a shortened working draft of a longer paper in preparation. Comments are welcome

out the implausible ones. The aim has instead been to produce semantic representations (on the basis of syntactic structure) that leave contextual factors unspecified, though capable of further specification.

An increasingly favoured approach to underspecification (e.g. [Crouch 1995, Bos 1995] introduces an extra layer of indirection. On the basis of syntactic structure, sentences are translated into representations that describe a set of possible meanings, or ways of constructing meanings. This description can then be further refined to take account of context.

This view of underspecification sits quite happily with the use of glue languages. Syntactic analysis results in a set of meaning constructors expressed in the glue language. These are used as premises to a proof whose conclusion must be a single assignment of a meaning to the sentence as a whole. In the case of sentences involving quantifiers, several different proofs resulting in distinct meaning assignments may be possible. In this sense, the glue language premises can be seen as describing a set of ways in which the meaning of the sentence can be constructed. What the glue language account so far lacks is (a) a way of refining these descriptions, and (b) a way of accounting for the contribution of context to the meaning derived.

Two Hypotheses

We can sum up the preceding discussion by means of the following two hypotheses about contextual-sensitivity:

1. Syntactic structure (and lexical semantics) alone is not sufficient to determine the meaning of a sentence.
   (a) Strict compositionality (homomorphism from syntax to meaning) therefore fails.
   (b) But we can preserve a form of compositionality:
   The meaning of the whole depends (i) on the meaning of the parts and (ii) on the way in which they are combined
   (c) Syntax and lexical semantics does not always fully specify the meaning of the parts (e.g. pronouns), or the way they are combined (e.g. scope ambiguities).
   (d) Context also contributes to the composition of meaning, and fills in the gaps left by syntax

2. Semantic composition updates context, in addition to depending on the contribution from context.

2 Meaning Update

Linear logic is a resource conscious logic. Unlike material implication ($\rightarrow$), linear implication ($\multimap$) does *not* support inferences of the form

$$A \otimes (A \multimap B) \vdash A \otimes B$$

(where $\otimes$ is linear multiplicative conjunction). Both the A and the $A \multimap B$ get used up in inferring B, so that there is no A left to conjoin with the inferred B.

This can be used to model update as follows. Suppose the $\rightsquigarrow$ is an (uninterpreted) symbol associating constituents with their meanings. Thus

$$\sigma \rightsquigarrow \text{sleep(john)}$$

associates the meaning 'John slept' with a constituent σ. Suppose, for the sake of argument, the we have a sentential modifier whose meaning constructor is

$$\forall \phi.\ \sigma \rightsquigarrow \phi \multimap \sigma \rightsquigarrow \text{probably}(\phi)$$

This can be used to update the meaning assigned to σ through the application of modus ponens

$$\sigma \rightsquigarrow \text{probably(sleep(john))}$$

However, in applying the rule of modus ponens, both the original meaning assignment to σ and the meaning constructor of the modifier are consumed. Consequently, we can't conclude that both $\sigma \rightsquigarrow$ sleep(john) and $\sigma \rightsquigarrow$ probably(sleep(john)). The original meaning of σ has been updated, and is no longer available.

2.1 Scope Ambiguities

As a fuller introduction to glue language semantics, we will describe the treatment of the scope ambiguity in the sentence

> Every candidate appointed a manager.

as presented in [Dalrymple et al. 1995b].

Glue language premises (meaning constructors) are associated with nodes in a semantic projection of the f-structure for the sentence. For ease of exposition, we have given the nodes mnemonic names, though no significance attaches to the names.

- *s*: the projection of the verb *appoint*, the verb phrase and the sentence as a whole
- *subj*: the projection of the subject noun phrase *every candidate*
- *subj.var*: the projection of the subject determiner, *every*
- *subj.restr*: the projection of the subject noun, *candidate*
- *obj*: the projection of the object noun phrase *a manager*
- *obj.var*: the projection of the object determiner, *a*
- *obj.restr*: the projection of the object noun, *manager*

Lexical meaning constructors derived from the semantic projection are (recall that in building the projection, variables over nodes in the general lexical entries will have been instantiated to refer to specific nodes in the projection)

- **appoint**:
  $\forall X, Y.\ (\mathit{subj} \rightsquigarrow X \otimes \mathit{obj} \rightsquigarrow Y) \multimap s \rightsquigarrow \mathit{appoint}(X, Y)$
- **every**:

$$\begin{aligned}\forall scope,R,S.\ &(\forall x.subj.var \rightsquigarrow x \multimap subj.restr \rightsquigarrow R(x)) \\ &\otimes (\forall x.subj \rightsquigarrow x \multimap scope \rightsquigarrow S(x)) \\ \multimap\ &scope \rightsquigarrow every(R,S)\end{aligned}$$

- **a**:

$$\begin{aligned}\forall scope,R,S.\ &(\forall x.obj.var \rightsquigarrow x \multimap obj.restr \rightsquigarrow R(x)) \\ &\otimes (\forall x.obj \rightsquigarrow x \multimap scope \rightsquigarrow S(x)) \\ \multimap\ &scope \rightsquigarrow a(R,S)\end{aligned}$$

- **candidate**:
  $\forall X.\ subj.var \rightsquigarrow X \multimap subj.restr \rightsquigarrow candidate(X)$

- **manager**:
  $\forall X.\ obj.var \rightsquigarrow X \multimap obj.restr \rightsquigarrow manager(X)$

In these meaning constructors, the variable *scope* ranges over nodes in the semantic projection, upper case X, Y, R, S are higher-order variables ranging over meanings, and the lower case x is a first order variable.

The meaning constructors serve as premises to a proof, the aim of which is to derive a single expression of the form $s \rightsquigarrow M$, where M is a possible meaning for the sentence s. In order to arrive a single meaning assignment as the conclusion, it is necessary to use up all the premises to the proof, for otherwise we would be left with a conjunctive conclusion.

As we will see, in this case the premises allow two conclusions of this form to be derived:

$$s \rightsquigarrow every(candidate, \lambda v.a(manager, \lambda u.appoint(v,u)))$$

and

$$s \rightsquigarrow a(manager, \lambda u.every(candidate, \lambda v.appoint(v,u)))$$

There might seem to be an air of lurking inconsistency here, in that we have shown that two incompatible meanings can be the meaning of s. But this is not so. Just because we can prove that $s \rightsquigarrow M_1$ and also that $s \rightsquigarrow M_2$, for incompatible meanings M_1 and M_2, the resource consumption of the linear logic glue language does not permit us to conclude $s \rightsquigarrow M_1 \otimes s \rightsquigarrow M_2$. In establishing one meaning for s we use up premises, so that these cannot be reused in the middle of the same proof to derive an alternative, incompatible meanings for some of the nodes.

How do these proofs proceed? We start off with a (multiplicative) conjunction of the premises, thus ensuring that each premise can only be used once. Rules of inference used include modus ponens and universal instantiation. The currying equivalence between $(A \otimes B) \multimap C$, $A \multimap (B \multimap C)$ and $B \multimap (A \multimap C)$ is also employed.

Both proofs contain the same steps to combine subject and object determiners and nouns. Taking **every** and **candidate**, note that with a universal instantiation taking R to *candidate*, and renaming X as x, the noun constructor can identified with the first conjunct in the determiners antecedent. We can thus combine them using the currying equivalence and applying modus ponens (similarly for the object noun phrase) to form:

- **every-candidate**:
  $\forall scope, S.\ (\forall x.\ subj \rightsquigarrow x \multimap scope \rightsquigarrow S(x)) \multimap scope \rightsquigarrow every(candidate, S)$

- **a-manager**:
  $\forall scope, S.\ (\forall x.\ obj \rightsquigarrow x \multimap scope \rightsquigarrow S(x)) \multimap scope \rightsquigarrow a(manager, S)$

In forming these constructors, the original premises are consumed.

The proofs diverge when it comes to combining the noun phrases with the verb. We can consume the verb meaning in one of two ways by currying, to get either of:

- **appoint1**:
  $\forall X.\ subj \rightsquigarrow X \multimap (\forall Y.\ obj \rightsquigarrow Y \multimap s \rightsquigarrow appoint(X, Y))$

- **appoint2**:
  $\forall Y.\ obj \rightsquigarrow Y \multimap (\forall X.\ subj \rightsquigarrow X \multimap s \rightsquigarrow appoint(X, Y))$

We can then combine **appoint1** with **a-manager** by instantiating universal variables as follows

$$scope \mapsto s,\ S \mapsto \lambda v.appoint(X, v)$$

and identifying x with Y This instantiates **a-manager** to

- **a-manager1**:
  $(\forall Y.\ obj \rightsquigarrow Y \multimap s \rightsquigarrow appoint(X, Y)) \multimap s \rightsquigarrow a(manager, \lambda v.appoint(X, v))$

which can then be combined with **appoint1** through transitivity of implication to give:

- **appoint1-a-manager**:
  $\forall X.\ subj \rightsquigarrow X \multimap s \rightsquigarrow a(manager, \lambda v.appoint(X, v)$

This can then be combined with **every-manager** via the substitutions

$$scope \mapsto s,\ S \mapsto \lambda u.a(manager, \lambda v.appoint(u, v))$$

and identifying x with X, to give

- **every-candidate-appointed-a-manager**:
  $s \rightsquigarrow every(candidate, \lambda v.a(manager, \lambda u.appoint(v, u)))$

The alternative proceeds by combining **every-candidate** with **appoint2** (using the substitutions $scope \mapsto s,\ S \mapsto \lambda u.appoint(u, Y)$ and identifying x with X) to get

- **every-candidate-appointed2**:
  $\forall Y.\ obj \rightsquigarrow Y \multimap s \rightsquigarrow every(candidate, \lambda u.appoint(u, Y))$

which can then be combined with **a-manager** to give the alternative scoping, using the substitutions $scope \mapsto s,\ S \mapsto \lambda v.every(candidate, \lambda u.appoint(u, v))$.

The two scopings are the only meanings for s that can be derived from the lexical premises.

3 Context Update

We will now introduce a context assignment, $\hookrightarrow$, analogous to the meaning assignment $\rightsquigarrow$. Nodes in semantic projections are assigned both a meaning and a contextual contribution. The meanings and/or contextual contributions of some nodes may depend on the meanings and/or contextual contributions of other nodes; and the meaning/context constructors may also update these assignments by means of the linear implication, $\multimap$.

The main difference brought about by introducing context assignments in addition to meaning assignments is in the desired result of glue language derivations. Previously, we needed to establish conclusions of the form

$$\Gamma \vdash s \rightsquigarrow M$$

where Γ was the entire set of lexical premises, and a single meaning assignment occurred on the right hand side. Now, however, the desired result is

$$\Gamma, \Delta \vdash s \rightsquigarrow M \otimes \delta_0 \otimes \ldots \otimes \delta_i$$

Here, Γ is a set of lexical premises as before. Δ is a set of context assignments, and $\delta_0 \ldots \delta_i$ is a set of possibly updated context assignments.

An important feature of context assignments that needs to be captured is that, unlike meaning assignments, they do not need to be used exactly once. They may not be used at all (e.g. an NP that is never referred back to by an anaphor), or they may be used repeatedly. However, when contextual contributions are used repeatedly, each re-use is liable to update the contextual contribution made. It is for this reason that it is inappropriate to make use of linear logic's 'of course' modality, !, to allow zero or repeated use of context assignments. Use of the modality would undo the effects of any update. Instead we modify the form of the desired results of glue language derivations to allow any number of context assignments to occur on the right hand side of the turnstile. These output context assignment may either be passed on directly from the input assignments in Δ, or may be updated versions of assignments in Δ, or context assignments to entirely new constituents. The output assignments $\delta_0, \ldots, \delta_i$ will form the input assignments Δ for the next sentence to be interpreted.

3.1 A Simplified Illustration

The use of context assignments can be best illustrated by looking at the simple mini-discourse

A man walked. He whistled.

for which we will give an E-type analysis of the pronoun.

Taking the first sentence, *A man walked*, we have the following lexical constructors

- **a**:
  $\forall scope, R, S.\ \forall x.subj1.var \rightsquigarrow x \multimap subj1.restr \rightsquigarrow R(x))$
  $\otimes \forall x.subj1 \rightsquigarrow x \multimap (scope \rightsquigarrow S(x) \otimes subj1 \hookrightarrow \lambda y.y = x)$
  $\multimap scope \rightsquigarrow a(R, S) \otimes subj1 \hookrightarrow \lambda y.R(y) \wedge S(y)$

- **man**:
  $\forall X.\ subj1.var \rightsquigarrow X \multimap subj1.restr \rightsquigarrow man(X)$

- **walked**:
  $\forall X, Y.\ subj1 \rightsquigarrow X \multimap (s1 \rightsquigarrow walk(X) \otimes subj1 \hookrightarrow \lambda y.y = X)$

Starting with a null context, we can combine these constructor to show that

a, man, walked $\vdash$
$s1 \rightsquigarrow a(man,walk) \otimes subj1 \hookrightarrow \lambda y.man(y) \wedge walk(y)$

In other words, as well as building the meaning of the s node, we have also constructed a contextual assignment for the subject noun phrase, $subj$, which associates it with the property of being a man that walked.

For the second sentence, *He whistled*, we have the constructors

- **he**:
  $\forall scope, R, S.\ \forall x.subj2 \rightsquigarrow x \multimap (scope \rightsquigarrow S(x) \otimes subj2 \hookrightarrow \lambda y.y = x)$
  $\multimap subj1 \hookrightarrow P$
  $\multimap scope \rightsquigarrow exists(P, S)$
  $\otimes subj2 \hookrightarrow \lambda y.P(y) \wedge S(y)$
  $\otimes \forall Q.\ subj2 \hookrightarrow Q \multimap (subj \hookrightarrow Q \otimes subj1 \hookrightarrow Q)$

- **whistled**:
  $\forall X, Y.\ subj2 \rightsquigarrow X \multimap (s2 \rightsquigarrow whistle(X) \otimes subj2 \hookrightarrow \lambda y.y = X)$

Here we have assumed some process of anaphoric linking in building up the semantic projection, the links the pronoun $subj2$ to the antecedent contextual contribution of $subj1$. The pronoun existentially quantifies over the property associated with its antecedent.

These constructors can be put together, along with the context assignment for the first sentence to show that

he, whistled, $subj1 \hookrightarrow \lambda y.man(y) \wedge walk(y), \vdash$
$s2 \rightsquigarrow exists(\lambda x.man(x) \wedge walk(x),\ \lambda x.whistle(x))$
$\otimes subj1 \hookrightarrow \lambda y.man(y) \wedge walk(y) \wedge whistle(y)$
$\otimes subj2 \hookrightarrow \lambda y.man(y) \wedge walk(y) \wedge whistle(y)$

Notice how the context assignment to the antecedent NP has been updated in addition to producing a new assignment for the pronoun. The meaning derived for the second sentence entails that derived for the first sentence, and is truth-conditionally equivalent to a DPL like meaning for the entire discourse,

$$\exists x.(man(x) \wedge walk(x)) \wedge whistle(x)$$

As suggested in the introduction, a glue language treatment of context update differs from dynamic approaches to semantics in its location of update. Dynamic and update semantics treats the meanings resulting from semantic composition as update functions on context. But in the example above, it is the (glue language) composition itself that brings about the update.

3.2 Interactions with Scope

The simplified illustration of context update above unfortunately does not readily extend to handling interactions of context with scope.[1] To see how context assignments need to be revised to handle scope, we will reconsider the sentence *Every candidate appointed a manager.*

- **appoint**:
  $\forall X.\ subj \rightsquigarrow X \multimap subj \rightsquigarrow X \otimes subj \hookrightarrow \lambda x.x = X \otimes$
  $\forall Y.\ obj \rightsquigarrow Y \multimap obj \rightsquigarrow Y \otimes obj \hookrightarrow \lambda y.y = Y \otimes$
  $\forall X, Y.\ (subj \rightsquigarrow X \otimes obj \rightsquigarrow Y) \multimap$
  $s \rightsquigarrow appoint(X, Y)$
  $\otimes(\forall P, Q.\ (subj \hookrightarrow P \otimes obj \hookrightarrow Q) \multimap s \hookrightarrow \exists x, y.P(x) \wedge Q(y) \wedge appoint(x, y))$
  $\otimes \top$

- **every candidate**:
  $\forall scope, S, S', \Gamma, \Delta.$
  $(\forall x, P.\ subj \rightsquigarrow x \multimap$
  $scope \rightsquigarrow S(x) \otimes subj \hookrightarrow P(x)$
  $\otimes subj \hookrightarrow P(x) \multimap (\Gamma \multimap scope \hookrightarrow S'(x))$
  $\otimes \Delta)$
  $\multimap (scope \rightsquigarrow every(candidate, S)$
  $\otimes \Gamma \multimap (\Gamma \otimes subj \hookrightarrow \lambda x.candidate(x) \wedge S'(x))$
  $\otimes \Gamma \multimap scope \hookrightarrow every(candidate, S')$
  $\otimes \Delta)$

- **a manager**:
  $\forall scope, S, S', \Gamma, \Delta.$
  $(\forall x, P.\ obj \rightsquigarrow x \multimap$
  $scope \rightsquigarrow S(x) \otimes obj \hookrightarrow P(x)$
  $\otimes obj \hookrightarrow P(x) \multimap (\Gamma \multimap scope \hookrightarrow S'(x))$
  $\otimes \Delta)$
  $\multimap (scope \rightsquigarrow a(manager, S)$
  $\otimes \Gamma \multimap (\Gamma \otimes obj \hookrightarrow \lambda x.manager(x) \wedge S'(x))$
  $\otimes \Gamma \multimap scope \hookrightarrow a(manager, S')$
  $\otimes \Delta)$

The meaning constructor for **appoint** sets up two initial conditional context assignments for the subject and object noun phrases (the property denoting the same thing as whatever the noun phrases do). Dependent on the meaning of the subject and object, it also sets up a meaning assignment for the s node, and a contextual assignment to s which is dependent on whatever contextual properties are assigned to the subject and object NPs. $\top$ is a vacuously true proposition, present solely for irritating technical reasons.

The meaning constructor for **every candidate** is a conditional that updates (a) the meaning assignment to the NP's scope, (b) the contextual assignment of the NP, and (c) the conditional context assignment to the scope. The variables Γ and Δ range over glue language propositions. Γ corresponds to the contextual assignments on which the context

[1] A more brutal way of putting this is that it can only cope with simple sentences built using intransitive verbs!

assignment to the scope depends (excluding the assignment to the NP itself). Δ ranges over all other contextual assignments that may have been built up, and these are transferred to the consequent of the conditional without update. The constructor for **a manager** is analogous.

We can rearrange the meaning constructor for **appoint** to get

- **appoint1**:
  $\forall X.\ subj \rightsquigarrow X \multimap subj \rightsquigarrow X \otimes subj \hookrightarrow \lambda x.x = X \otimes$
  $\forall X.\ subj \rightsquigarrow X \multimap$
  $\quad \forall Y.\ obj \rightsquigarrow Y \multimap$
  $\quad\quad obj \hookrightarrow \lambda x.y = Y \otimes$
  $\quad\quad s \rightsquigarrow appoint(X, Y) \otimes$
  $\quad\quad obj \hookrightarrow \lambda y.y = Y \multimap (subj \hookrightarrow P \multimap s \hookrightarrow \exists xy.\ P(X) \wedge y = Y \wedge appoint(x, y)) \otimes$
  $\quad\quad \top$

The second and third conjuncts of **appoint** have the form $A \multimap (A \otimes B)$ and $A \otimes C \multimap D$, from which we can derive $C \multimap (A \multimap B \otimes C)$, where A is $obj \rightsquigarrow Y$, B is $obj \hookrightarrow \lambda y.y = Y$, C is $subj \rightsquigarrow X$, and D is the consequent of the third conditional.

This matches the antecedent of **a manager** (Γ instantiated to $subj \hookrightarrow P$, and Δ to $\top$). Consequently, we can apply modus ponens to get

- **appoint a manager**:
  $\forall X.\ subj \rightsquigarrow X \multimap subj \rightsquigarrow X \otimes subj \hookrightarrow \lambda x.x = X \otimes$
  $\forall X.\ subj \rightsquigarrow X \multimap$
  $\quad s \rightsquigarrow a(manager, \lambda v.appoint(X, v)$
  $\quad \otimes \forall P.\ subj \hookrightarrow P \multimap$
  $\quad\quad subj \hookrightarrow P \otimes obj \hookrightarrow \lambda y.manager(y) \wedge \exists x', y'.P(x') \wedge y' = y \wedge appoint(x', y')$
  $\quad \otimes \forall P.\ subj \hookrightarrow P \multimap s \hookrightarrow a(manager, \lambda y.\exists x', y'.P(x') \wedge y' = y \wedge appoint(x', y'))$
  $\quad \otimes \top$

By a similar process, this can be combined with **every candidate** to give

- **every candidate appoint a manager**:
  $s \rightsquigarrow every(candidate, \lambda u.a(manager, \lambda v.appoint(u, v)$
  $\otimes \forall P.\ subj \hookrightarrow P \multimap$
  $\quad subj \hookrightarrow P \otimes obj \hookrightarrow \lambda y.manager(y) \wedge \exists x', y'.P(x') \wedge y' = y \wedge appoint(x', y')$
  $\otimes s \hookrightarrow every(candidate, \lambda x.a(manager, \lambda y.\exists x', y'.x' = x \wedge y' = y \wedge appoint(x', y')))$
  $\otimes subj \hookrightarrow \lambda x.candidate(x) \wedge a(manager, \lambda y.\exists x', y'.x' = x \wedge y' = y \wedge appoint(x', y'))$
  $\otimes \top$

The context assignments resulting from this derivation have a conditional form, so that the context assignments to narrow scope quantifiers depend on the assignment to wider scope quantifiers.

3.2.1 Bound Variable Anaphora

Giving pronouns a meaning constructor of the general form

- **pro**:
  $\forall scope, S, S', \Gamma, \Delta.$
  $\quad (\forall x, P.\ pro \rightsquigarrow x \multimap$

$scope \rightsquigarrow S(x) \otimes pro \hookrightarrow P(x)$
$\otimes pro \hookrightarrow P(x) \multimap (\Gamma \multimap scope \hookrightarrow S'(x))$
$\otimes \Delta)$
$\multimap \forall Q.ante \hookrightarrow Q$
$\multimap (scope \rightsquigarrow every(candidate, S)$
$\otimes \Gamma \multimap (\Gamma \otimes pro \hookrightarrow \lambda x.candidate(x) \wedge S'(x))$
$\otimes \Gamma \multimap scope \hookrightarrow every(candidate, S')$
$\otimes \forall P.\ pro \hookrightarrow P \multimap (pro \hookrightarrow P \otimes ante \hookrightarrow P)$
$\otimes \Delta)$

we now sketch how bound variable anaphora can be treated. Scoping the pronoun with the antecedent having wide scope will give rise to (shown heavily abbreviated):

- **pro scope**
  $\forall X.\ ante \rightsquigarrow X \multimap ante \rightsquigarrow X \otimes ante \hookrightarrow \lambda x.x = X \otimes$
  $\forall X, P.\ ante \rightsquigarrow X \otimes ante \hookrightarrow P \multimap$
  $scope \rightsquigarrow exists(P, \ldots) \otimes \ldots$

The two conjuncts can be combined to instantiate P to $\lambda x.x = X$ and replace the consequent of the first conditional with that of the second to get

- **pro scope1**
  $\forall X.\ ante \rightsquigarrow X \multimap scope \rightsquigarrow exists(\lambda x.x = X, \ldots)\ otimes \ldots$

At this point, the antecedent can be scoped, binding the variable X introduced by the pronouns antecedent.

If we attempt to give the antecedent narrow scope with respect to the pronoun, scoping the pronoun leads to a conclusion of the form

- **pro ante scope**
  $\forall P.\ ante \hookrightarrow P \multimap$
  $scope \rightsquigarrow quant(R, \lambda x.exists(P, ..x..)) \otimes$
  $\forall Q.\ pro \hookrightarrow Q \multimap (pro \hookrightarrow Q \otimes ante \hookrightarrow ...) \otimes$
  $pro \hookrightarrow ....$

But there is nothing else present to match the antecedent of this conditional (we cannot use the assignment derivable from the consequent, since this depends on the antecedent). Consequently, this scope order does not lead to a derivation giving a conclusion of the correct form.

3.2.2 Donkey Anaphora

For *every farmer who owns a donkey beats it* (where '*every farmer* takes wide scope), the contextual property picked up by the anaphor will depend on the contextual assignment to the noun phrase *every farmer who owns a donkey.* Until this NP is given scope, its contextual property is $\lambda x.x = X$, where X will get bound once the NP is scoped. Thus we derive a meaning

every(λx. farmer(x) ∧ a(donkey, λy. owns(x,y)),
λx. exists(λy. donkey(y) ∧ ∃x1,y1. x1=x ∧ y1=y ∧ own(x',y'),
λy. beat(x,y)))

This is a weak reading of the donkey sentence. The more familiar strong reading can be obtained by using a universal quantifier to bind the antecedent's contextual property. The choice of pronominal quantifiers is beyond the scope of this paper.

4 Underspecification

So far we have sketched how context may contribute to and be updated by semantic composition, which are represented as glue language derivations. We now turn to integrating this with a view of underspecification based on the idea that we have semantic compositions where either (a) the meanings of certain constituents are not fully specified, or (b) the way in which they are combined is not fully specified. In glue language terms, this amounts to (a) having a choice about which contextual assignments to make use of in a derivation, and (b) a choice about the order in which certain steps in the derivation are carried out.

4.1 Scope Specification

To begin with we will confine our attention to underspecification of type (b), as exemplified by scope ambiguity. The two derivations in section 2.1 that lead to different scopings differ as to the order in which the subject and object noun phrase meanings are consumed. If we had some way of further specifying this ordering, we would have a way of further constraining the derivations permitted, and thus a way of specifying scope.

It might be though that a simple ordering on premise usage would be sufficient to constrain scope, but for various reasons this turns out not to be the case. But an ordering over the nodes in the semantic projection suffices.

Each node is a 'channel' controlling how certain semantic elements combine to form meanings, and where each channel term of the form $\mathit{Channel} \rightsquigarrow \mathit{Meaning}$ acts as either a consumer or producer on the specified channel [Dalrymple et al. 1993a]. A successful glue language derivation is one that matches up consumers and producers on all channels, except for one remaining producer on the channel corresponding to the sentence as a whole. The lexical meaning constructors ensure that there are normally only a few orderings of production and consumption on the various channels that meet the requirements. By imposing further ordering constraints on the nodes, one can monotonically eliminate possible derivations.

Scoping a noun phrase matches a consumer with a producer on the NP channel. In terms of the derivations sketched in section 2.1, this means that the channel term for the NP disappears from the rest of the derivation. By constraining the order in which NP channel terms disappear, both relative to each other and to those for other nodes, we constrain the range of derivations.

The reader can check that in the derivations in section 2.1, the order of disappearance of channel terms is

- Wide scope subject NP:

  $s \succ subj \succ obj$

- Wide scope object NP:

  $s \succ obj \succ subj$

The scope of NPs relative to one another is given directly by the node ordering: if $subj \succ obj$ then $subj$ has wide scope over obj. The ordering between NP nodes and the nodes of their immediate scopes is superficially the reverse of what one might expect. If a node s is the immediate scope of an NP np, then the ordering is $s \succ np$. That is, the s node still remains an active channel after the NP is scoped. But for any other prospective scope node, scp, if $np \succ scp$ this means that the meaning of scp is consumed within the primary scope of np, so that np has scope over scp (see [Crouch & Genabith 1996] for examples of this).

4.2 Formal Properties

To show that node orderings really do what we claim above, we need a more precise formulation of glue language derivations, and to establish certain properties of the ordering.

4.2.1 Derivations and Node Orders

Assume sequent style natural deduction rules for the linear logic glue language (derived from [Troelstra 1992])

$$\otimes I\,\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \qquad \otimes E\,\frac{\Gamma \vdash A \otimes B \qquad \Delta, A, B \vdash C}{\Gamma, \Delta \vdash C}$$

$$\multimap I\,\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \qquad \multimap E\,\frac{\Gamma \vdash A \multimap B \qquad \Delta \vdash A}{\Gamma, \Delta \vdash B}$$

$$\forall I\,\frac{\Gamma \vdash A[x/y]}{\Gamma \vdash \forall x\ A} \qquad \forall E\,\frac{\Gamma \vdash \forall x\ A}{\Gamma \vdash A[x/t]}$$

$$\lambda_1\,\frac{\Gamma \vdash A' \qquad A' \rightarrow_\lambda A}{\Gamma \vdash A} \qquad \lambda_2\,\frac{\Gamma \vdash A \qquad A' \rightarrow_\lambda A}{\Gamma \vdash A'}$$

$$Axiom\,\frac{}{A \vdash A}$$

where $\rightarrow_\lambda$ indicates λ-reducibility. The standard side condition applies to universal introduction: y does not occur free in Γ.

A derivation is a tree-like structure of sequents, whose leaves are instances of the axiom sequent $A \vdash A$. (In glue language derivations, there will be one leaf for each lexical premise). Each sequent in the tree must be derivable from those immediately preceding it via one of the above rules of inference. Represent derivations $\mathcal{D}$ as triples $\langle S, >_S, \$ \rangle$ where S is the set of points in the tree, $>_S$ is a transitive, asymmetric ordering over them, and \$ is a function mapping the points onto their corresponding sequents.

Sequents within a glue language derivation will contain literals (channel terms) of the form *Channel* $\rightsquigarrow$ *Meaning*, where *Channel* is a constant referring to some node in the semantic projection. A channel or node is *active* at a certain point s in a derivation if a literal referring to that channel occurs both on the left and the right hand side of s's sequent:

Definition 1 (Active Channel/Node) *A channel c is active at point s in a derivation iff*
a) $\$(s) = \Gamma \vdash \phi$,
b) A literal of the form $c \rightsquigarrow M_1$ *is a subexpression of* Γ, *and*
c) A (possibly different) literal of the form $c \rightsquigarrow M_2$ *is a subexpression of* ϕ

(In the more informal derivations illustrated previously, a channel being active amounts to it not yet having disappeared from the derivation. In the sequent style derivations assumed here, channels that cease to be active may still be referred to in the left hand side of a sequent, Γ, but not in the right hand side).

Given the ordering $>_S$ over points in a derivation, and the definition of an active channel / node, we can now define an ordering over nodes

Definition 2 ($A \succ B$) *A node A remains active in a derivation* $\mathcal{D}$ *after a node B,* $A \succ B$, *iff*
a) There is some point s in $\mathcal{D}$ *at which A is active,*
b) For all points $s' >_S s$, *B is not active at* s', *and*
c) There is some point s'' *s.t.* $s >_S s''$ *at which B is active.*

4.2.2 Properties of the Ordering

We need to show that (i) only scoping an NP moves all the NP's channel terms to the left hand side of the sequent, and (ii) no subsequent derivation steps can move a channel term back to the right hand side. For (ii) we confine our attention to normalized derivations (in the proof theoretic sense), since a detour — an introduction step immediately followed by an elimination step — can always vacuously reintroduce and then eliminate a channel term.

Point (i) follows from the fact that there must be no outstanding dependencies on an NP when it is scoped (as detailed in [Dalrymple et al. 1995b]) and inspection of the canonical form of the final $\multimap$-elimination step involved in scoping:

$$\frac{\Gamma_1 \vdash (\forall x.np \rightsquigarrow x \multimap scp \rightsquigarrow S(x)) \multimap scp \rightsquigarrow Q(R,S) \qquad \Gamma_2 \vdash \forall x.np \rightsquigarrow x \multimap scp \rightsquigarrow S(x)}{\Gamma_1, \Gamma_2 \vdash scp \rightsquigarrow Q(R,S)}$$

Also, given that (a) a single determiner premise introduces a producer on the NP channel, (b) a single premise (corresponding to the lexical head of the constituent subcategorizing for the NP) introduces a consumer on the NP channel, and (c) any other premises dependent on the NP meaning (e.g. anaphors) are both consumers and producers (i.e. have NP channel terms in both the antecedent and consequent of an implication), it follows that only scoping moves all channel terms for an NP to the left-hand side of a sequent.

For point (ii) note that only three rules can add channel terms to the right hand side of a sequent: the axiom $A \vdash A$, $\multimap$-introduction, and $\forall$-elimination (universal instantiation). All the other rules either leave things unchanged, or move channel terms from the right hand to the left hand sides of sequents.

If $A \vdash A$ is used to reintroduce an NP channel term, there needs to be some active consumer on that channel still present. Otherwise it will not be possible to eliminate the channel term again, as demanded by the final conclusion of the derivation, which must be a single channel term for the sentence. But (see [Dalrymple et al. 1995b]) when the NP is scoped there must be no outstanding consumers.

Universal instantiation can reintroduce an NP channel by instantiating a variable introduced by one of the premises. The only variables of the right type to be instantiated to a node in a semantic projection are the '*Scope*' variables. But these should only be instantiated to nodes that could serve as scopes for quantifiers.

With $\multimap$ -introduction we can undo the effects of the $\multimap$ -elimination in the final step of scoping. But as there are no other outstanding consumers on the NP node, we can only eliminate the NP channel term by repeating the $\multimap$ -elimination, introducing a detour in the derivation.

Hence in normalized glue language derivations it is only scoping that moves NP channel terms irrevocably to the left hand side of sequent. So node orderings really do constrain NP scope. By supplementing glue language premises with sets of node orderings (the output of scope resolution), we have a powerful mechanism for underspecifying scope.

It should also be noted that the introduction of context assignments does not affect the order in which meanings are consumed and produced with respect to scoping. Hence, this method of scope specification remains applicable.

4.3 Anaphoric Underspecification

With scope, we have looked at a case of underspecification arising out of the way that otherwise specified meanings are composed together. We now turn briefly to the other case, where the meanings themselves are not fully specified. In glue language terms, this form of underspecification amounts to having a choice about which premises to employ in a derivation.

Since all meaning assignments bar one must be consumed by the derivation, the only scope for premise choice lies in which contextual assignments to use. The meaning constructors for pronouns in section 3 fix the choice of contextual assignment by specifying which node is to provide the contextual property bound by the pronoun. This could be unfixed by having a universal quantification over both the contextual property and the antecedent node to which the property is assigned. We can then allow partial coindexing of anaphors with their antecedents at the level of f-structure. When an anaphor is coindexed this constrains the glue language derivation to make use of a particular premise when scoping the anaphor. Those anaphors that are not coindexed at f-structure can make use of any contextual premise that leads to a legitimate derivation.

4.4 Underspecification with Glue Languages

Given the foregoing, we would like to think of resolving underspecification as being a process of monotonically building up constraints on the set of derivations permitted by the glue language premises. Here we have only considered node-order constraints for scope, and coindexing for anaphors, but a wider variety of constraints would almost certainly be needed.

It should be pointed out that we are not proposing glue languages as a form of underspecified semantic representation. Rather, glue language is to be seen as providing a way of giving a semantics to such representations. That is, the denotation of an underspecified semantic representation is a set of compositions, represented as glue language derivations. From these derivations we can, to be sure, derive a set of possible meanings for the sentence or fragment being represented, but these do not (directly) provide the semantics of the underspecified representation. Given the close relation between f-structure and underspecified

representations like Quasi Logical Form [van Genabith & Crouch 1995], it would be natural to view f-structure (supplemented with such things as a means for representing scope node orderings) as a candidate for an underspecified representation with a glue language semantics.

It should also be said that the glue language semantics does not determine the way in which underspecification is to be resolved. In the case of scope, the range of possible node-orderings is mapped out by the glue language semantics, but it does not dictate particular choices of ordering within this range. Similarly for the resolution / coindexing of anaphors.

It is also worth noting that a glue language semantics does not violate the disjunctive fallacy for underspecification, which holds that a sentence ambiguous between ϕ and ψ means $(\phi \vee \psi)$. Instead it correctly analyzes it as either meaning ϕ or meaning ψ (see p. 4).

5 Further Work

This paper only presents an initial exploration of the use of glue languages for modelling contextual update and underspecification. Much further work obviously needs to be done to establish this approach. For instance, in the realm of anaphora nothing has yet been said about the choice between strong and weak readings of pronouns (which roughly amounts to binding the antecedent's contextual property with a universal or an existential quantifier). Nor has anything been said about forms of anaphora (like one anaphora) that don't assume strict identity between antecedent and anaphor. Another major testing area would be the analysis of ellipsis in a glue language framework.

References

[Bos 1995] J. Bos 1995. Predicate Logic Unplugged In *Proceedings 10th Amsterdam Colloquium.*

[Crouch 1995] R. Crouch 1995. Ellipsis and Quantification: a substitutional approach. In *Proceedings 7th EACL*, pages 229–236.

[Crouch & Genabith 1996] R. Crouch and J. van Genabith 1996 F-structure: A Glue for QLF and UDRT? 1995. In *FraCaS Deliverable D15*, pages 217–254.

[Dalrymple et al. 1993a] M. Dalrymple, A Hinrichs, J. Lamping, and V. Saraswat 1993. The Resource Logic of Complex Predicate Interpretation. Xerox Technical Report ISTL-NLTT-1993-08-03.

[Dalrymple et al. 1995a] M. Dalrymple, A. Kehler, J. Lamping, and V. Saraswat 1995. The Semantics of Resource Sharing in Lexical-Functional Grammar. In *Proceedings 7th EACL*, pages 31–38.

[Dalrymple et al. 1993b] M. Dalrymple, J. Lamping, and V. Saraswat 1993. LFG Semantics via Constraints. In *Proceedings 6th EACL*, pages 97–105.

[Dalrymple et al. 1995b] M.Dalrymple, J. Lamping, F.Pereira, and V.Saraswat 1995. Quantifiers, Anaphora, and Intensionality. cmp-lg/9504029.

[van Genabith & Crouch 1995] J. van Genabith and R. Crouch 1995. Direct and Underspecified Interpretations of LFG f-structures. *Proceedings Coling 96.*

[Troelstra 1992] A. Troelstra 1992. Lecture Notes on Linear Logic. CSLI Lecture Notes, 29.