First-Order Glue

Miltiadis Kokkonidis
Meta Research

Abstract. Glue has evolved significantly during the past decade. Although the recent move to type-theoretic notation was a step in the right direction, basing the current Glue system on system F (second-order λ-calculus) was an unfortunate choice. An extension to two sorts and ad hoc restrictions had to be improvised to avoid inappropriate composition of meanings. As a result, the current system is unnecessarily complicated. A first-order Glue system is proposed. This new system is not only simpler and more elegant as it captures the exact requirements for Glue-style compositionality without ad-hoc improvisations, but it also turns out to be more powerful than the current two-sorted (pseudo-) second-order system. First-order Glue supports all existing Glue analyses as well as new, more elegant and/or more demanding analyses.

Keywords: Glue, first-order linear logic, compositional semantics, syntax-semantics interface.

1. Introduction

The Principle of Compositionality(Janssen, 1997) has played an important role in practically all post-Montague work on formal natural language semantics. Some researchers share Montague's belief that compositionality is an essential semantic principle. Others feel compositionality in the traditional sense is rather limiting and propose alternative semantic principles. Typically Montague-style compositionality is replaced with a more relaxed version.

Dalrymple et al.(1993) argued for a concept of compositionality in LFG(Kaplan and Bresnan, 1982) based primarily on functional, rather than constituent structure, a semantic projection function and linear logic as a Glue language for putting together the meaning of a phrase from the meanings of its parts. The motivation for this line of research is that having a resource logic driving semantic composition fits well with the unordered nature of f-structure, while preserving the fundamental semantic principle (underlying also Montague-style compositionality) that the meaning of a phrase is formed by a combination of the meanings of each and every part of the phrase used exactly once(Klein and Sag, 1985). It follows that a restatement of the semantic versions of the completeness and coherence principles of Kaplan and Bresnan(1982) comes for free with this approach.

Glue has the ability to derive the various readings of a sentence given its semantic atoms. It is essentially a theorem prover and various 'transformations' such as currying/uncurrying and type-lifting are proof steps performed wherever needed offering maximum flexibility while at the same time removing the need for uniform worst-case typing. For example, Glue allows proper names to be treated as simple entities, instead of as second-order predicates which would be required for uniformity with quantified noun phrases. In many ways Glue derivations are similar to type-driven derivations possible in the Lambek-van Benthem tradition of Categorial Grammar, or Type-Logical Categorial Grammar as we will be referring to it(Morril, 1994; Moorgat, 1997). CG deals with issues of syntax and semantics together, which is both the source of its elegance and a potential problem that feeds much of current CG research. Glue only deals with semantic assembly, leaving syntactic issues to other parts of the framework within which it is used. Although, originally designed for LFG, it can also been successfully integrated with other frameworks: LTAG(Frank and van Genabith, 2001), HPSG(Asudeh and Crouch, 2002), CG(Asudeh and Crouch, 2001) and CFG(Asudeh and Crouch, 2001).

The first and foremost challenge for a derivational approach is to get all possible readings of a sentence while avoiding erroneous interpretations. This requires making the right choices on two levels: the formal system and analyses of linguistic data within that system. Sometimes the desire for new analyses that address shortcomings of earlier approaches brings about changes to the formal system usually making it more complicated and less elegant. CG is a popular example.

A remarkable fact about Glue is that both the change to a type-theoretic flavour of Dalrymple et al.(1997a) and the change to a first-order system proposed here were not motivated by a need for more power and new logical operators that would help with certain analyses. The foundational intuitions behind Glue as laid out by Dalrymple et al.(1993) and especially in subsequent work (Dalrymple et al., 1995a, 1995b, 1997b) have successfully captured the requirements of a compositional program of semantics for LFG (and other grammar formalisms) and so have not been the subject of revision although the formal systems that aim to capture the essence of these intuitions have. Indeed, one of the claims in this paper is that the proposed system captures these requirements in the most direct way possible. As a result, instead of a more complicated system, we will get a simpler one which happens to also be more powerful.

While the foundational intuitions behind Glue have changed very little since (Dalrymple et al., 1993) and have remained intact since (Dalrymple et al., 1995a, 1995b), the formal system for Glue has been evolving. The original Glue system (G0) of Dalrymple et al.(1993) was quickly superseded by the system later introduced by Dalrymple et al.(1995a). The system presented there (G1) was in many ways a simplification of G0, as it did away with rules and the use of the

linear logic '!' modality. G1 was the system that brought Glue into the mainstream and much research has been done using it. The latest development was the development of a type-theoretic notation for Glue based on System F (G2). Dalrymple et al.(1997a) aimed to explore the relation between Glue and CG approaches to the syntax-semantics interface and as a result introduced G2, a Glue system closer to systems used in the TLCG camp.

Although G2 was presented as a system that only served as a bridge between the LFG/Glue world and the TLCG world, it later became the prominent Glue system replacing G1 as the system of choice for Glue researchers. The type-theoretic notation was neater, more concise and more readable than the notation of G1. Most importantly, it separated "assembled" meaning from the linear logic types that controlled the derivation following the Curry-Howard isomorphism. Moreover, G2 notation was already familiar to its intended audience as type theoretic notation is commonplace in the field. For the same reason, G2 made Glue accessible to a much wider audience that included logicians and computer scientists. The present paper argues for the next step in the evolution of Glue: an even simpler, first-order system (G3) that improves on the current one (G2).

In Section 2 we see how from a minimal set of assumptions and methodological choices the First-Order Glue approach emerges as a candidate for carrying out a program of compositional[1] semantics. As the idea of a type system for meaning assembly is developed we reach a crossroads. One direction leads to G2 and the other to G3. The G3 direction had never before been considered. In Section 3 we see that for all its System F (second-order typed λ-calculus) notation and claimed heritage G2 is equivalent to a restricted version of the proposed first-order Glue system. Every G2 derivation can be encoded in a fragment of G3. Moreover, the computational properties of G2 are preserved for analyses transferred from G2 to the restricted fragment of G3. In Section 4 we examine the common characteristics of the design of G2 and the design of G3. We also discusses why the various ad hoc innovations of G2 are needed to obtain the behaviour that comes naturally in first-order Glue (G3). In Section 5 we conclude that G3 is a step forward in the evolution of Glue and propose that this step be made.

[1] The term 'compositional' is not used in this paper for a progam of semantics faithful to the letter of Montague's work, but to a broader program of which Montague's work is a particular flavour.

2. Towards first-order Glue

The proposed Glue system is a term assignment system for a fragment of first-order intuitionistic linear logic. Both the design decision to use linear-logic, and the decision to use predicates as types, follow from fairly basic observations and assumptions in the *spirit* of Montague's work. We will see how one can get from those assumptions and observations to the proposed Glue system thus demonstrating the plausibility of the intuitions behind the design of the new and previous Glue systems.

richard walks. (1)

richard walks slowly. (2)

If we assume that a sentence such as (1) is a truth statement then we can try to formalize this by giving the semantic expression that corresponds to it a truth type, which in the Montague tradition (ignoring intensionality) is t. We also make the assumption that there is a systematic process that produces the meaning (or meanings) of a sentence from the semantic contributions of the words that appear in it and certain syntactic structures such as the relative clause (or other factors, such as ironic tone in the speaker's voice, which we will ignore in this paper).

So, we should ask what the semantic contributions of '*richard*' and 'walks' are and if there are any semantic contributions due to special syntactic constructs. A methodological preference in Glue and other similar systems is to put emphasis on the words of a sentence and only search for meaning contributions in syntactic constructs when absolutely necessary. In (1) there are two words. The first, '*richard*' denotes an entity and we may assign the type e to its semantic contribution. Then, assuming no other semantic contribution has been made by the sentence and that our system does not rely on rules that combine for example something of type e and something of, say, type iv (intransitive verb) to give something of type t, it is up to the semantic contribution of 'walks' to return something of type t given something of type e. Reasoning that way we may decide to give the semantic contribution of 'walks' the type $e \to t$. Thus we are making the claim that the semantic contribution of 'walks' is a function from entities to truth values.

We can treat modification in a similar fashion. The difference between (1) and (2) is the adverbial modifier 'slowly'. Given that the semantic contributions of '*richard*' and 'walks' can be combined into something of type t in the way just described and that the meaning of (2) will also be of type t, the semantic contribution of 'slowly' has a type that explains why if it is omitted from (2) the resulting sentence (1) would be well-formed, whereas if '*richard*' or 'walks' were omitted the same would not be true. Some adverbs are sentence modifiers $(t \to t)$ while others are verb phrase modifiers $((e \to t) \to (e \to t))$. Modification in Glue is discussed in more detail by Dalrymple(2001).

Table I. Type assignments based on intuitionistic propositional logic

Category	*Type*
proper name	e
determiner	$(e \to t) \to (e \to t) \to t$
noun	$e \to t$
adjective	$(e \to t) \to (e \to t)$
S adverb	$(e \to t) \to (e \to t)$
VP adverb	$t \to t$
intransitive verb	$e \to t$
transitive verb	$e \times e \to t$

Table I shows a possible type assignment for various word categories. It is by no means exhaustive, but it suffices for our following examples. Later on, we will use these type assignment as the basis for the new, more refined type assignment possible in a type system based on a fragment of linear predicate logic.

$$\begin{aligned} \mathit{richard} &: e \\ \mathit{walk} &: e \to t \end{aligned} \tag{3}$$

According to our analysis, the two elemental semantic contributions (or *semantic constructors* as they are known in Glue literature) (3) of sentence (1) are combined into the typed composite term (4) which gives the meaning of the entire sentence[2] (and verifies its type is t).

$$\mathit{walk\ richard} : t \tag{4}$$

The intended meaning *slowly walk richard*[3] of (2) is also a simple combination through function application of the semantic constructors of its words. But for the composition of the meaning of some sentences function application is not enough.

The cat likes Alonso. (5)

$$\begin{aligned} \mathit{the} &: (e \to t) \to (e \to t) \to t \\ \mathit{cat} &: e \to t \\ \mathit{like} &: e \times e \to t \\ \mathit{alonso} &: e \end{aligned} \tag{6}$$

The only readily available combination by function application of the elemental semantic contributions (6) of sentence (5) is that of the two components of the subject noun phrase which gives the composite meaning contribution of the subject noun phrase which needs a predicate $(e \to t)$ argument in order to return a complete sentence (t).

$$\begin{aligned} \mathit{the\ cat} &: (e \to t) \to t \\ \mathit{like} &: e \times e \to t \\ \mathit{alonso} &: e \end{aligned} \tag{7}$$

[2] We will typically omit parenteses around arguments an write fx instead of $f(x)$.
[3] Function application associates to the left. So $f\ x\ y$ is the same as $(f\ x)\ y$.

In sentence (5) we encounter the semantic contribution of a transitive verb, *like*. We treat such verbs as functions from pair of entities, the subject and the object entity, to sentences. Unfortunately, we only have the object entity, *alonso*, readily available. Given a subject entity, say x, we could form the pair $(x, alonso)$. Then, combining that with *like* we can form the term $like(x, alonso)$. What we have just described, is a function from an unknown subject entity x to the truth value $like(x, alonso)$. In λ-calculus notation we write this as $\lambda x.like(x, alonso)$. Since we now have an appropriate argument for the composite semantic contribution of the subject noun phrase *the cat*, we have a complete sentence:[4]

$$the\ cat\ \lambda x.like(x, alonso) : t \tag{8}$$

The reasoning made in order to reach a composition of the elemental semantic contributions of sentence (5) into its composite semantic representation (8) involved ontology (e and t types) as well as roles. Part of our reasoning for reaching (8) is captured in the following derivation.

$$\dfrac{\dfrac{T \vdash T \quad C \vdash C}{T, C \vdash the\ cat : (e \to t) \to t} \quad \dfrac{\dfrac{L \vdash L \quad \dfrac{x : e \vdash x : e \quad A \vdash A}{A, x : e \vdash (x, alonso) : e \times e}}{L, A, x : e \vdash like(x, alonso) : t}}{L, A \vdash \lambda x.like(x, alonso) : e \to t}}{T, C, L, A \vdash the\ cat\ \lambda x.like(x, alonso) : t}$$

$$\begin{array}{ll} where & T = the : (e \to t) \to (e \to t) \to t \\ & C = cat : e \to t \\ & L = like : e \times e \to t \\ & A = alonso : e \end{array} \tag{9}$$

Above we have a type derivation in the $(\to, \times)$ fragment of the simply typed λ-calculus(Figure 1). It is interesting to note that the type derivation essentially follows the same kind of reasoning we used for composing the given elemental semantic contributions and especially the reasoning that allowed us to build an $e \to t$ function from the elemental semantic contributions for 'like' and 'Alonso'. Traditionally the type system serves as a mechanism for checking well-formedness of the compositionally derived semantic expressions for a sentence, so up to a point it is not surprising that there appears to be a relation between the type derivation for the meaning term of a sentence and the reasoning that lead to the formation of that meaning term.

Up to this point we have not had a systematic way of deriving a composite term representing the meaning of a sentence. Given the inevitably close relation between meaning composition and typing rules it is reasonable to ask if the type system could do more for us than just check the well-formedness of composite meaning terms obtained using some kind of compositional methodology.

[4] A λ-abstraction term extends as much as it can: $\lambda x.E_1\ E_2\ \ldots\ E_n$ is the same as $\lambda x.(E_1\ E_2\ \ldots\ E_n)$, not $(\lambda x.E_1)\ E_2\ \ldots\ E_n$.

$$\frac{\Gamma \vdash E : T}{\Gamma,\ N : T' \vdash E : T} \quad (Weakening)$$

$$\frac{\Gamma,\ N : T,\ N : T' \vdash E : T''}{\Gamma,\ N : T \vdash E : T''} \quad (Contraction)$$

$$\frac{N : T,\ \Gamma,\ N' : T',\ \Gamma' \vdash E : T''}{N' : T',\ \Gamma,\ N : T,\ \Gamma' \vdash E : T''} \quad (Exchange)$$

$$N : T \vdash N : T \quad (Axiom)$$

$$\frac{\Gamma, X : T \vdash E : T'}{\Gamma \vdash \lambda X.E : T \to T'} \quad (\to Intro.)$$

$$\frac{\Gamma \vdash E : T' \to T \quad \Gamma' \vdash E' : T'}{\Gamma, \Gamma' \vdash E\ E' : T} \quad (\to Elim.)$$

$$\frac{\Gamma \vdash E : T \quad \Gamma' \vdash E' : T'}{\Gamma, \Gamma' \vdash (E, E') : T \times T'} \quad (\times Intro.)$$

$$\frac{\Gamma, X : T, X' : T' \vdash E : T'' \quad \Gamma' \vdash E' : T \times T'}{\Gamma, \Gamma' \vdash \mathbf{let}\ (X, X') = E'\ \mathbf{in}\ E : T''} \quad (\times Elim.)$$

Figure 1. Typing rules for the multiplicative fragment of the simply-typed λ-calculus

The question we will try to answer is if we can take the type system as the mechanism for systematically composing elemental semantic contributions into complex semantic terms. The answer, it turns out, depends on the type system. We can take all semantic constructors as the typing context and try to find all distinct (up to α-equivalence) β/η irreducible terms that have the type t in that context. This guarantees that any reading thus derived will be well-typed, having the sentence type (t)[5]. Moreover, for computationally well-behaved type systems such as G2 and G3, it is guaranteed that given appropriate types for elemental semantic contributions all readings of the sentence will be derived by a correct implementation. Whereas other compositionality mechanisms may wrongly restrict some readings by their designer's oversight, a type system won't: all readings will be well-typed, so it will not reject any of them. Not all type systems, or type assignments will be appropriate, but this approach definitely seems promising.

[5] Since we are dealing with a type system for meaning assembly not a specific semantic representation language here, the well-wormedness of the semantic expressions is not automatically guaranteed, but typically trivial to prove given an appropriate semantic analysis.

Computational issues aside, the only potential problem with using a type system for meaning assembly is that there may be terms that can be proven to have type t given the typing context formed of the elemental semantic contributions of a sentence that do not represent a possible reading of that sentence. For this not to happen the type-system must capture the linguistic constraints of meaning assembly. As we will see soon, the propositional intuitionistic type system of the simply-typed λ-calculus does not, but a type-system based on linear predicate logic(Girard, 1987) does.

The first constraint that intuitionistic logic does not capture is that each semantic contribution is to be used exactly once in the forming of the composite sentence meaning. The Weakening rule found in intuitionistic logic and type systems based on it[6] means that semantic contributions may be (incorrectly) omitted. For example the semantic contributions of sentence (10) can be used to derive a meaning representation that captures the meaning of (11). This is clearly undesirable.

Mary sings happily. (10)

*

$$\dfrac{\dfrac{mary : e \vdash john : e \quad sing : e \to t \vdash sing : e \to t}{mary : e, sing : e \to t \vdash sing(john) : t}\ {\to_E}}{mary : e, sing : e \to t, happily : (e \to t) \to (e \to t) \vdash sing(mary) : t}\ Weakening$$

Mary sings. (11)

The Contraction rule also found in intuitionistic logic allows a semantic contribution by a part of a sentence[7] to be used any number of times in the composite semantic representation for the sentence.

Vijay complains John sings loudly. (12)

*

$$\dfrac{\dfrac{\vdots}{\begin{array}{l} vijay : e, complain : e \times t \to t, \\ \quad john : e, sing : e \to t, \\ \quad loudly : (e \to t) \to (e \to t), \\ \quad loudly : (e \to t) \to (e \to t) \end{array} \vdash \begin{array}{l} loudly\ (\lambda x.complain(x, \\ \quad loudly\ sing\ john))\ vijay \end{array} : t}\ Contraction}{\begin{array}{l} vijay : e, complain : e \times t \to t, \\ \quad john : e, sing : e \to t, \\ \quad loudly : (e \to t) \to (e \to t) \end{array} \vdash \begin{array}{l} loudly\ (\lambda x.complain(x, \\ \quad loudly\ sing\ john))\ vijay \end{array} : t}$$

Vijay complains loudly John sings loudly. (13)

[6] There is a famous result known as the Curry-Howard isomorphism(Howard, 1980) linking logics to type systems. It started with the type system of the simply typed λ-calculus and propositional intuitionistic logic, but a Curry-Howard isomorphism exists for various other logic-type system pairs.

[7] Contraction does not allow the use of arbitrary 'invisible' assumptions in the proof process — this would defeat the purpose of any logic. It only allows an *existing* assumption to be used an arbitrary number of times.

Linear logic (Girard, 1987) rejects the Weakening and Contraction rules. This guarantees that each semantic contribution (assumption) is used exactly once in the composite semantic representation (conclusion). Special modalities (? and !) are introduced to recover the same effect only in formulas where that is desirable. Resource sensitivity in linear logic also comes with a split of the traditional conjunction and disjunction connectives. There are two conjunctions, $\otimes$ (times) and & (with) and two disjunctions, $\oplus$ (plus) and $\parr$ (par). There are also four constants, $\mathbf{1}, \bot, \top, \mathbf{0}$, which are respectively the neutral elements of the aforementioned four conjunctive and dijunctive connectives. Negation ($^{\bot}$) can defined by De Morgan equations. Double linear negation like its classical but unlike its intuitionistic counterpart cancels itself out. Linear implication can also be defined in terms of negation and the par connective in the exact same way implication can be defined in terms of negation and disjunction in both intuitionistic and classical logic. Finally, the quantiers $\forall$ and $\exists$ behave (from a sequent calulus viewpoint) similarly to their intuitionistic and classical counterparts.

Linear logic is very expressive and powerful, but for our purposes we only need the $(\otimes, \multimap, \forall)$ fragment of linear predicate logic. The first-order universal quantifier of G3 corresponds to a certain flavour of type polymorphism that is more appropriate for our purposes than the polymorphism supported by the second-order quantification of System F, the system on which Dalrymple et al.(1997b) based G2. However, given our discussion so far it is not obvious why we need predicate rather than propositional logic to be the basis of our type system.

As a matter of fact a system based on the $(\otimes, \multimap)$ fragment of linear propositional logic suffices to avoid the problems weakening and contraction may cause and is the natural next step in our quest to find a type system that captures compositionality constraints now that we have rejected the $(\times, \rightarrow)$ fragment of linear intuitionistic logic. All one has to do to move from the intuitionistic system to its linear counterpart is replace $\rightarrow$ with $\multimap$ and $\times$ with $\otimes$ and remember not to use weakening and contraction in type derivations. The unavailability of weakening and contraction as type derivation rules is manifest in a very simple way in meaning representations: every typed constant (or formula label, depending on which side of the Curry-Howard we are looking) that appears in the context Γ formed of the initial semantic contributions will appear exactly as many times on the right-hand side of the turnstile in a non-binding position (not immediately following a λ) as it appears on the left hand side and every λ-bound variable appears exactly once in its binder's scope.

$$\begin{array}{ll} (a) & tom : e,\ like : e \otimes e \multimap t \ \nvdash\ like(tom, jerry) : t \\ (b) & tom : e,\ like : e \otimes e \multimap t \ \nvdash\ like(tom, jerry) : t \\ (c) & tom : e,\ like : e \otimes e \multimap t \ \nvdash\ like : e \otimes e \multimap t \\ (d) & like : e \otimes e \multimap t \ \nvdash\ \lambda x.like(x, x) : e \multimap t \end{array} \tag{14}$$

The system based on $(\otimes, \multimap)$ propositional logic system avoids all the pitfalls of (14) whereas its intuitionistic counterpart would only prohibit the intuitionistic version of (a). The type system Montague used was very similar to the one we have already rejected; it was based on implicative intuitionistic propositional logic $(\rightarrow)$ and included the Weakening, Contraction and Exchange structural rules. However, he did not use the type system to drive the derivation of semantic forms, but a methodology based on the constituent structure of a sentence that was restrictive enough to avoid the aforementioned pitfalls, as well as the one we will discuss next.

Both TLCG and Glue offer an elegant alternative, type-driven semantic form derivation, that makes cumbersome manipulation of syntax unnecessary for obtaining different readings. TLCG is more faithful to Montague's program as it maintains a close relation to constituent syntactic structure, whereas Glue can work equally well using functional structure information. Unlike Glue, TLCG also rejects the Exchange (or Permutation) structural rule, the one structural rule of classical and intuitionistic logic linear logic does not reject. The rejection of Exchange comes with a split of implication into a forward and backward variant. That the Exchange rule is allowed in linear logic has the consequence that our current type system allows the derivation of a semantic form for (16) from the semantic contributions of (15). After all, viewed as multisets (which is what the Exchange rule allows) the typing contexts of the two sentences are exactly the same. The problem is that the linear type system with e and t as its only base types does not distinguish between subject and object entities, between verb phrases and nouns and so on. The moment we correctly incorporate functional role information into a linear type system, is the moment we get a Glue system that can be used for a program of compositionality based on functional structure.

Jerry likes Tom (15)

$$f : \left[\begin{array}{ll} \textsc{pred} & \text{`}like\text{'} \\ \textsc{subj} & s : \left[\ \textsc{pred}\ \text{`}\textsc{Jerry}\text{'}\ \right] \\ \textsc{obj} & o : \left[\ \textsc{pred}\ \text{`}\textsc{Tom}\text{'}\ \right] \end{array}\right]$$

Reading 1
$jerry : e :, like : e \otimes e \multimap t, tom : e \vdash like(jerry, tom) : t$
**Reading 2*
$jerry : e :, like : e \otimes e \multimap t, tom : e \vdash like(tom, jerry) : t$

Tom likes Jerry (16)

Our simple linear type system captures the requirement that each semantic contribution is used exactly once in meaning representations corresponding to readings of a sentence. What we need to do now is incorporate functional information into the base types. For that purpose, we use f-structure labels such as o, s and f as seen in the f-structure for (15). Note that an f-structure label captures the function of the semantic contribution in the sentence in absolute terms, rather than relative to a constituent of the sentence such as a subordinate clause. For example, in complex sentences specifying that a semantic contribution has the SUBJect function is not enough, as different clauses may have different subjects, but the f-structure label uniquely identifies the subject in question. We will briefly consider two alternative ways of using f-structure labels in order to capture functional information in the base types:

1. We can simply make a move from propositional to predicate logic in our type system. The types e and t are no longer base types (propositions), but base type constructors of arity one (1-predicates) taking a label in the f-structure as their argument. A base type constructor with all its arguments is a base type. Thus $jerry : e(s)$ is the subject entity, $tom : e(o)$ is the object entity of sentence f, and $like : e(s) \otimes e(o) \multimap t(f)$, is a function from a subject-object pair of sentence f to its truth value.

2. We can keep the propositional type system and use f-structure labels directly as base types, thus encoding functional roles but at the cost of losing the distinction between e and t values. The second proposal is simple but unfortunately inadequate.

Propositional logic does not allow quantification, but predicate logic does. As we shall soon see quantification over labels in the Glue type system is required in the types of certain kinds of semantic contribution: quantifier noun phrases, coincidentally, and possibly others. In the first proposal support for quantification comes with the system, whereas in the second it is not supported due to its propositional nature. That fact alone is a perfectly valid reason to reject the second proposal in favour of first-order Glue. Jumping ahead again, first-order Glue with labels as first-order parameters of types has the additional advantage that its notion of quantification is exactly what is needed. System F on the other hand to which the designers of G2 resorted in order to allow quantification over types (since labels are types in G2) has a notion of higher-order quantification which was unsuitable and had to be restricted. But even that restriction alone was not enough. Having no distinction between entity and truth values causes other problems. G2 had to be extended so as to have an e and a t sort to solve these problems. G2, the end result of all these ad-hoc extensions and restrictions, is at the end a much more complex formal system than the second alternative above. The first proposal, G3 with an $(e/1, t/1)$ type system, is the right choice that just had not been considered until now. These matters are discussed in more detail in Section 4.

$$\frac{N : T,\ \Gamma,\ N' : T',\ \Gamma' \vdash E : T''}{N' : T',\ \Gamma,\ N : T,\ \Gamma' \vdash E : T''}\ (Exchange)$$

$$N : T \vdash N : T\ ^{(Axiom)}$$

$$\frac{\Gamma, X : T \vdash E : T'}{\Gamma \vdash \lambda X.E : T\ \multimap T'}\ (\multimap Intro.)$$

$$\frac{\Gamma \vdash E : T'\ \multimap T \quad \Gamma' \vdash E' : T'}{\Gamma, \Gamma' \vdash E\ E' : T}\ (\multimap Elim.)$$

$$\frac{\Gamma \vdash E : T \quad \Gamma' \vdash E' : T'}{\Gamma, \Gamma' \vdash (E, E') : T \otimes T'}\ (\otimes Intro.)$$

$$\frac{\Gamma, X : T, X' : T' \vdash E : T'' \quad \Gamma' \vdash E' : T \otimes T'}{\Gamma, \Gamma' \vdash \mathbf{let}\ (X, X) = E'\ \mathbf{in}\ E : T''}\ (\otimes Elim.)$$

$$\frac{\Gamma \vdash E : T}{\Gamma \vdash E : \forall V.T}\ [where\ V \notin FV(Types(\Gamma))]\quad (\forall Intro.)$$

$$\frac{\Gamma \vdash E : \forall V.T}{\Gamma \vdash E : T[V := L]}\ (\forall Elim.)$$

Figure 2. First-Order Glue Inference Rules

Notational conventions: The standard syntactic conventions of linear logic formulas apply: the longest subformula parse is preferred, $\otimes$ binds tighter than $\multimap$, $\otimes$ is left-associative and $\multimap$ is right-associative. Also:

1. Instead of using parentheses around the list of arguments of base type constructors(predicates) in our system, we choose to write it as a subscript. This convention helps reduce the number of parentheses and makes for easier reading of complicated types.
2. We use lowercase Greek letters for variables and lowercase Roman letters for constants in types.
3. We prefer to write types (formulas if we take a logical rather than type theoretic perspective) in *Prenex Normal Form* (i.e. 'quantifiers first' form). Since every first-order logic formula has a PNF equivalent we can always do that.

So we write $\forall\alpha.(e_s \multimap t_s) \multimap (e_s \multimap t_\alpha) \multimap t_\alpha$ instead of the rather longer $\forall\alpha.(e(s) \multimap t(s)) \multimap (e(s) \multimap t(\alpha)) \multimap t(\alpha)$.

With functional information incorporated into the base types of our first-order Glue system only the correct reading for (15) is derivable. Below is the entire typing derivation in which we can clearly see that the Exchange rule has a role to play in the derivation of the correct reading (2nd line from the bottom) and also that thanks to our label sensitive-types it is now impossible for Exchange to be used to derive the erroneous reading. The convention we will use from now on that the antecedent in an linear typing sequent is a multiset will replace the Exchange rule.

$$\dfrac{\dfrac{\dfrac{l : e_s \otimes e_o \multimap t_f \vdash l : e_s \otimes e_o \multimap t_f}{l : e_s \otimes e_o \multimap t_f \vdash l : e_s \otimes e_o \multimap t_f} \quad \dfrac{j : e_s \vdash j : e_s \quad h : e_o \vdash h : e_o}{j : e_s, h : e_o \vdash (j,h) : e_s \otimes e_o}}{l : e_s \otimes e_o \multimap t_f,\ j : e_o, h : e_s, \vdash l(j,h) : t_f}}{j : e_s,\ l : e_s \otimes e_o \multimap t_f,\ h : e_o \vdash l(j,h) : t_f}$$

The proposed first-order $(e/1, t/1)$ G3 system solves the problems of an intuitionistic propositional (e, t) type system that allows the derivation of additional composite meaning representations corresponding to invalid readings. It is also powerful enough to meet the challenge of providing all possible readings for natural language sentences involving universal and existential quantification.

Continuing the expansion of the small fragment of English covered here, we note that some noun phrases are analysed as quantifiers, rather than as simple entities. A simple example is given in (17).

Everyone smiles. (17)

$$f : \left[\begin{array}{l} \text{PRED} \ \text{`SMILE'} \\ \\ \text{SUBJ} \quad s : \left[ \ \text{PRED} \ \text{`}everyone\text{'} \ \right] \end{array} \right]$$

Typing context:

$$\Gamma = \begin{array}{l} everyone : \forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha, \\ smile : e_s \multimap t_f \end{array}$$

Reading: $(\alpha = f)$

$$\Gamma \vdash everyone\ smile : t_f$$

Readers unfamiliar with the Glue treatment of quantification in natural language may wonder why universal quantification is used. They may expect that the type of *everyone* would be $(e_s \multimap t_f) \multimap t_f$. There is a very practical reason why this approach is not taken: it works well for simple sentences like (17) and (18), but not for more complicated examples like (20). But this is not simply a matter of trial and error. The ingenious idea of Dalrymple et al.(1995a) to underspecify the semantic constructors using universal quantification over labels actually has a very good type theoretic and linguistic reason which we will discuss next in addition to the cold, practical reason just given.

The fundamental idea behind our analysis is that the potential for combination of quantifier noun phrases should not be any more or less restricted than that of simple entity noun phrases with regards to being combined with some predicate. Assuming we had a simple entity, say *fernando*, as the subject of (17), its type would have been e_s. Such a value could have combined with any predicate $e_s \multimap t_\alpha$ as its argument irrespective of what this α label is, giving a t_α result. The following type lifting judgement is a concise way of expressing the same fact:

$$\mathit{fernando} : e_s \vdash \lambda P.P\ \mathit{fernando} : \forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha$$

We would expect a quantifier noun phrase to behave like a type lifted simple entity noun phrase. Therefore it should come as no surprise that the type of *everyone* in (17) is exactly $\forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha$. The quantifier noun phrase is not restricted to combine only with $e_s \multimap t_f$ predicates, as an analysis of quantification giving it instead the type $(e_s \multimap t_f) \multimap t_f$ would have it. Instead it can combine with any predicate $e_s \multimap t_\alpha$ for any label α. It takes such a predicate as its argument, and returns a t_α truth value, thus exhibiting analogous behaviour to a simple entity.

We may also note that the presented analysis of quantification avoids LFG-specific solutions that might have also have been able to give the expected range of readings even in complicated examples. Dalrymple et al.(1995a) not only gave us a very powerful and elegant analysis of quantification but also one that does not tie Glue to LFG thus paving the way for Glue to be used with a variety of syntax formalisms.

Everyone likes someone (18)

$$f : \begin{bmatrix} \text{PRED} & \text{`}\mathit{like}\text{'} \\ \text{SUBJ} & s : \begin{bmatrix} \text{PRED} & \text{`}\mathit{everyone}\text{'} \end{bmatrix} \\ \text{OBJ} & o : \begin{bmatrix} \text{PRED} & \text{`}\mathit{someone}\text{'} \end{bmatrix} \end{bmatrix}$$

Typing context:

$$\Gamma = \begin{array}{l} \mathit{everyone} : \forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha, \\ \mathit{like} : e_s \otimes e_o \multimap t_f, \\ \mathit{someone} : \forall\beta.(e_o \multimap t_\beta) \multimap t_\beta \end{array}$$

First reading: $(\alpha = \beta = f)$

$$\Gamma \vdash \mathit{everyone}\ \lambda x.\mathit{someone}\ \lambda y.\mathit{like}(x, y) : t_f$$

Second reading: $(\alpha = \beta = f)$

$$\Gamma \vdash \mathit{someone}\ \lambda y.\mathit{everyone}\ \lambda x.\mathit{like}(x, y) : t_f$$

Sentence (18) is a typical example of a sentence in which we observe the phenomenon of quantifier scope alteration which presented a challenge for compositional semantics, a challenge Montague(1973) first took on, with considerable success. Our approach effortlessly and naturally gives us the desired readings both in simple sentences as well as more complex ones.

After we started using types with first-order universal quantification in addition to giving the term for each reading we started giving the instantiations of all label variables. This is meant to help appreciate the role of quantification in the type system. Admittedly our analysis produces exactly the same readings for (18) (and similarly for (17)) as the analysis without label variables that we rejected. This is fairly obvious as both α and β were instantiated to f anyway.

Indeed if that was the case for all possible sentences the theoretical reasoning about the necessity of first-order universal quantification in the type system could be overruled on a practical basis. That would not make our analysis wrong, but it mean that our first-order Glue system is unnecessarily powerful. But that is not the case. We will shortly come to an example (sentence (20)) where our analysis does make a difference.

We have reached an interesting point where we have covered only a small fragment of English but have used all the Glue type constructors we are going to need. We will now concentrate on using the formal framework to provide analyses for a slightly larger fragment of English.

Every child plays (19)

$$f:\left[\begin{array}{ll}\text{PRED} & \text{`PLAY'}\\ \\ \text{SUBJ} & s:\left[\begin{array}{ll}\text{SPEC} & \text{`}every\text{'}\\ \text{PRED} & \text{`CHILD'}\end{array}\right]\end{array}\right]$$

Typing context:

$$\Gamma = \begin{array}{l} every : \forall\alpha.(e_s \multimap t_s) \multimap (e_s \multimap t_\alpha) \multimap t_\alpha,\\ child : e_s \multimap t_s,\\ play : e_s \multimap t_f\end{array}$$

Reading: $(\alpha = f)$

$\Gamma \vdash every\ child\ play : t_f$

We start by extending the range of noun phrases we can handle. Up to now we have only encountered nouns phrases that are simple entities ('Mary'), and quantifiers ('everyone'). In (19) we finally encounter a composite noun phrase consisting of a generalized quantifier (the determiner 'every') and its restrictor predicate (the common noun 'child'). In Section 4 we will see various alternative analyses, but for now we will use one in which the common noun and the verb take the same type of argument (e_s) and at the same time distinguishes the role of the verb from that of the noun by having the former return a value of type t_f and the latter a value of type t_s. A generalised quantifier and a restrictor are expected to combine and behave exactly like a generalised quantifier. This is exactly why the argument type of the generalised quantifier matches the type of the restrictor ($e_s \multimap t_s$) and its return type is the type of a quantifier ($\forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha$). The type of 'every', $\forall\alpha.(e_s \multimap t_s) \multimap ((e_s \multimap t_\alpha) \multimap t_\alpha)$, is simply $(e_s \multimap t_s) \multimap (\forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha)$ converted to Prenex Normal Form.

Every representative of a company took a sample. (20)

$$f:\left[\begin{array}{ll}\text{PRED} & \text{`TOOK'}\\ \text{SUBJ} & s:\left[\begin{array}{ll}\text{SPEC} & \text{`}every\text{'}\\ \text{PRED} & \text{`REPRESENTATIVE'}\\ \text{OBL}_{\text{OF}} & r:\left[\begin{array}{ll}\text{SPEC} & \text{`A'}\\ \text{PRED} & \text{`FIRM'}\end{array}\right]\end{array}\right]\\ \text{OBJ} & o:\left[\begin{array}{ll}\text{SPEC} & \text{`A'}\\ \text{PRED} & \text{`SAMPLE'}\end{array}\right]\end{array}\right]$$

Typing Context:

$$\Gamma=\begin{array}{l}every:\forall\alpha.(e_s\multimap t_s)\multimap(e_s\multimap t_\alpha)\multimap t_\alpha,\\ rep:e_s\multimap t_s,\\ of:e_r\multimap(e_s\multimap t_s)\multimap(e_s\multimap t_s),\\ a_1:\forall\beta.(e_r\multimap t_r)\multimap(e_r\multimap t_\beta)\multimap t_\beta,\\ firm:e_r\multimap t_r,\\ take:e_s\otimes e_o\multimap t_f,\\ a_2:\forall\gamma.(e_r\multimap t_r)\multimap(e_r\multimap t_\gamma)\multimap t_\gamma,\\ sample:e_o\multimap t_o\end{array}$$

First reading: $(\alpha=f,\ \beta=s,\ \gamma=f)$

$\Gamma\vdash every\,(\lambda x.a_1\,firm\,\lambda y.of y\,rep\ x)\,\lambda x.a_2\,sample\,\lambda z.take(x,z)$

Second reading: $(\alpha=f,\ \beta=s,\ \gamma=f)$

$\Gamma\vdash a_2\,sample\,\lambda z.every\,(\lambda x.a_1\,firm\,\lambda y.of y\,rep\ x)\,\lambda x.take(x,z)$

Third reading: $(\alpha=f,\ \beta=f,\ \gamma=f)$

$\Gamma\vdash a_2\,sample\,\lambda z.a_1 firm\,\lambda y.every\,(of y\,rep)\,\lambda x.take(x,z)$

Fourth reading: $(\alpha=f,\ \beta=f,\ \gamma=f)$

$\Gamma\vdash a_1\,firm\,\lambda y.every\,(of y\,rep)\,\lambda x.a_2\,sample\,\lambda z.take(x,z)$

Fifth reading: $(\alpha=f,\ \beta=f,\ \gamma=f)$

$\Gamma\vdash a_1\,firm\,\lambda y.a_2\,sample\,\lambda z.every\,(of y\,rep)\,\lambda x.take(x,z)$

The only type of piece of the puzzle that is the typing context of (20) we have not seen before is that of the semantic contribution of 'of'. We treat *of* x where x is an entity as a modifier of common nouns, so of is a function from entities to common noun modifiers. The Glue analysis of quantification correctly predicts the five possible ways of putting the puzzle pieces together i.e. the five readings of (20). This would not have been possible if β was not free to be instantiated to either s or f. The first instantiation gives the first two readings and the latter the remaining three. A good example is just as important or even more important than a good theory.

Tempting as it may be we will not examine analyses of modification, anaphora and other phenomena. During the last ten years the Glue literature has been gaining in both volume and insight. The emphasis in this paper is to propose a more elegant and powerful formal framework for Glue. Analyses given in G2 (or G1) can be transferred easily to G3. Dalrymple(1999) collected a number of interesting Glue papers[8], but there is also much recent exiting work.

[8] The original five-readings example of which (20) is a lexical variation can also be found there.

3. Relating Ad-hoc Glue to First-Order Glue

We have transferred a flagship Glue analysis, that of quantification, to G3 (with only a small modification to avoid making reference to G1/G2-style semantic projection structures). We will now proceed to establish that any G2 analysis can easily be faithfully transferred to G3. In fact G2 can be seen as a somewhat impoverished version of G3. This is puzzling if one thinks that G2 is based on a higher-order system and on top of that has been extended to a two-sorted system whereas G3 is a humble first-order system. We will prove that for all its System F heritage the current Glue system, is actually closer to the proposed first-order system than to System F[9] despite notational appearances. The key to the puzzle is the observation that whereas quantification over propositional formulas in System F is beyond the encoding capacity of a first-order system, quantification over base types is not.

DEFINITION 1. *F1 is the fragment of System F with the restriction on type inference rules that a type variable can only be replaced by a base type (variable or constant).*

DEFINITION 2. *G3/t is the fragment of G3 that only includes typed terms in which no base type constructor other than $t/1$ is used and the only individuals (labels) in the types are atomic (variables or constants, but not the result of some function).*

Notation: Λ ranges over λ-calculus terms, P over proposition symbols in F1 and constant arguments in G3 types, X over variables in F1 and G3 types, T and T' over F1 type formulas and D ranges over F1 type derivation trees.

DEFINITION 3. *The function ω_{F1} that maps F1 typed terms to typed terms in G3/t is defined as follows:*

$$\omega_{F1}(\Lambda : T) = \Lambda : \tau_{F1}(T)$$

where

$$\begin{array}{lcl} \tau_{F1}(P) & = & t(P) \\ \tau_{F1}(X) & = & t(X) \\ \tau_{F1}(T \multimap T') & = & \tau_{F1}(T) \multimap \tau_{F1}(T') \\ \tau_{F1}(T \otimes T') & = & \tau_{F1}(T) \otimes \tau_{F1}(T') \\ \tau_{F1}(\forall X.T) & = & \forall X.\tau_{F1}(T) \end{array}$$

THEOREM 1. *ω_{F1} is a bijection.*

Proof Given any two different F1 typed terms their ω_{F1} images will be different and every G3/t typed term is an ω_{F1} image of a F1 typed term. This is obvious from the definition of ω_{F1}. □

[9] We are assuming a linear version of System F that supports $\otimes$ and has a simplified syntax for the $\forall$ rules.

$$\frac{N : T,\ \Gamma,\ N' : T',\ \Gamma' \vdash E : T''}{N' : T',\ \Gamma,\ N : T,\ \Gamma' \vdash E : T''}\ (Exchange)$$

$$N : T \vdash N : T\ (Axiom)$$

$$\frac{\Gamma, X : T \vdash E : T'}{\Gamma \vdash \lambda X.E : T\ \multimap T'}\ (\multimap Intro.)$$

$$\frac{\Gamma \vdash E : T'\ \multimap T \quad \Gamma' \vdash E' : T'}{\Gamma, \Gamma' \vdash E\ E' : T}\ (\multimap Elim.)$$

$$\frac{\Gamma \vdash E : T \quad \Gamma' \vdash E' : T'}{\Gamma, \Gamma' \vdash (E, E') : T \otimes T'}\ (\otimes Intro.)$$

$$\frac{\Gamma, X : T, X' : T' \vdash E : T'' \quad \Gamma' \vdash E' : T \otimes T'}{\Gamma, \Gamma' \vdash \mathbf{let}\ (X, X) = E'\ \mathbf{in}\ E : T''}\ (\otimes Elim.)$$

$$\frac{\Gamma \vdash E : T}{\Gamma \vdash E : \forall V.T}\ [where\ V \notin FV(Types(\Gamma))]\ (\forall Intro.)$$

$$\frac{\Gamma \vdash E : \forall V.T}{\Gamma \vdash E : T[V := L]}\ (\forall Elim.)$$

Figure 3. (Linear) System F Inference Rules

DEFINITION 4. *Two type systems, T1 and T2, are directly interchangeable if and only if there is a bijective function $\mathcal{G}$ from T1 to T2 type derivation trees such that for any T1 type judgement J and (possibly empty) list of type judgements $J_1, \ldots, J_n$ there is an inference rule in T1 by which J can be derived in T1 given $J_1, \ldots, J_n$ if and only if there is an inference rule in T2 by which $\mathcal{G}(J)$ can be derived in T2 given $\mathcal{G}(J_1), \ldots, \mathcal{G}(J_n)$, and has the same, if any, side-conditions.*

THEOREM 2. *F1 and G3/t are directly interchangeable.*

Proof Trivial. The bijection used is

$$\mathcal{G}_1(A_1 : T_1, \ldots, A_n : T_n \vdash \Lambda : T) = \omega_{F1}(A_1 : T_1), \ldots, \omega_{F1}(A_n : T_n) \vdash \omega_{F1}(\Lambda : T).$$

□

That F1 and G3/t are directly interchangeable is a very strong result. It basically means that they are indeed different only in notation. The following derivation given in both systems is a simple demonstration of that.

$$\dfrac{\dfrac{q{:}\forall\alpha.(s\multimap\alpha)\multimap\alpha\vdash q{:}\forall\alpha.(s\multimap\alpha)\multimap\alpha}{q{:}\forall\alpha.(s\multimap\alpha)\multimap\alpha\vdash q{:}\,(s\multimap f)\multimap f} \quad p{:}s\multimap f\vdash p{:}s\multimap f}{q{:}\forall\alpha.(s\multimap\alpha)\multimap\alpha,\, p{:}s\multimap f\vdash q(p){:}f} \tag{21}$$

$$\dfrac{\dfrac{q{:}\forall\alpha.(t(s)\multimap t(\alpha))\multimap t(\alpha)\vdash q:\forall\alpha.(t(s)\multimap t(\alpha))\multimap t(\alpha)}{q{:}\forall\alpha.(t(s)\multimap t(\alpha))\multimap t(\alpha)\vdash q{:}\,(t(s)\multimap t(f))\multimap t(f)} \quad p{:}t(s)\multimap t(f)\vdash p{:}t(s)\multimap t(f)}{q{:}\forall\alpha.(t(s)\multimap t(\alpha))\multimap t(\alpha),\, p{:}t(s)\multimap t(f)\vdash q(p){:}t(f)} \tag{22}$$

We can now use our experience with F1 to relate G2 to G3. G2 is based on System F extended to two sorts, but the restrictions placed upon quantification, which only allow it to work with base types of the t sort mean that it is still very much a first-order system in disguise.

DEFINITION 5. *G2 is the fragment of System F extended to two sorts t and e with the restriction on type inference rules that a type variable can only be replaced by a base type of the t sort.*

DEFINITION 6. *G3/t[e] is the fragment of G3 that only includes typed terms in which no base type constructors other than $t/1$ and $e/1$ is used, the only individuals (labels) in the types are atomic (variables or constants, but not the result of some function), no label is used as an argument to both $t/1$ and $e/1$ type constructors, and variables can not be arguments to $e/1$ type constructors.*

Notation: Λ ranges over λ-calculus terms, P over proposition symbols in G2 and constant arguments in G3 types, X over variables in G2 and G3 types, T and T' over G2 type formulas and D ranges over System F1 type derivation trees.

DEFINITION 7. *The function ω that maps G2 typed terms to typed terms in G3/t[e] is defined as follows:*

$$\omega(\Lambda : T) = \Lambda : \tau(T)$$

where

$$\begin{array}{lcl}
\tau(P_t) & = & t(P) \\
\tau(P_e) & = & e(P) \\
\tau(X) & = & e(X) \\
\tau(T \multimap T') & = & \tau(T) \multimap \tau(T') \\
\tau(T \otimes T') & = & \tau(T) \otimes \tau(T') \\
\tau(\forall X_t.T) & = & \forall X.\tau(T)
\end{array}$$

THEOREM 3. *ω is a bijection.*

Proof Given any two different G2 typed terms their ω images will be different and every G3/$t[e]$ typed term is an ω image of a G2 typed term. □

THEOREM 4. *G2 and G3/$t[e]$ are directly interchangeable.*

Proof Trivial. The bijection used is

$$\mathcal{G}(A_1{:}T_1,\ldots,A_n{:}T_n \vdash \Lambda{:}T) = \omega(A_1{:}T_1),\ldots,\omega(A_n{:}T_n) \vdash \omega(\Lambda{:}T).$$

□

So G2 and G3/$t[e]$ are indeed different only in notation. For an example we turn to what could have been a derivation of the composite meaning of a sentence with a quantifier and an intransitive verb.

$$\dfrac{\dfrac{q{:}\forall\alpha_t.(s_e \multimap \alpha) \multimap \alpha \vdash q{:}\forall\alpha.(s_e \multimap \alpha) \multimap \alpha}{q{:}\forall\alpha_t.(s_e \multimap \alpha) \multimap \alpha \vdash q{:}\,(s_e \multimap f_t) \multimap f_t} \quad p{:}s_e \multimap f_t \vdash p{:}s_e \multimap f_t}{q{:}\forall\alpha_t.(s_e \multimap \alpha) \multimap \alpha,\ p{:}s_e \multimap f_t \vdash q(p){:}f_t} \tag{23}$$

$$\dfrac{\dfrac{q{:}\forall\alpha.(t(s) \multimap t(\alpha)) \multimap t(\alpha) \vdash q{:}\forall\alpha.(t(s) \multimap t(\alpha)) \multimap t(\alpha)}{q{:}\forall\alpha.(t(s) \multimap t(\alpha)) \multimap t(\alpha) \vdash q{:}(t(s) \multimap t(f)) \multimap t(f)} \quad p{:}t(s) \multimap t(f) \vdash p{:}t(s) \multimap t(f)}{q{:}\forall\alpha.(t(s) \multimap t(\alpha)) \multimap t(\alpha),\ p{:}t(s) \multimap t(f) \vdash q(p){:}t(f)} \tag{24}$$

Given the strength of our result regarding G2 and G3/$t[e]$ it is fairly obvious that to convert a G2 implementation to a G3/$t[e]$ implementation, all one needs to do is change the syntax of types. It is clear that G3/$t[e]$, therefore also G2, is a restricted fragment of G3. The definition of G3/$t[e]$ was carefully chosen to reflect the properties of G2. It tells us exactly what G2 amounts to in the realm of first-order Glue.

Seeing G2 as a restricted version of G3 we might wonder if any of the restrictions are necessary. The easy answer is 'none'. G3 in its entirety does not hide any traps for our program of computational linguistics. It is up to its users to come up with correct analyses and it gives them more tools to do this easier and more elegantly (user-chosen base-type constructors, not necessarily of arity 1, functions on labels, and a better use of labels in forming base types[10]). Section 4 deals with the design of G2 from a different perspective and answers many similar questions.

[10] The same label can be used for example as a parameter to both an $e/1$ and a $t/1$ base-type constructor. Interestingly, that would not have been of any use if the Glue-semantics interface had not changed when G2 replaced G1. In G1 semantic projection each label was associated with a semantic expressions, which would have meant it could either be an entity or a truth value, not both. In G2 (and G3) labels are just used for forming the Glue types of semantic contributions.

4. On the design of G2 and G3

Certainly the big question is if a system such as G2 and G3 can encode analyses for the vast range of phenomena found in natural languages. However, in a paper that proposes a new formalism on the grounds that its design is better than its predecessor's it is worth exploring the decisions that shaped G2 and how G3 improves on some of them.

Probably the most 'advertised' feature of Glue is its resource sensitivity. All Glue systems are based on linear logic which is a resource sensitive logic. We may wonder if resource sensitivity is essential. In Section 2 two pairs of sentences, (11)&(10) and (12)&(13), were used to show the kind of problems Weakening and Contraction may cause respectively. These examples all involved modification and the analyses discussed used a simple (e, t) propositional type system. In fact many things can go wrong with such a type system. For instance, in the (e, t) analysis of Section 2 verbs $(e \multimap t)$ and common nouns $(e \multimap t)$ are interchangeable and so are subject and object noun phrases. It is reasonable to ask if a better type system with an appropriate analysis can solve the same problems without being resource sensitive.

Indeed, label sensitivity coupled with a different analysis of modification can render the Weakening and Contraction rules useless even in the absence of resource sensitivity. Here is an example of a typing context for (10) using a non-standard analysis of modification that disallows throwing away or duplicating modifiers[11]:

$$mary : e_s,\ sing : e_s \to t_g,\ hapilly : (e_s \to t_g) \to (e_s \to t_f).$$

In this analysis the use of the intermediate label g guarantees that one has to apply the modifier to the verb in order to then combine the result with the subject entity. Multiple modifiers will only combine in the prescribed order. In Glue both this analysis and the standard, unordered, analysis are available. In the absence of resource sensitivity only this strict ordering can guarantee modifiers are not omitted or duplicated in the meaning assembly process. Moreover the ordered analysis avoids the problems its unordered counterpart has in the treatment of multiple modification(*'ancient fake golden crown'*) and its interaction with modifier modification(*'apparently wrong third answer'*).

The ordered modification analysis may demonstrate that resource sensitivity is somewhat overrated but that does not constitute an argument for rejecting it in a system for meaning composition. But we can argue that G3, unlike the intuitionistic type system considered, allows both ordered and unordered analyses exactly because it is resource sensitive. Furthermore, eliminating two structural inference rules, as Glue (linear logic) does, reduces the complexity of the system, whereas having them and trying hard not to allow them to be applied has no true advantage. The final argument in favour of resource sensitivity in Glue is that it is an essential principle of compositional semantics; that Glue formally enforces it is one of its most attractive features.

[11] A first-order intuitionistic type system (allowing Weakening and Contraction) with our parameters-as-subscripts convention is assumed.

Label sensitivity is not a concept that applies to a type system or a logic per se, but to formal systems for meaning assembly of semantic contributions associated to structures with identifying labels. The label of syntactic (and other structures such as 'semantic' projections) are used to form the base types in G2 and G3 (the two Glue systems that are type systems). However, the two systems do this rather differently.

The decision to use labels as base types in G3 is the source of most of its unnecessary complexity. Certainly, the idea of using labels as propositional types seems simple enough at first. After all, propositional logic is simpler than predicate logic. The problem is that in order to support the Glue analysis of quantification(Dalrymple et al., 1995a) in natural language, quantification over labels must also be supported. Otherwise it is impossible to deal with examples like (20) as discussed in Section 2. In G3 labels are parameters of predicates and first-order quantification is readily available, but in G2 labels are (base) types, so quantification over types is needed instead. This leads to a leap from a simple system based on propositional logic to a second-order system such as System F.

G2 is based on System F but is not System F. The most important difference is that higher order quantification, the hallmark of System F (second-order λ-calculus) is prohibited in G2. To see why this is so, we may consider the following System F typing context

$$\begin{aligned} \mathit{everyone}: &\ \forall A.(s \multimap A) \multimap A, \\ \mathit{love}: &\ s \otimes o \multimap f, \\ \mathit{someone}: &\ \forall B.(o \multimap B) \multimap B. \end{aligned}$$

Since

$$\mathit{love} : s \otimes o \multimap f \vdash \lambda x.\lambda y.\mathit{love}(x, y) : s \multimap o \multimap f,$$

in System F higher-order quantification over A allows the following:

$$\begin{aligned} \mathit{everyone}: &\ \forall A.(s \multimap A) \multimap A, \\ \mathit{love}: &\ s \otimes o \multimap f \end{aligned} \vdash \mathit{everyone}\ \lambda x.\lambda y.\mathit{love}(x, y) : o \multimap f$$

What we wanted was to allow A and B to range over truth base types. What we got instead was A instantiated to a type of a function from entities to truth values. Instantiating B to f we get:

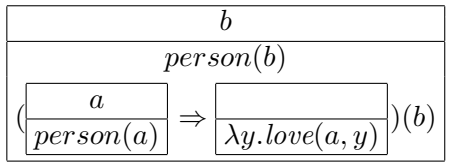
$$\begin{aligned} \mathit{everyone}: &\ \forall A.(s \multimap A) \multimap A, \\ \mathit{love}: &\ s \otimes o \multimap f, \\ \mathit{someone}: &\ \forall B.(o \multimap B) \multimap B \end{aligned} \vdash \mathit{someone}\,(\mathit{everyone}\,\lambda x.\lambda y.\mathit{love}(x, y)) : f$$

The problem is not that System F can not derive the two readings of sentence (18), but that it can derive additional readings. However, we are not talking about situations similar to those considered in Section 2 where we were getting extra readings corresponding to different sentences than the ones analysed, but about ill-formed (for our purposes) readings that System F considers perfectly fine. To better illustrate the problem we present the readings in DRT as obtained by the analysis of van Genabith and Crouch(1997) or Kokkonidis(2003). Even though the 'reading' just obtained in System F looks somewhat similar to the reading of (18) where '*someone*' takes wide scope, also obtainable in System F, only the latter corresponds to a well-formed DRS.

System F

$someone\ (everyone\ \lambda x.\lambda y.love(x,y))$

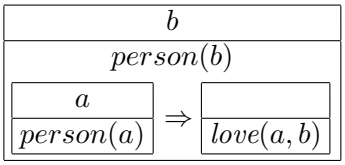
DRT



System F & Glue

$someone\ \lambda y.everyone\ \lambda x.love(x,y)$

DRT



The last thing we want is a formal system for compositionality that gives us as sentence readings λ-calculus terms that correspond to ill-formed expressions in the meaning language. The problem with System F is that its notion of quantification over types does not distinguish between a simple base type and a complex type. We have shown that System F and G2 may look similar but in fact behave rather differently. Since what we wanted was to quantify over (truth) labels which G2 has taken to be base types, we may wonder if G2 amounts to System F with quantification restricted to base types. As it turns out the designers of G2 (and G1) have also considered the possibility of not having two sorts(Dalrymple, 2003, personal communication). The problem they found with doing that is rather amusing: if there is no distinction between e types and t types, the type of a quantifier noun phrase can "wrap around itself". For instance, take the typing context

$$everyone : \forall A.(s \multimap A) \multimap A,\ sleep : s \multimap f.$$

Let us instantiate A to s. Nothing is stopping us from doing that in a System F with quantification restricted to base types since s is indeed a base type. Without using up any resources (semantic contributions in the typing context) we may obtain the identity function for entities of type s:

$$\vdash \lambda x.x : s \multimap s$$

Combining the quantifier and the identity function we get a subject entity:

$$everyone : \forall A.(s \multimap A) \multimap A \vdash everyone\ \lambda x.x : s$$

And to complete the trick, we use that as the argument of *sleep* to get our 'reading':

$$everyone : \forall A.(s \multimap A) \multimap A,\ sleep : s \multimap f \vdash sleep\ (everyone\ \lambda x.x) : f$$

An attempt to use a System F with quantification restricted to base types as a formal system for meaning assembly fails just as badly[12] as trying to use System F with no restrictions. The reason G2 works is because its designers extended System F to two sorts (t and e) thus enabling the system to distinguish between truth and entity types in addition to restricting quantification to base types — actually only of the t sort because quantification over base types of the e sort was not needed in practice.

This discussion hopefully sheds some light into why G2 is as complicated as it is. The problems started because labels as base types (or propositions in G2's predecessors) seemed like a good idea, but as it turns out it is not. G3 has a first-order type system where labels are mere arguments to base types. The claim made in this paper is that this is exactly how Glue should have been designed all along. The advantages become clear when we consider how G3 avoids the problems G2 was designed to solve, not with a series of patches to an initial inadequate design, but *by* its initial design.

- Whereas the *designers* of G2 had to extend a version of System F to two sorts to avoid the quantifiers from 'wrapping around themselves', G3 can handle any number of base type the *users* are free to chose, $t/1$ and $e/1$ being a fairly natural choice that incidentally prevented us from running into such problems.

- Glue analyses require support of label sensitivity and quantification over labels from the formal system. Propositional logic does not support quantification and second-order logic supports second-order quantification i.e. quantification over types. G3 is a first-order system where labels are arguments of the base types ensuring label sensitivity supporting first-order quantification where variables range not over complex types, but over individuals, which for our purposes are labels of syntactic structures.

G3 and G2 also represent different approaches to the problem of finding the ideal type-theoretic Glue framework on a different level. G3 is a general-purpose type system, which we have decided to use with $e/1$ and $t/1$ as the base types as a formal system for meaning composition. G2 is a special-purpose system specifically designed for meaning composition: the e and t sorts are part of the system, the restricted version of second-order quantification has a very specific task to serve (the Glue analysis of quantification in natural languages) which also explains why it is only supported for the sort for which this is needed (t). A formal system designed for a specific purpose may have certain advantages over a general-purpose one. But G2 is neither simpler than G3, nor does its design reveal the nature of the problem better than $(e/1, t/1)$ G3 (the opposite is true), except that one can guess from the design of G2 that quantification over e types is unnecessary in Glue.

[12] The interested reader may want to check out what the 'DRS' corresponding to *sleep everyone* $\lambda x.x$ looks like.

Next we turn our attention to the syntax-Glue interface. That interface has been traditionally been through 'semantic projections' in LFG although what a semantic projection is has changed over the years. Semantic projections of f-structures were directly *given a meaning* in G0: "The semantic projection of an f-structure, written with a subscript σ, is a representation of the meaning of that f-structure."(Dalrymple et al., 1993). But in G1 'semantic projections' lost their initial meaning (excuse the pun). Semantic projections in G1 were not the meaning of their corresponding f-structures but attribute-value structures *associated* to a meaning through constraints using the $\rightsquigarrow$ relation: "semantic projections can carry more information than just the association to the meaning for the corresponding f-structure"(Dalrymple et al., 1995a). One use of the internal structure of G1 semantic projections was in the treatment of noun phrases having a generalised quantifier and a common noun as its prominent constituents. The semantic projection of such a noun phrase would have a VARiable and a RESTRictor attribute. [13] One may reasonably ask what the information held in these attributes is. Well, they really are dummy attributes that hold no information whatsoever. But in G1, at least, they were associated with a meaning through the $\rightsquigarrow$ relation: VAR would be, say, associated to a universally quantified X and RESTR to, say, $R(X)$ where R is the predicate that corresponds to the meaning of the common noun.

In G2 and G3 their role is even more insignificant: VAR and RESTR attributes are not there to be directly or indirectly associated with a meaning, nor to hold any kind of information, but to form the Glue type of a semantic contribution. In G3 we actually take advantage of that change: the same label can be used for example as a parameter to both an $e/1$ and a $t/1$ base-type constructor. Interestingly, that would not have been of any use if the Glue-semantics interface had not changed when G2 replaced G1. In G1 semantic projection each label was associated with a semantic expressions, which would have meant it could either be an entity or a truth value, not both. In G2 and G3 there is no such restriction stemming from the syntax-Glue interface, but in G2 the restriction remained in the typing system.

Post-G0 semantic projections with dummy attributes are not the best characteristic of Glue. van Genabith and Crouch(1997) avoided to explain what semantic projections are when they presented examples involving generalised quantifiers. Andrews(2003) goes further. He directly attacks semantic projections, which he disposes of, but not having solved the problem of dummy attributes he transfers them into f-structures, something the Glue designers wanted to avoid. Where to put the dummy attributes is also a concern also when Glue is introduced into a grammatical framework other than LFG. G3 can work with G2-style 'semantic projections' and it can also work with dummy attributes. We have chosen to use neither as this simplifies the syntax-Glue interface.

[13] The Glue analysis of common nouns using VAR and RESTR attributes was the first, but not the only Glue analysis using dummy attributes. For example, Dalrymple(2001) uses such attributes also in her treatment of modification.

We will see the three ways G3 helps us avoid dummy attributes, which arguably, having no informational content, have no role in *any* structure.The true problem they have been used to solve in G1 & G2 is that new types are needed for some analyses. G3 offers three different ways of dealing with this need. Providing a type-based alternative to dummy attributes opens the possibility for Glue without them, and possibly alsofor LFG Glue without G2-style 'semantic' projections.

Every child (25)

f-structure	*G2-style semantic projection*
$s : \left[\begin{array}{ll} \text{SPEC} & q : \text{'EVERY'} \\ \text{PRED} & n : \text{'BABY'} \end{array} \right]$	$s_\sigma : \left[\begin{array}{ll} VAR & v : [\,] \\ RESTR & r : [\,] \end{array} \right]$

Typing Context:
$every : ? \multimap (e_s \multimap t_\alpha)$,
$child : ?$

1. G3 is not restricted to two base type constructors, $e/1$ and $t/1$. In a situation where we need new types, we can introduce new type constructors. For example, some possibilities for the type of *child* include $var_s \multimap restr_s$ and cn_s, similar types with different location parameters and types with different base type constructors possible also of a higher arity such as $e_{s,c} \multimap t_s$ ($e/1$ and $e/2$ are distinct base type constructors).
2. An individual (label in G3) in first-order logic may be a constant, a variable, or the result of a function of individuals. If *socrates* is an individual, $wife(socrates)$ is another individual. Similarly if s is a G3 label, $var(s)$ and $restr(s)$ are also labels, if we relax the definition of 'label' to be any individual in the type system of G3. So without introducing new base type constructors we could give *child* the type $e_{var(s)} \multimap t_{restr(s)}$.
3. In G2 a label is either of the e or the t sort. So L labels give L base types in G2. In G3 labels are potential arguments to base type constructors. If there are C such base type constructors, L labels give $L \times C$ base types in G3. For example, the analysis for common nouns used in this paper refers only to the NP f-structure label but to two types: $e_s \multimap t_s$ (NP e-t). If we allow ourselves to also refer to the common noun f-structure label we have three more possibilities as can be seen in the table below:[14]

Analysis	G3	G2
NP et	$e_s \multimap t_s$	N/A
CN et	$e_c \multimap t_b$	N/A
NP e-CN t	$e_s \multimap t_c$	$s_e \multimap b_t$
CN e-NP t	$e_c \multimap t_s$	N/A

[14] An unexpected discovery was that one of these four analyses can also be used in G2, namely the last one. So in some cases there may be ways of avoiding dummy fields in G2 too. But that does not change that in G3 there will always be more choices. In particular, Glue researches may feel more comfortable with the second and third analysis, than with the first and fourth where the argument of the common noun is the same as that of the verb (assuming an intransitive verb).

5. Conclusions

The Glue approach has been a breakthrough in syntax-semantics interface research in general and for LFG a particularly important development. The move from G1 to G2 is relatively recent and a move to G3 should really have to have good supporting reasons. G3 does not rely on strange, however well motivated, restrictions and it is exactly what it appears to be. The standard Glue typing system has the look of a second-order system, borrowing the notation of System F, but due to the restrictions placed on quantification, it is in fact directly-interchangeable equivalent to a fragment of the proposed system. All existing analyses in the Glue literature can be effortlessly transferred to the proposed system. Moreover, the proposed system makes possible more elegant alternative analyses, and may help researchers develop analyses that would have not been accommodated very well by the old system, for example analyses involving type predicates other than e and t and analyses with type predicates involving more than one label.

It is up to the greater Glue research community to cast its vote on the issue of notation change. Predicate logic is familiar on its own, but has not really been commonly used as a type system. I believe the Glue approach is one of these very few situations where it is useful, given an interpretation of predicate arguments as f-structure labels. This completes the move from G1 to a system comparable to CG started in Dalrymple et al.(1997b) as now Glue has a formalism without artificial constraints and irregularities.

6. Acknowledgements

I am grateful to Mary Dalrymple for a number of very useful discussions during the last two years that not only provided an essential understanding of LFG and the Glue approach, but also invaluable enthusiasm and encouragement. I hope the result lives up to her expectations. Thanks also to Avery Andrews, Ash Asudesh, Dick Crouch and John Lamping for reading and commenting on a previous version of this work.

References

Andrews, A.: 2003, 'Glue Logic, Projections and Modifiers'. Unpublished compuscript.

Asudeh, A. and R. Crouch: 2001, 'Glue semantics: A general theory of meaning composition'. Talk given at Stanford Semantics Fest 2, March 16, 2001.

Asudeh, A. and R. Crouch: 2002, 'Glue semantics for HPSG'. In: F. van Eynde, L. Hellan, and D. Beermann (eds.): *Proceedings of the 8th international HPSG conference*. CSLI Publications.

Dalrymple, M. (ed.): 1999, *Semantics and Syntax in Lexical Functional Grammar: The Resource Logic Approach*. MIT Press.

Dalrymple, M.: 2001, *Lexical Functional Grammar*, No. 42 in Syntax and Semantics Series. Academic Press.

Dalrymple, M., V. Gupta, F. C. Pereira, and V. Saraswat: 1997a, 'Relating Resource-based Semantics to Categorial Semantics'. In: *Proceedings of the Fifth Meeting on Mathematics of Language (MOL5)*. Schloss Dagstuhl, Saarbrücken, Germany. An updated version was printed in Dalrymple(1999).

Dalrymple, M., J. Lamping, F. C. Pereira, and V. Saraswat: 1995a, *A deductive account of quantification in LFG*. Center for the Study of Language and Information, Stanford, California.

Dalrymple, M., J. Lamping, F. C. Pereira, and V. Saraswat: 1995b, 'Linear Logic for meaning assembly'. In: S. Manandha, G. P. Lops, and W. Nutt (eds.): *Proceedings of Computational Logic for Natural Language Processing*. Edinburgh.

Dalrymple, M., J. Lamping, F. C. Pereira, and V. Saraswat: 1997b, 'Quantifiers, anaphora, and intensionality'. *Journal of Logic, Language and Information* **6**(3), 219–273. Reprinted in (Dalrymple, 1999).

Dalrymple, M., J. Lamping, and V. Saraswat: 1993, 'LFG semantics via constraints'. In: *Proceedings of the Sixth Meeting of the European ACL*. University of Utrecht, pp. 97–105.

Frank, A. and J. van Genabith: 2001, 'LL-based semantics construction for LTAG-and what it teaches us about the relation between LFG and LTAG'. In: *Proceedings of the LFG01 conference*. CSLI Publications.

Girard, J.-Y.: 1987, 'Linear logic'. *Theoretical Computer Science* **50**, 1–102.

Howard, W. A.: 1980, 'The formulae-as-types notion of construction'. In: J. Hindley and J. Selden (eds.): *To H.B. Curry: Essays on combinatory logic, lambda calculus and formalism*. Academic Press. Conceived in 1969. Sometimes cited as Howard(1969).

Janssen, T. M. V.: 1997, 'Compositionality'. In: J. van Benthem and A. ter Meulen (eds.): *Handbook of Logic and Language*. Amsterdam: Elsevier, pp. 417–473.

Kaplan, R. M. and J. Bresnan: 1982, 'Lexical Functional Grammar: A formal system for grammatical representation'. In: J. Bresnan (ed.): *The Mental Representation of Grammar Relations*. MIT Press, pp. 173–281.

Klein, E. and I. A. Sag: 1985, 'Type driven translation'. *Linguistics and Philosophy* **8**, 163–201.

Kokkonidis, M.: 2003, 'Glue + λDRT'. Unpublished manuscript.

Montague, R.: 1973, 'The proper treatment of quantification in ordinary English'. In: K. J. J. Hintikka, J. M. E. Moravcsik, and P. Suppes (eds.): *Appoaches to Natural Language*. pp. 221–242. Reprinted in **?**(**?**) and **?**(**?**).

Montague, R.: 1974. New Haven, Connecticut: Yale University Press. Edited and with an introduction by Richard H. Thomason.

Moorgat, M.: 1997, 'Categorial Type Logics'. In: J. van Benthem and A. ter Meulen (eds.): *Handbook of Logic and Language*. Elsevier.

Morril, G. V.: 1994, *Type Logical Grammar: Categorial Logic of Signs*. Dordrecht: Kluwer.

Portner, P. and B. Partee (eds.): 2002, *Formal Semantics: The Essential Readings*. Blackwell.

van Genabith, J. and R. Crouch: 1997, 'How to glue a donkey or porting a dynamic meaning representation language into LFG's linear logic based glue language semantics'. In: *Proceedings of the International Workshop for Computational Semantics*. Tilburg, pp. 52–65.