

A Simple Linear First-Order Type System for Meaning Assembly

Miltiadis Kokkonidis
Oxford University

October 1, 2006

Abstract

Glue (Dalrymple, 1999), a compositional semantics framework based on linear logic (Girard, 1987) has evolved over the years. Evolution came hand-in-hand with simplification. In its most recent incarnation (Kokkonidis, 2006), Glue is nothing more and nothing less than a λ -calculus with a first-order ($-\circ, \forall$) linear type system used to find all terms (potentially corresponding to different readings) that have a given type. It is relatively easy for students to follow a derivation corresponding to a particular term of the appropriate type. However, it is not very easy for them to understand how they can obtain such terms and/or derivations themselves. This is closely related to the issue of how a computer can be instructed to do the same. Unfortunately, current implementation methods have little pedagogical value being significantly more difficult to grasp than the type inference system of Glue. The alternative type inference system for Glue presented here encodes the simple tricks a Glue expert uses, and, as a result, is suitable as both the basis of an implementation and as a teaching tool.

1 Glue as the Syntax-Semantics interface

Syntactic analysis offers one view of a sentence. It also helps in getting a semantic view of the sentence, by assembling the Glue typing context Γ consisting of meaning placeholders and their types and dictating what the target type is. Glue, in turn, finds all distinct (up to α -equivalence) $\beta\eta$ -irreducible terms M that have the target type T given that context:

$$\Gamma \vdash M : T.$$

As each meaning placeholder corresponds to a semantic expression, Glue readings correspond to semantic readings, yet another view of the sentence.

By using each semantic contribution exactly once and by respecting the role it is meant to play as encoded in the labels of syntactic structures that become arguments to the commonly-used base types $e/1$ and $t/1$, Glue enforces the constraints of Compositionality while bridging the gap between syntax and semantics instead of having one interfering with the other.

2 The problem of obtaining Glue readings

The sentence below has five Glue readings. i.e. the five $\beta\eta$ -irreducible terms that can be assigned the target type t_f given the typing context Γ and the rules of First-Order Glue (Figure 1).

Every representative of a firm took a sample. (1)

Typing Context:

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

Readings:

$$\begin{array}{ll} \Gamma \vdash \text{every } (\lambda x.a_1 \text{ firm } \lambda y.\text{rep } y \ x) \lambda x.a_2 \text{ sample } (\text{take } x) : t_f & [\alpha = f, \beta = s, \gamma = f] \\ \Gamma \vdash a_2 \text{ sample } \lambda z.\text{every } (\lambda x.a_1 \text{ firm } \lambda y.\text{rep } y \ x) \lambda x.\text{take } x \ z : t_f & [\alpha = f, \beta = s, \gamma = f] \\ \Gamma \vdash a_2 \text{ sample } \lambda z.a_1 \text{ firm } \lambda y.\text{every } (\text{rep } y) \lambda x.\text{take } x \ z : t_f & [\alpha = f, \beta = f, \gamma = f] \\ \Gamma \vdash a_1 \text{ firm } \lambda y.\text{every } (\text{rep } y) \lambda x.a_2 \text{ sample } \lambda z.\text{take } x \ z : t_f & [\alpha = f, \beta = f, \gamma = f] \\ \Gamma \vdash a_1 \text{ firm } \lambda y.a_2 \text{ sample } \lambda z.\text{every } (\text{rep } y) \lambda x.\text{take } x \ z : t_f & [\alpha = f, \beta = f, \gamma = f] \end{array}$$

$$\begin{array}{c} N : T \vdash N : T \quad (\text{Axiom}) \\ \\ \frac{\Gamma, X : T \vdash E : T'}{\Gamma \vdash \lambda X.E : T \multimap T'} \quad (\multimap\text{Intro.}) \quad \frac{\Gamma \vdash E : T' \multimap T \quad \Gamma' \vdash E' : T'}{\Gamma, \Gamma' \vdash E \ E' : T} \quad (\multimap\text{Elim.}) \\ \\ \frac{\Gamma \vdash E : T}{\Gamma \vdash E : \forall V.T} \quad (\forall\text{Intro.}) \quad \frac{\Gamma \vdash E : \forall V.T}{\Gamma \vdash E : T_{[V:=L]}} \quad (\forall\text{Elim.}) \\ \text{[where } V \notin FV(\text{Types}(\Gamma))\text{]} \end{array}$$

Figure 1: Standard First-Order Glue Type-Inference Rules

The task is to find the $\beta\eta$ -irreducible expressions M of type t_f given Γ . The standard system offers little, if any, guidance. All that is obvious is that because t_f is not a function type and does not start with a quantifier either, neither \multimap -Introduction, nor \forall -Introduction is the first rule (when building the derivation bottom-up) to be used, and neither is the Atom rule because the context contains more than one element. Assuming \forall -Elimination is the first rule to be used, the new typing judgement that would need to be shown to have a derivation would either be something like $\Gamma \vdash M : \forall\alpha.t_\alpha$ or $\Gamma \vdash M : \forall\alpha.t_f$ where M is still unknown and the new target type is even less helpful than t_f in determining the next step. Assuming \multimap -Introduction is the first rule to be used the situation gets dramatically worse: there will now be two typing judgements, $\Gamma_F \vdash F : T \multimap t_f$ and $\Gamma_A \vdash A : T$. The unknown M is being replaced by a combination of two unknowns F and A , a new, unknown, type T is introduced, and Γ is split into Γ_F and Γ_A without any indication of which, if any, of its elements go to one and which to the other. This is very discouraging indeed.

3 Obtaining all readings

The first key concept that someone trying to understand how Glue works needs to grasp is that universally quantified variables are there to offer a kind of polymorphism: each of them can be instantiated to a different label expression in each derivation. Keeping that understanding and assuming each variable symbol is bound uniquely (as we can always do), we make \forall implicit (at the negligible cost of disallowing free variables) and do away with its rules. An assignment function σ is now used that maps a number of variables to label expressions not involving variables in its domain.

The second key concept is that a β -irreducible term is best understood as a list of terms the head (first item) of which is atomic i.e. a constant or variable taken straight from the typing context. The standard convention that $(\dots((F A_1) A_2) \dots A_N)$ can be written $F A_1 A_2 \dots A_N$ is crucial. A term $\lambda X.M$ is β -irreducible iff M is. A term $F_{N-1} A_N$ is β -irreducible iff both F_{N-1} and A_N are and F_{N-1} is not of the form $\lambda X.E$. But if F_{N-1} is not of that form, it will either be atomic or equal to $F_{N-2} A_{N-1}$ for β -irreducible F_{N-2} and A_{N-1} . Continuing like that we see that $F_{N-1} A_N = F A_1 A_2 \dots A_N$, where F is atomic, for some $N > 0$. An atomic F with no arguments ($N = 0$) is also irreducible. So in general a β -irreducible term has the form of a list $F A_1 A_2 \dots A_N$ with an atom as the head and a list of $N \geq 0$ β -irreducible argument terms as its tail.

The third key concept is that of a type T_F ending in a type T . We want to select an atomic F of type T_F such that if we supply it $N \geq 0$ arguments it will return a value of the target type T' .

Arguments	List:Type
0	$every : (e_s \multimap t_s) \multimap (e_s \multimap t_\alpha) \multimap t_{\alpha[\sigma]}$
1	$every A_1 : (e_s \multimap t_\alpha) \multimap t_{\alpha[\sigma]}$
2	$every A_1 A_2 : t_{\alpha[\sigma]}$

In our example only four out of the seven meaning placeholders in Γ have a type ending in the target type (t_f): *every*, a_1 , a_2 , *take*. Of them only *take* is a choice that will not give a reading.

Finally, if we want to disallow η -reducible (sub)terms this can easily be expressed in the \multimap -Introduction rule. However, it is a waste to discover a term $\lambda X.E X$ starting with the \multimap -Introduction rule only to reject it and obtain the equivalent term E , which it contains, again using \multimap -Elimination. Since, up to η -equivalence, the set of terms for an implicational type $S \multimap T$ that one can get by introducing a variable $X : S$ and then λ -abstracting over it contains all terms that can be obtained not doing that, it makes sense to have the \multimap -Introduction rule deal with all implicational target types and constrain \multimap -Elimination to apply to base types only. It would be possible to do all this retaining the property that resulting Glue readings are in η -normal (irreducible) form, but it is simpler and also closer to linguists' practice not to.

Let us see how our system works in practice. Of the four possible heads for a reading, let us chose *every*. The problem is then reduced to splitting Γ into

Γ_1 and Γ_2 so that $\Gamma_1 \vdash A_1 : e_s \multimap t_s$ and $\Gamma_2 \vdash A_2 : e_s \multimap t_f$ and discovering terms A_1 and A_2 . The splitting of Γ comes for free: Γ_2 will be whatever was not used for obtaining A_1 . Let us work on $\Gamma_1 \vdash A_1 : e_s \multimap t_s$. The implicational type means we should use \multimap -Introduction. So now we have $A_1 = \lambda x.A'_1$ and $\Gamma_1, x : e_s \vdash A'_1 : t_s$ to get to. Possible heads for A'_1 are a_1 and rep , but choosing rep leads to a dead-end. So $A'_1 = a_1 A_{1.1} A_{1.2}$. $\Gamma_{1.1} \vdash A_{1.1} : e_r \multimap t_r$ and $\Gamma_{1.2} \vdash A_{1.2} : e_r \multimap t_s$ turn out to be $firm : e_r \multimap t_r \vdash \lambda y.firm\ y : e_r \multimap t_r$ and $rep : e_r \multimap e_s \multimap t_s, x : e_s \vdash \lambda y.rep\ y\ x : e_r \multimap t_s$ respectively. All that remains now is to set Γ_2 to the remaining elements of Γ (a_2 , $sample$, and $take$) and figure out what A_2 is to complete the reading ... Remaining readings are obtained similarly. Clearly, the new system minimizes the search space and guides its user much better.

$$\boxed{
\begin{array}{c}
\frac{\Gamma, X : T \vdash_i E : T'}{\Gamma \vdash_i \lambda X.E : (T \multimap T')} \quad (\multimap Intro.) \\
\\
\frac{\Gamma_1 \vdash_i A_1 : T'_1 \quad \dots \quad \Gamma_N \vdash_i A_N : T'_N \quad F : (T_1 \multimap \dots \multimap T_{N+1}), \Gamma_1, \dots, \Gamma_N \vdash_i F\ A_1 \dots A_N : T_{N+1}[\sigma]}{[T_1[\sigma]=T'_1[\sigma], \dots, T_N[\sigma]=T'_N[\sigma], \text{ and } T_{N+1} \text{ is a base type.}]} \quad (\multimap Elim.)
\end{array}
}$$

Figure 2: New First-Order Glue Type-Inference Rules

4 Conclusions

The challenge faced here was providing a clear, easy-to-follow methodology for obtaining all terms M in some normal form, such that $\Gamma \vdash M : T$ for a given typing context Γ and target type T . The aim is not to have students doing the work of a Glue implementation manually for all their lives; computers can take care of mundane tasks like that. However, an amount of practise will improve their grasp of Glue to the extent that they can confidently analyse linguistic phenomena at the Glue level and understand both the consequences of their own analyses and the existing literature much better. In addition to helping computational linguistics students understand Glue and meaning assembly better, this system provides a clear framework for β -irreducible proofs in a somewhat restricted setting: linear (\forall , \multimap) first-order logic. It can either be used as a teaching tool in that setting as-is, with or without the linguistic motivation, or modified for richer logics.

References

- Dalrymple, M. (Ed.) (1999). *Semantics and Syntax in Lexical Functional Grammar: The Resource Logic Approach*. MIT Press.
- Girard, J.-Y. (1987). Linear logic. *Theoretical Computer Science* 50, 1–102.
- Kokkonidis, M. (2006). First-order glue. *Journal of Logic, Language and Information*. To appear.