Proceedings of SALT 23: 257–275, 2013

Unrestrained β-reduction *

Udo Klein
Bielefeld University

Wolfgang Sternefeld
Tübingen University

Abstract A major argument for syntactic reconstruction is based on the well-known fact that semantic reconstruction by β-reduction is possible only if the term to be substituted for a variable does not contain any variable that would become bound as a result of substitution: e.g., the expression $(\lambda x_2 \forall x_1 P(x_1, x_2))(x_1)$ cannot be β-reduced to $\forall x_1 P(x_1, \underline{x_1})$, since the underlined occurrence of x_1 would become bound. This way, we derive a theoretical argument for syntactic reconstruction. However, syntactic reconstruction is not without its problems, simply because the surface form and the reconstructed form may still differ with respect to other syntactic, semantic, and information theoretic properties. This is particularly troublesome for minimalist theories which do not allow for multiple levels of representation.

In this paper we propose a technique that might help to overcome these difficulties (i.e., the limitation imposed by β-reduction on semantic reconstruction) by defining a translation function **T** for expressions of a predicate logic L_0 with λ-abstraction into expressions of a higher-order language L_1, with the desirable property $\mathbf{T}((\lambda x_2 \forall x_1 P(x_1, x_2))(x_1)) = \mathbf{T}(\forall x_1 P(x_1, x_1))$. In linguistic applications this will facilitate the binding of a pronoun without presupposing c-command. We will sketch a formal proof showing that unrestricted β-reduction is a property of the target expressions in L_1, the translations of L_0 under **T**.

Keywords: beta-reduction, lambda-conversion, semantic reconstruction, syntactic reconstruction, λ-calculus

1 Introduction

In semantics of natural language, λ-abstraction is omnipresent and fundamental; ever since the work of Richard Montague it has resided at the core of compositionality. For example, quantifier raising or quantifying-in crucially relies on λ-abstraction; quantifier raising in turn is essential for the binding of pronouns, and binding itself presupposes a way of identifying variables that is expressed by coindexation. It is

* We gratefully acknowledge financial support by grants of the German Science Foundation SFB 673 for Klein and SFB 833 for Sternefeld.

this connection between lambda abstraction and coindexation that is at issue in our paper. As will be illustrated in this introduction, coindexation poses a problem for the semantics of λ-abstraction in the context of so-called reconstruction.

According to received wisdom, the sentences in (1) get a different interpretation depending on whether or not the pronoun is coindexed with the quantifier:

(1) a. nobody$_k$ doubts that he$_k$ is smart.
b. nobody$_k$ doubts that he$_j$ is smart.

Given that quantifying expressions are interpreted as generalized quantifiers, the representations in (2) differ based on whether the variable is interpreted as bound or as free. The relevant binder cannot be the generalized quantifier itself, which only expresses a relation between sets, but it must be the λ-operator:

(2) a. nobody$_k$ λx_k x_k doubts that x_k is smart.
b. nobody$_k$ λx_k x_k doubts that x_j is smart.

Bound and free variables also play a crucial role in the interpretation of movement. Consider standard examples of topicalization, as in (3):

(3) a. That pizza$_i$, I won't eat t$_i$.
b. Such examples$_i$, I thought you said that Tom believes the explanation needs t$_i$.

Again, λ-abstraction provides for a straightforward semantic interpretation of (3) by converting the trace into a bound variable:

(4) a. That pizza λx_i I won't eat x_i.
b. Such examples λx_i I thought you said that Tom believes the explanation needs x_i.

However, this immediate connection between binding, coindexation and movement is undermined by topicalized sentences like (5):

(5) His mother$_i$ λx_j nobody$_i$ hates x_j.[1]

The problem is that the pronoun in (5) has left the syntactic domain of its binder.

1 Unfortunately, the example seems to be rather marked in English, and even the reconstructed form seems to be marked, compare (ia) with the more natural (ib):

(i) a. ?nobody$_i$ hates his$_i$ mother.
b. nobody$_i$ hates their$_i$/one$_i$'s mother.

But in a language like German, both (iia) and (iib) are perfectly natural and unmarked:

Perhaps more natural examples in English are topicalized sentences. Consider the following scenario. A psychopath has a quite different view of the world. He knows that what he does is violent, but he considers it justified by the circumstances...

(6) That he is alone in his interpretation, no psychopath realises.

Likewise:

(7) a. Quite how socially privileged he$_i$ is, no student$_i$ realizes.
b. That they$_i$ are handicapped, few students$_i$ realize.
c. That he$_i$ is handicapped, no autistic individual$_i$ realizes.

Again, the obvious problem with these constructions is that the pronoun has been moved out of the scope of the binding expression. The intended meaning is of course still represented by coindexation, but it seems that this meaning cannot be derived by interpreting the surface structure.

The reason for this is rooted in the λ-calculus, which permits β-reduction of some term $\lambda x\phi(t)$ to $\phi[x/t]$ only if the term t to be substituted for (all free occurrences of) the variable x in ϕ does not contain any free variables that would become bound as a result of substitution. In (8), the last occurrence of the variable y would end up being bound by $\forall y$ as a result of substituting it for x in $\forall y$ **adore**(y,x), and therefore β-reduction is banned.

(8) $\lambda x \forall y$ **adore**$(y,x)(y) \not\rightarrow_\beta \forall y$ **adore**(y,y)

Without this restriction on β-reduction the λ-calculus would be inconsistent, since it would be possible to derive the equivalence of arbitrary terms (cf. Barendregt 1981: 25).

This restriction can not be used as an argument for syntactic reconstruction, because the surface form and the reconstructed form may still differ with respect to other syntactic and semantic properties. In a framework like Minimalism, this is a contradiction because a potential difference cannot even be formulated in that theory (because surface form is not a level of representation). For example, it has been shown that syntactic reconstruction predicts Condition (C) effects that are not attested in the data, therefore syntactic reconstruction is not a solution for the bound variable problem (cf. e.g., Salzmann 2006). Moreover, there are contexts that do not permit a reconstructed reading because some blocking material intervenes between the topicalized item and the trace. As shown by Heycock (2011), the nature of these interveners must be semantic rather than syntactic.

(ii) a. Niemand$_i$ hasst seine$_i$ Mutter.
b. Seine$_i$ Mutter hasst niemand$_i$.

To be explicit, in many natural language contexts syntactic reconstruction fails because: (i) syntactic movement is highly implausible, as in all sorts of clefts, cf. (9); (ii) it contradicts independent principles of grammar such as Binding Principle C or the licensing conditions for NPIs (e.g., (9b,c)) and idioms (e.g., (9d)); and (iii) reconstruction would not help in other cases of binding without c-command, as in "telescoping" (e.g., (9e)):

(9) a. What [*nobody* did t] was buy a picture of *himself*.
b. NPIs: Buy a/**any* picture of *himself* was what [*nobody* did t].
c. NPIs: ... but steal some/**any*thing, [*nobody* did t]
d. Idioms: (*)What [Mary *didn't* lift t] was *a finger*.
e. Telescoping: The picture of his_i mother that $every_i$ *soldier* kept t wrapped in a sock was not much use to him_i.

The overall conclusion so far is that in some domains syntactic reconstruction is on the wrong track whereas a surface-true semantic approach to reconstruction (augmented with potential semantic restrictions) would give correct results. This surface-true semantic approach calls for a mechanism that allows for extended variable binding, i.e., variable binding without c-command, for example via the trace of some kind of movement that is interpreted by β-reduction.

The alternative that we shall explore in this paper is giving up the idea that pronouns simply denote individuals under an assignment. But unlike the variable-free approach of Jacobson (1999) and the continuations approach of Barker (2002), we aim to keep the assumption that variable binding is analysed in terms of coindexation. We propose that pronouns denote what has been called the "global extension" of a variable in Zimmermann & Sternefeld 2013. Global extensions differ from ordinary local extensions in taking assignment functions as part of the denotation of any expression α, so that given a local extension $[\![\alpha]\!]_g$ its global extension $[\![\alpha]\!]$ denotes $\lambda g [\![\alpha]\!]_g$. Global extensions are needed to show that predicate logic is fully compositional, and they will be taken advantage of in the following to ensure that binding is fully compositional as well. However, as shown in Zimmermann & Sternefeld 2013, there is a price to pay for compositionality, in that the variable assignment itself has to be included into the ontology of first order logic. The point will be made explicit below, in assuming that variable assignments will be included into an extended formal language whose normal extensions are the global extensions of ordinary predicate logic.

Our strategy will be that of indirect interpretation. That is, we will provide for an interpretation of a λ-expression by specifying a translation function **T** that translates first order logic plus λ-expressions into a more complex higher order logic.

(10) NL expression $\xrightarrow{\mathbf{T}(\mathrm{e})} \alpha \in L_0 \xrightarrow{\mathbf{T}(\alpha)} \alpha^+ \in L_1 \xrightarrow{[\![\cdot]\!]} [\![\alpha^+]\!] = [\![\mathbf{T}(\alpha)]\!]$

The formal language L_1 receives its traditional interpretation $[\![\cdot]\!]$ which we presuppose in this paper. It can then be shown that (11) is a special case of a general equivalence between β-reduced and unreduced formulas.

(11) a. $\mathbf{T}([\lambda x \lambda y.P(y,x)](y)) = \mathbf{T}(\lambda y.P(y,y))$
b. $\mathbf{T}([\lambda x \exists y.P(y,x)](y)) = \mathbf{T}(\exists y.P(y,y))$

We proceed as follows: Section 2 introduces assignment functions into the formal language of a sorted and typed predicate logic L_1 with λ-abstraction, and we show that each formula of first order predicate logic can be translated into a type shifted formula of L_1. We demonstrate how this first step can account for most linguistic problems of variable binding by reconstruction. Section 3 attacks the key problem for λ-abstraction, namely the case illustrated in (11), and presents the solution step-by-step. First we deal with iterated abstraction and application, then with quantification, and finally with an asymmetry between binding by quantifiers and binding by λ-abstraction. In section 4 we formally prove that the system defined in section 3 allows for unrestricted semantic reconstruction.

2 Indices and assignment functions

Our point of departure is Bennett 1979, who intended to design a denotation for questions as open formulas, but as an open formula standardly denotes a truth value, such ordinary denotations are unusable for that purpose. However, if open formulas are instead represented as sets of assignment functions (their global extensions), they have enough internal structure to be useful. Assignment functions themselves are, as usual, construed as functions from variables to entities of the usual sort. But now, these variables must denote entities of the model, and for this purpose Bennett simply took integers to be the representatives of variables, namely those integers that normally appear as the subscripts of variables in $x_1, x_2, \ldots x_n$.[2] The correspondence between open formulas of L_0 and sets of assignment functions of L_1 illustrated in (12) is straightforward:

(12) a. L_0: $P(x_1, x_6, x_7)$

2 We could as well have taken the variables themselves as the domain of the assignment function, with variables simply denoting themselves in the manner proposed in Zimmermann & Sternefeld 2013, but at the risk of conflating object language and meta-language. We decided for natural numbers in order to make clear that they stand for arbitrary objects of L_1, having the formal status of constants of a particular type n, whereas quantification still ranges over 'real variables' as explained below.

b. L_1: $\lambda g.P(g(1),g(6),g(7))$

In (12b), the variable g is a function from an index to an individual. Indices (or pointers, sometimes also called discourse markers) are constants of type n, therefore g has type $\langle n,e\rangle$.[3] To start with, we assume that (12b) is the translation of (12a) into the target language L_1. Likewise, the translation of variables x_i (for any $i \in \mathbb{N}$) is $\lambda g.g(i)$ which will also be the translation of pronouns of Natural Language. Such an expression is called a pseudo-variable, for the obvious reason that it does not contain any free variable.

At this point we can already grasp the basic intuition that will enable us to interpret β-reduction in the desired way: the relevant feature of the translation of variables and open formulas is that none of the translations in L_1 will contain any free variables whatsoever. This is the crucial feature of the system: as there are no free variables in the translated formulas, the problematic cases for β-reduction simply do not arise in the target language. Moreover, the interpretation/translation is in an obvious sense equivalent to its source. Let $[\![.]\!]_g$ be the usual interpretation function for L_0. Let $\mathbf{T}(\alpha)$ be the translation of a formula or term into L_1. Given the result of the translation procedure (to be specified precisely further below), namely that $\mathbf{T}(\alpha)$ never contains a free variable, the interpretation function for the resulting formulas of L_1 does not depend on an assignment for variables; this will simply be the function $[\![.]\!]$. In order to compare the standard interpretation of L_0 with its new interpretation via L_1, let both $[\![.]\!]$ and $[\![.]\!]_g$ depend on the same model for constants of L_0 (but this additional index $\mathscr{M}$ is omitted in what follows). The equivalence can then be expressed as in (13):

(13) $[\![\alpha]\!]_g = [\![\mathbf{T}(\alpha)]\!](g) = [\![\mathbf{T}(\alpha)(g)]\!]$ for any assignment function g.[4]

To get the complete picture we must deal with quantifiers. Bennett's analysis is simply a restatement of the usual truth conditions for quantification of L_0, now expressed in L_1 rather than in the meta-language of L_0. Accordingly, the first thing to do is express modified assignments in L_1:

(14) **Modified assignments:**
$g[i/y] := (\iota f)(f(i) = y \wedge \forall n(n \neq i \rightarrow f(n) = g(n)))$

These are needed for stating universal quantification as shown in (15):

(15) **Universal Quantification: (first version)**

3 The functions g are subsets of $\mathbb{N} \times D$ (where D is the domain of entities), whereas the assignment functions of standard predicate logic are subsets of $VAR \times D$ (where VAR is the set of variables).

4 Note that the first g is an expression of the meta language of L_0, the second g belongs to the meta language of L_1 and the third is an expression (a variable) of L_1.

$$\mathbf{T}(\forall x_i \phi) = \lambda g \forall x_i \mathbf{T}(\phi)(g[i/x_i])$$

Note that hitherto we only translated the meta language of L_0 into the object language L_1; the only new device needed to do so is to shift indices (pointers, discourse referents) from the meta language into the language of L_1. The remaining clauses for deriving full-fledged predicate logic are given in (16):

(16) a. $\mathbf{T}(\neg\phi) = \lambda g \neg \mathbf{T}(\phi)(g)$
b. $\mathbf{T}([\phi \wedge \chi]) = \lambda g[\mathbf{T}(\phi)(g) \wedge \mathbf{T}(\chi)(g)]$

It is obvious that up to now nothing has changed in the semantics of logical expressions.

As the reader may verify, the new format already solves most reconstruction problems in that reconstruction of propositions into the domain of quantifiers is a result of λ-abstraction over propositions. In fact, such a move is also essential for any non-syncategorematic and fully compositional treatment of quantifiers. As an illustration, let us return to (7) repeated as (17):

(17) That he$_i$ is smart, nobody$_i$ doubts.

Assume that *doubts* roughly translates as (18) with p_j being a variable of type $\langle\langle n,e\rangle,\langle s,t\rangle\rangle$ for propositions, to be interpreted as the trace of movement bound by the topicalized clause, and $g(i)$ a subject pseudo-variable to be evaluated by nobody$_i$:

(18) $\lambda g.\mathbf{doubt}(g(i), p_j(g))$

Assume that the lexical meaning of *nobody$_i$* is $\lambda q \lambda g' \neg \exists y_i q(g'[i/y_i])$ with q a variable of the same type as p_j above. Now, applying *nobody$_i$* to (18) we derive (19):

(19) $\lambda q \lambda g' \neg \exists y_i q(g'[i/y_i])(\lambda g.\mathbf{doubt}(g(i), p_j(g)))$
$= \lambda g' \neg \exists y_i [\lambda g.\mathbf{doubt}(g(i), p_j(g))](g'[i/y_i])$
$= \lambda g' \neg \exists y_i \mathbf{doubt}(g'[i/y_i](i), p_j(g'[i/y_i]))$
$= \lambda g' \neg \exists y_i \mathbf{doubt}(y_i, p_j(g'[i/y_i]))$

An important issue we are ignoring here is that the index of the variable y has to match the pseudo-variable in subject position (a coindexation which goes under the label of theta marking). The next steps are straightforward: As usual, the effect of movement is captured by λ-abstraction over the free variable in (19). The resulting λ-abstract will then be applied to $\lambda g.\mathbf{smart}(g(i))$ as the translation of *he$_i$ is smart*,

and intensional functional application yields (20):[5]

(20) $\lambda p_j \lambda g \neg \exists y_i \mathbf{doubt}(y_i, p_j(g[i/y_i]))(\lambda g'.\hat{}\mathbf{smart}(g'(i)))$
= $\lambda g \neg \exists y_i \mathbf{doubt}(y_i, \lambda g'.\hat{}\mathbf{smart}(g'(i))(g[i/y_i]))$
= $\lambda g \neg \exists y_i \mathbf{doubt}(y_i, \hat{}\mathbf{smart}(g[i/y_i](i)))$
= $\lambda g \neg \exists y_i \mathbf{doubt}(y_i, \hat{}\mathbf{smart}(y_i))$

This is exactly what were aiming for. More applications of the system just described and further discussion can be found in Sternefeld 2001, an analysis of (9e) is given by Sternefeld (in press).

As should be obvious, intensionality is irrelevant for the problem under discussion, hence we will ignore intensions and dismiss with the semantic type s. Accordingly, propositional variables have the simplified type $\langle\langle n,e\rangle,t\rangle$ and the logic to be developed below is extensional.

3 Unrestricted semantic reconstruction

The goal we are attempting to reach in this paper is more ambitious than the examples discussed above would suggest. What we want to develop is a formal system that not only works for the reconstruction of open propositions but for β-reduction in general. This aim is much more difficult to attain. The problem so far is that for examples like (5) there is simply no open proposition that could be reconstructed; what is needed is the semantic reconstruction of a variable (or more generally, a term) simpliciter.

As the system we are going to develop is quite complex, we will try to motivate each step by showing what goes wrong in a simpler system, developing the translation in a piecemeal fashion. We begin by showing that we need continuations of assignments in order to account for the interplay between quantification and beta reduction. Second, we discuss iterated abstraction and functional application. Third, we show that delayed binding via quantification differs from delayed binding via abstraction, and discuss how to account for this asymmetry, namely by introducing an index set which keeps track of all the indices quantified over.

Let us reconsider quantification in example (21):

(21) $\mathbf{T}(\forall x_7\ \mathbf{adore}(x_7,x_9)) = \lambda g \forall x_7[\lambda g\ \mathbf{adore}(g(7),g(9))(g[7/x_7])]$
$= \lambda g \forall x_7\ \mathbf{adore}(g[7/x_7](7),g[7/x_7](9))$
$= \lambda g \forall x_7\ \mathbf{adore}(x_7,g(9))$

Unfortunately, this result is not yet appropriate to deal with λ-abstraction over

5 Note that the choice of variables g or g' is made for mnemotechnical reasons only; one variable g would indeed suffice to do the job.

indiviuals. The reason is that $g[7/x_7](9) = g(9)$ does not preserve the information about the modified value for 7, which is crucial for unrestrained beta reduction in the formula $\lambda x_9 \forall x_7$ **adore**$(x_7, x_9)(x_7)$. To preserve this information we introduce *continuation functions* c from assignments to assignments. As we will see below, the critical object position will not contain $g(9)$ but $c(g[7/x_7])(9)$, which still allows us to access the modified assignment.

The required extension for atomic formulas is given (22), the obvious modification for quantification is given in (23):

(22) $\mathbf{T}(\mathbf{P}(x_{i_1}, \ldots, x_{i_n})) = \lambda c \lambda g . \mathbf{P}(c(g)(i_1), \ldots, c(g)(i_n))$

(23) **Quantification (second version, to be revised):**
$\mathbf{T}(\forall x_i \alpha) = \lambda c \lambda g \forall x_i [\mathbf{T}(\alpha)(F(c)(i))(g[i/x_i])]$, where

$$F(c)(i)(g)(j) = \begin{cases} g(j), & \text{if } i = j, \qquad (\forall x_i \text{ binds } x_j \text{ in } \alpha) \\ c(g)(j), & \text{else} \qquad\qquad\qquad\quad \text{(no binding)} \end{cases}$$

Nothing changes in case of variable binding, but in the case of other free variables, we can retain the information about the modified assignment g. Let us see how this works in the following example:

(24)

$$\begin{aligned} \mathbf{T}(\forall x_7 \ \mathbf{adore}(x_7, x_9)) &= \lambda c \lambda g \forall x_7 [\mathbf{T}(\mathbf{adore}(x_7, x_9))(F(c)(7))(g[7/x_7])] \\ &= \lambda c \lambda g \forall x_7 \ \mathbf{adore}(F(c)(7)(g[7/x_7])(7), F(c)(7)(g[7/x_7])(9)) \\ &= \lambda c \lambda g \forall x_7 \ \mathbf{adore}(g[7/x_7](7), c(g[7/x_7])(9)) \\ &= \lambda c \lambda g \forall x_7 \ \mathbf{adore}(x_7, c(g[7/x_7])(9)) \\ &\neq \lambda c \lambda g \forall x_7 \ \mathbf{adore}(x_7, c(g)(9)) \end{aligned}$$

As we will see later, it is crucial that the modified assignment has "left a trace" at the position of x_9.

Before continuing it may be useful to have a list of type assignments:

(25)

Variables of L_1:
$\tau(y_1), \tau(y_2), \ldots = e$
$\tau(u) = n$
$\tau(g), \tau(g'), \ldots \langle n, e \rangle$
$\tau(c), \tau(c'), \ldots = \tau(v) = \langle \tau(g), \tau(g) \rangle$
$\tau(h), \tau(h'), \ldots = \langle n, \langle \tau(c), \tau(c) \rangle \rangle$
$\tau(\Psi), \tau(\Psi'), \ldots = \langle \langle n, e \rangle, e \rangle$

Metavariables:
$\tau(i), \tau(j) = n$

Constants of L_1:
all constants of L_0
$\tau(1), \tau(2), \ldots = n$
$\tau(A) = \langle \tau(\Psi), \tau(h) \rangle$
$= \langle \tau(\Psi), \langle n, \langle \tau(c), \tau(c) \rangle \rangle \rangle$
$= \langle \tau(\Psi), \langle n, \langle \tau(c), \langle \tau(g), \tau(g) \rangle \rangle \rangle \rangle \rangle$
$\tau(F) = \langle n, \langle \tau(c), \tau(c) \rangle \rangle$
$= \langle n, \langle \tau(c), \langle \tau(g), \tau(g) \rangle \rangle \rangle$
$= \langle n, \langle \tau(c), \langle \tau(g), \langle n, e \rangle \rangle \rangle \rangle$

The idea behind the translation of λ-abstraction and functional application is illustrated in (26) by the intended results of the translations. The constant A in (26c) will be defined further below. The effect of A can be read off from the equations below; basically it replaces the binding index of x_3 in (26c) by the index of the argument x_4.

(26)

	α		$\mathbf{T}(\alpha)$
a.	$P(x_3)$		$\lambda c\lambda g P(c(g)(3))$
b.	$(\lambda x_3 P(x_3))$		$\lambda h\lambda c\lambda g.P(h(3)(c)(g)(3))$
c.	$(\lambda x_3 P(x_3))(x_4)$		$\lambda h\lambda c\lambda g.P(h(3)(c)(g)(3))(A(\mathbf{T}(x_4)))$
		$=$	$\lambda c\lambda g P(A(\mathbf{T}(x_4))(3)(c)(g)(3))$
		$=$	$\lambda c\lambda g P(c(g)(4))$
d.	$\lambda x_3\forall x_4 P(x_4,x_3)(x_4)$		$\lambda c\lambda g\forall x_4 P(x_4,x_4)$
e.	$\lambda x_3\forall x_5 P(x_5,x_3)(x_4)$		$\lambda c\lambda g\forall x_5 P(x_5,c(g[5/x_5])(4))$

As discussed above, the continuation function c prevents the immediate application of g to an argument by forming $c(g)$ first; the effect of the modification of g (the delayed application of g) will become apparent below. Besides c, we also need an additional function h that operates on the index of the binding variable; this too will be made precise further below. It is sufficient at this point to note that λ-abstraction introduces a new variable h, a function from indices and continuations to continuations, and an index i as one argument of h that represents the index of the binder. Moreover, functional application to a (pseudo-)variable is described by a constant A whose exact nature will be described further below.

Before going into the definition of A, there are two additional complications to be dealt with. When x_4 is substituted for x_3 we need to 'know' whether index 4 has been $\forall$-bound. This requires storing the $\forall$-bound indices in a set M of $\forall$-bound indices. Furthermore, to get the order of arguments right, iterated abstraction needs to be handled separately.

In order to handle the first problem, we slightly modify our definition of quantification by adding a new argument to c, namely the set of indexes that are bound at the point of evaluation. Naturally, this set is empty in atomic formulas:

(27) **Translation of atomic formulas:**
$\mathbf{T}(P(t_1,\ldots,t_n)) = \lambda c\lambda g P(t'_1,\ldots,t'_n)$, where for all i with $1 \leq i \leq n$

$$t'_i = \begin{cases} c(\varnothing)(g)(j) & \text{if } t_i = x_j \text{ for some integer } j \\ t_i & \text{else (i.e., if } \mathrm{t}_i \text{ is a constant of } \mathrm{L}_0) \end{cases}$$

The empty set will be expanded recursively by each index of a quantified variable, as shown in (28):

(28) **Translation of quantification:**
$\mathbf{T}(\forall x_i \alpha) = \lambda c \lambda g \forall x_i . \mathbf{T}(\alpha)(\lambda M . F(c)(M \cup \{i\}))(g[i/x_i])$, where

$$F(c)(M)(g)(j) = \begin{cases} g(j), & \text{if } j \in M \\ c(M)(g)(j), & \text{else} \end{cases}$$

The new definition including M will be taken advantage of only later (when defining the crucial but unfortunately complex function A); nonetheless it will help to illustrate the above definition by looking at the intended result, namely beta reduction into the scope of a quantifier. Consider the following example for delayed binding by quantification, showing that:

$$\mathbf{T}(\lambda x_2 \forall x_1 P(x_1, x_2)(x_1)) = \mathbf{T}(\forall x_1 P(x_1, x_1)) = \lambda c \lambda g \forall x_1 P(x_1, x_1)$$

(29) $\mathbf{T}(\lambda x_2 \forall x_1 P(x_1, x_2)(x_1))$ (translation of application, cf. below)
= $\mathbf{T}(\lambda x_2 \forall x_1 P(x_1, x_2))(A(\mathbf{T}(x_1)))$ (translation of abstraction, cf. below)
= $\lambda h \lambda c[\mathbf{T}(\forall x_1 P(x_1, x_2))(h(2)(c))](A(\mathbf{T}(x_1)))$ (translation of quantification)
= $\lambda h \lambda c[\lambda c' \lambda g' \forall x_1 [\mathbf{T}(P(x_1, x_2))(\lambda M . F(c')(M+1))(g'[1/x_1])](h(2)(c))]$
$(A(\mathbf{T}(x_1)))$ (translation of atomic formulas)
= $\lambda h \lambda c[\lambda c' \lambda g' \forall x_1 [\lambda c'' \lambda g'' P(c''(\varnothing)(g'')(1), c''(\varnothing)(g'')(2))(\lambda M . F(c')(M +$
$1))$
$(g'[1/x_1])](h(2)(c))](A(\mathbf{T}(x_1)))$ (conversion of c'')
= $\lambda h \lambda c[\lambda c' \lambda g' \forall x_1 [\lambda g'' P(\lambda M . F(c')(M + 1)(\varnothing)(g'')(1), \lambda M . F(c')(M +$
$1)(\varnothing)(g'')(2))(g'[1/x_1])](h(2)(c))](A(\mathbf{T}(x_1)))$ (conversion of M, twice)
= $\lambda h \lambda c[\lambda c' \lambda g' \forall x_1 [\lambda g'' P(F(c')(\{1\})(g'')(1), F(c')(\{1\})(g'')(2))(g'[1/x_1])]$
$(h(2)(c))](A(\mathbf{T}(x_1)))$ (conversion of g'', twice)
= $\lambda h \lambda c[\lambda c' \lambda g' \forall x_1 [P(F(c')(\{1\})(g'[1/x_1])(1), F(c')(\{1\})(g'[1/x_1])(2))]$
$(h(2)(c))](A(\mathbf{T}(x_1)))$ (conversion of c')
= $\lambda h \lambda c[\lambda g' \forall x_1 [P(F(h(2)(c))(\{1\})(g'[1/x_1])(1),$
$F(h(2)(c))(\{1\})(g'[1/x_1])(2))]](A(\mathbf{T}(x_1)))$ (def. of F)
= $\lambda h \lambda c[\lambda g' \forall x_1 P(g'[1/x_1](1), h(2)(c)(\{1\})(g'[1/x_1])(2))](A(\mathbf{T}(x_1)))$
(definition of modification)
= $\lambda h \lambda c[\lambda g' \forall x_1 P(x_1, h(2)(c)(\{1\})(g'[1/x_1])(2))](A(\mathbf{T}(x_1)))$ (conversion of h)
= $\lambda c[\lambda g' \forall x_1 P(x_1, A(\mathbf{T}(x_1))(2)(c)(\{1\})(g'[1/x_1])(2))]$ (def. of A, cf. below)
= $\lambda c[\lambda g' \forall x_1 P(x_1, \mathbf{T}(x_1)(F(c)(\{1\})(g'[1/x_1]))))]$ (translation of x_1)
= $\lambda c[\lambda g' \forall x_1 P(x_1, \lambda g . g(1)(F(c)(\{1\})(g'[1/x_1]))))]$ (conversion of g)
= $\lambda c[\lambda g' \forall x_1 P(x_1, F(c)(\{1\})(g'[1/x_1])(1))]$ (definition of F)
= $\lambda c \lambda g' \forall x_1 P(x_1, g'[1/x_1](1))$ (definition of modification)
= $\lambda c \lambda g' \forall x_1 P(x_1, x_1)$

Note that at this point we did not yet make essential use of the fact that M is a set, in contrast to the definition of F in (23). The relevance of M will only unfold later, when discussing (38).

Let us now turn to the tricky part of the framework, namely the definition of λ-abstraction. Naively, one would expect that the $\lambda x_i \ldots$ should correspond to some function $\lambda\Psi \ldots A \ldots$ to be applied to $\mathbf{T}(x_j)$, i.e., $\lambda g.g(i)$; and where A is a function that takes care of the indices of the binder i and the argument j so that x_i factually replaces x_j in the remainder of the formula. Abstracting away from the continuations and additional complications, this intuition can be expressed as in (30):

(30) **Translation of abstraction (to be revised):**
$\mathbf{T}(\lambda x_i \alpha) = \lambda\Psi\lambda g[\mathbf{T}(\alpha)(A(\Psi)(i)(g))]$,
where $\mathbf{T}(\alpha)$ is the translation of α and A is a constant function defined as

$$A(\Psi)(i)(g)(j) = \begin{cases} \Psi(g), & \text{if } i = j \\ g(j), & \text{else} \end{cases}$$

(31) **Translation of functional application (to be revised):**
$\mathbf{T}(\alpha(x_i)) = \mathbf{T}(\alpha)(\mathbf{T}(x_i))$, where $\mathbf{T}(x_i) = \lambda g.g(i)$.

To illustrate these definitions, we show that $\mathbf{T}(\lambda x_1 P(x_1,x_2)(x_3)) = \mathbf{T}(P(x_3,x_2))$:

(32)
$\mathbf{T}(\lambda x_1 P(x_1,x_2)(x_3))$ (by translation of functional application)
$= \mathbf{T}(\lambda x_1 P(x_1,x_2))(\mathbf{T}(x_3))$ (by translation of x_3)
$= \mathbf{T}(\lambda x_1 P(x_1,x_2))(\lambda g'.g'(3))$ (by translation of abstraction)
$= \lambda\Psi\lambda g P(A(\Psi)(1)(g)(1),A(\Psi)(1)(g)(2))(\lambda g'.g'(3))$ (by definition of A)
$= \lambda\Psi\lambda g[P(\Psi(g),g(2))](\lambda g'.g'(3))$ (conversion of Ψ)
$= \lambda g[P(\lambda g'.g'(3)(g),g(2))]$ (conversion of g')
$= \lambda g[P(g(3),g(2))]$ (translation of atomic formulas)
$= \mathbf{T}(P(x_3,x_2))$

Let us turn next to iterated abstraction. Recall that

(33) $\mathbf{T}(\lambda x_1 P(x_1,x_2)) = \lambda\Psi\lambda g P(A(\Psi)(1)(g)(1),A(\Psi)(1)(g)(2))$

What we want as a translation of $\lambda x_2 \lambda x_1 P(x_1,x_2)$ is:

(34) $\mathbf{T}(\lambda x_2 \lambda x_1 P(x_1,x_2)) =$
$\lambda\Psi'\lambda\Psi\lambda g P(A(\Psi')(2)(A(\Psi)(1)(g))(1),A(\Psi')(2)(A(\Psi)(1)(g))(2))$

As the reader may easily verify, applying this term to $\lambda g'.g'(1)$, the translation $\mathbf{T}(x_1)$ of x_1, we get the correct result, namely

(35) $\lambda\Psi'\lambda\Psi\lambda gP(A(\Psi')(2)(A(\Psi)(1)(g))(1),A(\Psi')(2)(A(\Psi)(1)(g))(2))(\mathbf{T}(x_1))$
(definition of A, definition of $\mathbf{T}$)

= $\lambda\Psi'\lambda\Psi\lambda gP(A(\Psi)(1)(g)(1),\Psi'(A(\Psi)(1)(g)))(\lambda g'g'(1))$ (def. of A)

= $\lambda\Psi'\lambda\Psi\lambda gP(\Psi(g),\Psi'(A(\Psi)(1)(g)))(\lambda g'g'(1))$ (conversion of Ψ')

= $\lambda\Psi\lambda gP(\Psi(g),\lambda g'.g'(1)(A(\Psi)(1)(g)))$ (conversion of g')

= $\lambda\Psi\lambda gP(\Psi(g),A(\Psi)(1)(g)(1))$ (definition of A)

= $\lambda\Psi\lambda gP(A(\Psi)(1)(g)(1),A(\Psi)(1)(g)(1))$ (conversion of g' below)

= $\lambda\Psi\lambda g[\lambda g'P(g'(1),g'(1))(A(\Psi)(1)(g))]$ (definition of atomic formulas)

= $\lambda\Psi\lambda g[\mathbf{T}(P(x_1,x_1))(A(\Psi)(1)(g))]$ (definition of $\mathbf{T}$, abstraction)

= $\mathbf{T}(\lambda x_1 P(x_1,x_1))$

Focusing just on the terms x_1 and x_2, we see that their translation in an atomic formula is $g(1)$ and $g(2)$. Recall that we omitted c because the continuation is irrelevant for the argument. After the first abstraction over x_1, the corresponding terms are $A(\Psi)(1)(g)(1)$ and $A(\Psi)(1)(g)(2)$, respectively. After the second abstraction over x_2, what we want to get are the terms $A(\Psi')(2)(A(\Psi)(1)(g))(1)$ (which by definition of A is identical to $A(\Psi)(1)(g)(1)$) and $A(\Psi')(2)(A(\Psi)(1)(g))(2)$ (which by definition of A is identical to $\Psi'(A(\Psi)(1)(g))$), respectively, which after application to $\lambda g'g'(1)$ both turn to $A(\Psi)(1)(g)(1) = \Psi(g)$.

So far, so good. But now the crucial question is how to arrive at (34) in a systematic (recursive) way on the basis of (33). According to our preliminary definition of abstraction and application, the only terms we can substitute in $A(\Psi)(1)(g)(1)$ are Ψ and g, so by abstracting over x_2 we need to get from the term $A(\Psi)(1)(g)(1)$ to the term $A(\Psi')(2)(A(\Psi)(1)(g))(1)$ just by substituting Ψ and g. As it happens, this is not feasible! This problem calls for a major conceptual revision concerning the division of labor between λ-abstraction and functional application.[6]

6 Note that our interpretation of lambda abstraction is non-standard as it does not satisfy alpha equivalence. For example, (iiia) and (iiib)

(iii) a. $\lambda x_2\lambda x_1.P(x_1,x_2)$
b. $\lambda x_2\lambda x_3.P(x_3,x_2)$

are equivalent in L_0, but this cannot hold for the respective translations in L_1. If this were the case, the results of applying (iiia) and (iiib) to x_1 should be identical, but as we have argued above, this outcome is unwarranted. This difference of interpretation also implies that depending on the choice of α it does not always hold that $\mathbf{T}(\lambda x_i\ldots(\alpha)) = \mathbf{T}(\lambda x_i\ldots)(\mathbf{T}(\alpha))$. It follows that the system is not alphabetically invariant when it comes to binding by lambda operators. In particular, the attempt to assimilate the format of lambda abstraction of L_1 to that of L_0 by saying that a set of individuals (or the characteristic function thereof) in L_0 should correspond to a set of pseudo-variables defined by something like $\lambda\Psi\mathbf{T}(\alpha)(A(\Psi))$ would not make much sense as this similarity disappears when it comes to functional application.

What we can do instead at this point is introduce variables h for $A(\Psi)$ and h' for $A(\Psi')$, so that our task can be reformulated as getting from $h(1)(g)(1)$ to $h'(2)(h(1)(g))(1)$ by replacing h and/or g.[7] This can be achieved by substituting h in $h(1)(g)(1)$ with $\lambda u\lambda v.h'(2)(h(u)(v))$, where u is a variable of type n and v is a variable of type $\langle n,e\rangle$. To see this, note that:

(36)
$$\begin{array}{lll} & (\lambda u\lambda v.h'(2)(h(u)(v)))(1)(g)(1) & (\beta\text{-reduction of } u)\\ = & (\lambda v.h'(2)(h(1)(v)))(g)(1) & (\beta\text{-reduction of } v)\\ = & h'(2)(h(1)(g))(1) & \end{array}$$

To make this work we need to first revise the translation of functional application. Instead of stipulating that $\mathbf{T}(\alpha(x_i)) = \mathbf{T}(\alpha)(\mathbf{T}(x_i))$ we now say that $\mathbf{T}(\alpha(x_i)) = \mathbf{T}(\alpha)(A(\mathbf{T}(x_i)))$. This involves a conceptual shift that moves A from the definition of abstraction to that of application. Second, adjusting the translation of abstraction requires a definition by cases, one of them being iterated abstraction as discussed above, the other being the simple case when α in $\lambda x_i\alpha$ is of type t. Here are our final definitions for abstraction and application:

(37) **Translation of abstraction (final version):**

$$\mathbf{T}(\lambda x_i\alpha) = \begin{cases} \lambda h\lambda c[\mathbf{T}(\alpha)(h(i)(c))], & \text{if } \alpha \text{ has type } t\\ \lambda h\lambda h'[\mathbf{T}(\alpha)(\lambda u\lambda v h(i)(h'(u)(v)))], & \text{else} \end{cases}$$

(38) **Translation of functional application (final version):**
$\mathbf{T}(\alpha(x_i)) = \mathbf{T}(\alpha)(A(\mathbf{T}(x_i)))$, where

$$A(\Psi)(i)(c)(M)(g)(j) = \begin{cases} \Psi(F(c)(M)(g)), & \text{if } i = j\\ c(M)(g)(j), & \text{else} \end{cases}$$

An additional twist comes in with the inclusion of F and M in the definition of A; this is motivated by a certain asymmetry between delayed quantification and abstraction binding. In order to understand the problem, consider the translation of $\lambda x_2 P(x_5,x_2)(x_1)$, which, as the reader may easily verify, turns out to be $\lambda c\lambda g' P(c(\emptyset)(g')(5),c(\emptyset)(g')(1))$,—the correct result. The crucial point here is that the continuation c blocks the application of g' to the index 1, which is a welcome result; otherwise the variable x_1 would not be accessible for replacement via beta conversion anymore. However, although the result at this point must not be $g(1)$, it is precisely this expression that would be required if x_1 were a bound variable, as would have been the case for example in $\lambda x_2\exists x_1 P(x_5,x_2)(x_1)$. It is precisely this effect that we are after in the context of unrestrained beta reduction. We therefore must know at the point of substitution whether the index i belongs to

7 Since A has type $\langle\tau(\Psi),\langle n,\langle\tau(g),\tau(g)\rangle\rangle\rangle$ it follows that $A(\Psi)$ is of type $\langle n,\langle\tau(g),\tau(g)\rangle\rangle$, so the variable h is of type $\langle n,\langle\tau(g),\tau(g)\rangle\rangle$, too.

a bound variable, and this is precisely the information M provides. Thus, if Ψ in (37) applies to $F(c)(M)(g)$ and if Ψ is $\lambda g.g(k)$, then F will test whether or not k is in M, and only if it is, the translation will turn out as $g(k)$, otherwise it is $c(M)(g)(k)$, as illustrated by the derivation in (29).

4 Proof of equivalence

Given the translation function **T** as defined above, we now sketch a proof that for each $\alpha \in L_0$ the target expressions $\mathbf{T}(\alpha)$ is semantically equivalent to an expression $\mathbf{T}(\alpha')$, where α' results from α by unrestricted β-reduction. We first have to define the syntactic operation that converts α to α'.

(39) **Definition (unrestricted substitution $[x//y]\alpha$)**:

a. If $\alpha = P(t_1,\ldots,t_n)$ is an atomic L_0-formula (with P an n-ary relation symbol), and $t_1,\ldots,t_n$ terms, then $[x//y]P(t_1,\ldots,t_n) = P(t'_1,\ldots,t'_n)$, where for all t'_i with $1 \leq i \leq n$: $t'_i = \begin{cases} y, & \text{if } t_i = x \\ t_i, & \text{else} \end{cases}$

b. $[x//y]\neg\alpha = \neg[x//y]\alpha$

c. $[x//y](\alpha \wedge \beta) = [x//y]\alpha \wedge [x//y]\beta$

d. $[x//y]\forall x_i\alpha = \begin{cases} \forall x_i\alpha, & \text{if } x = x_i \\ \forall x_i[x//y]\alpha, & \text{else} \end{cases}$

e. $[x//y]\lambda x_i\alpha = \begin{cases} \lambda x_i\alpha, & \text{if } x = x_i \\ \lambda x_i[x//y]\alpha, & \text{else} \end{cases}$

f. $[x//y](\alpha(z)) = [x//y]\alpha([x//y]z)$

(40) **Definition (unrestricted reduction r)**:

a. if α is atomic L_0-formula, then $\mathbf{r}(\alpha) = \alpha$

b. if $\alpha = \neg\alpha'$ is L_0-formula, then $\mathbf{r}(\alpha) = \neg\mathbf{r}(\alpha')$

c. if $\alpha = \beta \wedge \gamma$ is L_0-formula, then $\mathbf{r}(\alpha) = \mathbf{r}(\beta) \wedge \mathbf{r}(\gamma)$

d. $\mathbf{r}(\forall x_i\alpha) = \forall x_i\mathbf{r}(\alpha)$

e. $\mathbf{r}(\lambda x_i\alpha) = \lambda x_i\mathbf{r}(\alpha)$

f. $\mathbf{r}(\lambda x_i\alpha(x_z)) = [x_i//x_z]\mathbf{r}(\alpha)$

Example:

(41)	$\mathbf{r}(\lambda x_2\lambda x_3\forall x_2P(x_2,x_3)(x_2)(x_3))$	(definition of **r**, clause f)
=	$[x_2//x_3]\mathbf{r}(\lambda x_3\forall x_2P(x_2,x_3)(x_2))$	(definition of **r**, clause f)
=	$[x_2//x_3][x_3//x_2]\mathbf{r}(\forall x_2P(x_2,x_3))$	(definition of **r**, clause d)
=	$[x_2//x_3][x_3//x_2]\forall x_2\mathbf{r}(P(x_2,x_3))$	(definition of **r**, clause a)
=	$[x_2//x_3][x_3//x_2]\forall x_2P(x_2,x_3)$	(definition of substitution)
=	$[x_2//x_3]\forall x_2[x_3//x_2]P(x_2,x_3)$	(definition of substitution)

= $[x_2//x_3]\forall x_2 P(x_2,x_2)$ (definition of substitution)
= $\forall x_2 P(x_2,x_2)$

(42) **Lemma (reduction)**:
Let R be the smallest set of formulas of L_0 such that:

a. if α is an atomic formula of L_0, then $\alpha \in R$
b. if $\alpha \in R$, then $\neg\alpha \in R$
c. if $\alpha \in R$ and $\beta \in R$, then $\alpha \wedge \beta \in R$
d. if $\alpha \in R$, then $\lambda x_i \alpha \in R$ (for any x_i)
e. if $\alpha \in R$, then $\forall x_i \alpha \in R$ (for any x_i)

Then for all $\alpha \in L_0$ it holds that $\mathbf{r}(\alpha) \in R$.

(43) Proof of reduction lemma: by induction on the structure of α.

a. Base case: if α is an atomic formula, then $\mathbf{r}(\alpha) = \alpha$ (by definition of $\mathbf{r}$), and therefore $\alpha \in R$ (by definition of the set R)
b. Let $\mathbf{r}(\alpha) \in R$. Then $\mathbf{r}(\neg\alpha) = \neg\mathbf{r}(\alpha)$ (definition $\mathbf{r}$), and since by ind. hypothesis $\mathbf{r}(\alpha) \in R$ we also have by definition of R that $\neg\mathbf{r}(\alpha) \in R$.
c. Let $\mathbf{r}(\alpha) \in R$ and $\mathbf{r}(\beta) \in R$. Then by definition of $\mathbf{r}$ we have $\mathbf{r}(\alpha \wedge \beta) = \mathbf{r}(\alpha) \wedge \mathbf{r}(\beta)$, and since by ind. hypothesis $\mathbf{r}(\alpha) \in R$ and $\mathbf{r}(\beta) \in R$ we also have by definition of R that $\mathbf{r}(\alpha) \wedge \mathbf{r}(\beta) \in R$.
d. Let $\mathbf{r}(\alpha) \in R$. Then $\mathbf{r}(\lambda x_i \alpha) = \lambda x_i \mathbf{r}(\alpha)$ (definition $\mathbf{r}$), and since by ind. hypothesis $\mathbf{r}(\alpha) \in R$ we also have by definition of R that $\lambda x_i \mathbf{r}(\alpha) \in R$.
e. Let $\mathbf{r}(\alpha) \in R$. Then $\mathbf{r}(\forall x_i \alpha) = \forall x_i \mathbf{r}(\alpha)$ (definition $\mathbf{r}$), and since by ind. hypothesis $\mathbf{r}(\alpha) \in R$ we also have by definition of R that $\forall x_i \mathbf{r}(\alpha) \in R$.
f. Let $\mathbf{r}(\lambda x_i \alpha) \in R$. Then by definition of $\mathbf{r}$ we have $\lambda x_i \mathbf{r}(\alpha) \in R$. Therefore, $\mathbf{r}(\alpha) \in R$. Since substitution does not change membership in R, it follows further that for any x, y: $[x//y]\mathbf{r}(\alpha) \in R$, and so also for x_i, t, showing that $[x_i//t]\mathbf{r}(\alpha) \in R$, and by definition of $\mathbf{r}$ we have $\mathbf{r}(\lambda x_i \alpha(t)) \in R$.

What this essentially says is that every reduced formula $\mathbf{r}(\alpha)$ is built from atomic formulas using negation, conjunction, quantification and abstraction (but no application). This is important in the proof of the next theorem.

We now turn to the formulation of the central theorem, showing that the translation $\mathbf{T}(\alpha)$ of an arbitrary L_0-formula α is beta-equivalent to the translation $\mathbf{T}(\mathbf{r}(\alpha))$ of the reduced formula $\mathbf{r}(\alpha)$:

(44) **Theorem**:
Let α be an arbitrary L_0-formula. Then: $\mathbf{T}(\alpha) \equiv_\beta \mathbf{T}(\mathbf{r}(\alpha))$

(45) Proof: by induction on the structure of α.

a. Base case:
Let α be atomic formula. Then $\mathbf{r}(\alpha) = \alpha$, and therefore $\mathbf{T}(\alpha) = \mathbf{T}(\mathbf{r}(\alpha))$.

b. Negation: Assume that $\mathbf{T}(\alpha) \equiv_\beta \mathbf{T}(\mathbf{r}(\alpha))$. We show that $\mathbf{T}(\neg\alpha) \equiv_\beta \mathbf{T}(\mathbf{r}(\neg\alpha))$: $\mathbf{T}(\neg\alpha) \Longrightarrow$ (definition $\mathbf{T}$, negation) $\Longrightarrow \lambda c\lambda g\neg[\mathbf{T}(\alpha)(c)(g)]$ $\Longrightarrow$ (ind. hypothesis) $\Longrightarrow \lambda c\lambda g\neg[\mathbf{T}(\mathbf{r}(\alpha))(c)(g)] \Longrightarrow$ (definition $\mathbf{T}$) $\Longrightarrow$ $\mathbf{T}(\neg\mathbf{r}(\alpha)) \Longrightarrow$ (definition $\mathbf{r}$) $\Longrightarrow \mathbf{T}(\mathbf{r}(\neg\alpha))$

c. Conjunction: Assume that $\mathbf{T}(\alpha) = \mathbf{T}(\mathbf{r}(\alpha))$ and that $\mathbf{T}(\beta) = \mathbf{T}(\mathbf{r}(\beta))$. We show that $\mathbf{T}(\alpha \wedge \beta) = \mathbf{T}(\mathbf{r}(\alpha \wedge \beta))$: $\mathbf{T}(\alpha \wedge \beta) \Longrightarrow$ (definition of $\mathbf{T}$) $\Longrightarrow \lambda c\lambda g[\mathbf{T}(\alpha)(c)(g) \wedge \mathbf{T}(\beta)(c)(g)] \Longrightarrow$ (ind. hypothesis) $\Longrightarrow$ $\lambda c\lambda g[\mathbf{T}(\mathbf{r}(\alpha))(c)(g) \wedge \mathbf{T}(\mathbf{r}(\beta))(c)(g)] \Longrightarrow$ (definition of $\mathbf{T}$) $\Longrightarrow \mathbf{T}(\mathbf{r}(\alpha) \wedge$ $\mathbf{r}(\beta)) \Longrightarrow$ (definition of $\mathbf{r}$) $\Longrightarrow \mathbf{T}(\mathbf{r}(\alpha \wedge \beta))$

d. Quantification: Let α be such that $\mathbf{T}(\alpha) = \mathbf{T}(\mathbf{r}(\alpha))$. Let x_i be an arbitrary variable. We show that $\mathbf{T}(\forall x_i\alpha) \equiv_\beta \mathbf{T}(\mathbf{r}(\forall x_i\alpha))$: $\mathbf{T}(\forall x_i\alpha) \Longrightarrow$ (definition of $\mathbf{T}$) $\Longrightarrow \lambda c\lambda g\forall y_i[\mathbf{T}(\alpha)(\lambda M.F(c)(M+i))(g[i/y_i])] \Longrightarrow$ (induction hypothesis) $\Longrightarrow \lambda c\lambda g\forall y_i[\mathbf{T}(\mathbf{r}(\alpha))(\lambda M.F(c)(M+i))(g[i/y_i])] \Longrightarrow$ (definition of $\mathbf{T}$) $\Longrightarrow \mathbf{T}(\forall x_i\mathbf{r}(\alpha)) \Longrightarrow$ (definition of $\mathbf{r}$) $\Longrightarrow \mathbf{T}(\mathbf{r}(\forall x_i\alpha))$

e. Abstraction: Assume that $\mathbf{T}(\alpha) = \mathbf{T}(\mathbf{r}(\alpha))$. We show that $\mathbf{T}(\lambda x_i\alpha) = \mathbf{T}(\mathbf{r}(\lambda x_i\alpha))$.
First case: α is of type t: $\mathbf{T}(\lambda x_i\alpha) \Longrightarrow$ (definition $\mathbf{T}$) $\Longrightarrow \lambda h_u\lambda c[\mathbf{T}(\alpha)(h_u(i)(c))] \Longrightarrow$ (ind. hypothesis) $\Longrightarrow$ $\lambda h_u\lambda c[\mathbf{T}(\mathbf{r}(\alpha))(h_u(i)(c))] \Longrightarrow$ (definition of $\mathbf{T}$) $\Longrightarrow \mathbf{T}(\lambda x_i\mathbf{r}(\alpha))$ $\Longrightarrow$ (definition of $\mathbf{r}$) $\Longrightarrow \mathbf{T}(\mathbf{r}(\lambda x_i\alpha))$
Second case: α is a λ-term: $\mathbf{T}(\lambda x_i\alpha) \Longrightarrow$ (definition $\mathbf{T}$) $\Longrightarrow$ $\lambda h_u\lambda h_v\mathbf{T}(\alpha)(\lambda j\lambda f.h_u(i)(h_v(j)(f))) \Longrightarrow$ (induc. hypothesis) $\Longrightarrow$ $\lambda h_u\lambda h_v\mathbf{T}(\mathbf{r}(\alpha))(\lambda j\lambda f.h_u(i)(h_v(j)(f))) \Longrightarrow$ (definition $\mathbf{T}$) $\Longrightarrow$ $\mathbf{T}(\lambda x_i\mathbf{r}(\alpha)) \Longrightarrow$ (definition $\mathbf{r}$) $\Longrightarrow \mathbf{T}(\mathbf{r}(\lambda x_i\alpha))$

f. Application: Assume that $\mathbf{T}(\lambda x_i\alpha) = \mathbf{T}(\mathbf{r}(\lambda x_i\alpha))$, for arbitrary x_i and α. We show that for arbitrary x_z it holds that $\mathbf{T}(\lambda x_i\alpha(x_z)) = \mathbf{T}(\mathbf{r}(\lambda x_i\alpha(x_z)))$ $\mathbf{T}(\lambda x_i\alpha(x_z)) \Longrightarrow$ (definition $\mathbf{T}$) $\Longrightarrow \mathbf{T}(\lambda x_i\alpha)(A(\mathbf{T}(x_z))) \Longrightarrow$ (ind. hypothesis) $\Longrightarrow \mathbf{T}(\mathbf{r}(\lambda x_i\alpha))(A(\mathbf{T}(x_z))) \Longrightarrow$ (definition $\mathbf{T}$) $\Longrightarrow$ $\mathbf{T}(\mathbf{r}(\lambda x_i\alpha)(x_z)) \Longrightarrow$ (definition $\mathbf{r}$) $\Longrightarrow \mathbf{T}(\lambda x_i\mathbf{r}(\alpha)(x_z)) \Longrightarrow$ (lemma (46)) $\Longrightarrow \mathbf{T}([x_i//x_z]\mathbf{r}(\alpha)) \Longrightarrow$ (definition of $\mathbf{r}$) $\Longrightarrow \mathbf{T}(\mathbf{r}(\lambda x_i\alpha(x_z)))$

(46) **Lemma**:
For arbitrary x_i, x_z, α it holds that $\mathbf{T}(\lambda x_i\mathbf{r}(\alpha)(x_z)) = \mathbf{T}([x_i//x_z]\mathbf{r}(\alpha))$

For reasons of space we cannot include the proof of this lemma. A complete and longer version of this paper can be downloaded from www.s395910558.online.de/Downloads/beta-reduction-12.pdf and wwwhomes.uni-bielefeld.de/uklein/publications/beta-reduction-12.pdf.

5 Conclusion

Semantic reconstruction via β-reduction inherits (from the definition of β-reduction of the λ-calculus) the restriction that a term t can be substituted for a variable x only if t contains no variable that would become bound as a result of substitution. Given that the alternative approach via syntactic reconstruction is not without its own problems, we conclude that it is desirable to somehow overcome this restriction on semantic reconstruction, in order to allow for semantic reconstruction even in cases where a bound pronoun occurs outside the scope of its binder, for example when it is part of a topicalized noun phrase (a phenomenon we dubbed delayed quantification).

In this paper we propose a way of doing so by translating each expression α of the language L_0 of predicate logic (with λ-abstraction) into an expression $\mathbf{T}(\alpha)$ of a new language L_1. Crucially, the translation $\mathbf{T}$ is set up such that the formulas $\mathbf{T}(\alpha)$ contain no free variables. In particular, a variable x_i of L_0 is translated as $\lambda g.g(i)$ with g a function from entities of type n (i.e., integers) to entities of type e. Since the term $\lambda g.g(i)$ contains no free variables, it can be substituted for any variable without restriction. The main difficulty was in coming up with a novel (non-standard) semantics for abstraction, application and quantification which accounts for delayed abstraction as well as delayed quantification, and thus allows for pronouns to be bound even if they occur outside the syntactic scope of the binder. In the final section we introduce the notion of unrestricted reduction $\mathbf{r}$ (e.g., $\mathbf{r}(\lambda x_2 \forall x_1 P(x_1,x_2)(x_1)) = \forall x_1 P(x_1,x_1)$) and show that the translation of a formula $\alpha \in L_0$ is equivalent to the translation of its unrestricted reduction $\mathbf{r}(\alpha)$.

References

Barendregt, Hendrik P. 1981. *The Lambda Calculus - Its Syntax and Semantics*. Amsterdam: North Holland.

Barker, Chris. 2002. Continuations and the nature of quantification. *Natural Language Semantics* 10. 211–242.

Bennett, Michael. 1979. *Questions in Montague Grammar*. Mimeo. Indiana University Linguistics Club.

Heycock, Caroline. 2011. Relative reconstructions. Presented at the ZAS Reconstruction Workshop July 2011.

Jacobson, Pauline. 1999. Towards a variable free semantics. *Linguistics and Philosophy* 22. 117–185.

Salzmann, Martin. 2006. Resumptive prolepsis: A study in indirect a'-dependencies. *LOT Dissertation Series* 136.

Sternefeld, Wolfgang. 2001. Semantic vs. syntactic reconstruction. In Christian

Rohrer, Antje Roßdeutscher & Hans Kamp (eds.), *Linguistic Form and its Computation*, 145–182. Stanford, CA: CSLI Publications.

Sternefeld, Wolfgang. In press. Telescoping by delayed binding. In Manfred Krifka, Rainer Ludwig & Mathias Schenner (eds.), *Reconstruction Effects in Relative Clauses. Proceedings of the ZAS Workshop on Head Internal Relative Clauses*, Berlin: Akademie Verlag.

Zimmermann, Thomas Ede & Wolfgang Sternefeld. 2013. *Introduction to Semantics. An Essential Guide to the Composition of Meaning*. Berlin, New York: de Gruyter-Mouton.