Closure Under Alpha-Conversion* **

Randy Pollack

Laboratory for Foundations of Computer Science, University of Edinburgh,
The King's Buildings, Edinburgh, EH9 3JZ, Scotland
rap@dcs.ed.ac.uk

1 Introduction

Consider an informal presentation of simply typed λ-calculus as in [Bar92]. Leaving out some of the details, let σ, τ range over simple types, x, y range over a class of term variables, and M, N, range over the Church-style terms. A *statement* has the form $M : \sigma$, where M is the *subject* of the statement and σ is its *predicate*. A *context*, ranged over by Γ, is a list of statements with only variables as subjects. A context is *valid* if it contains only distinct variables as subjects; defined inductively by

$$\textsc{nil-valid} \qquad \bullet \ \mathsf{valid}$$

$$\textsc{cons-valid} \qquad \frac{\Gamma \ \mathsf{valid}}{\Gamma, x{:}\sigma \ \mathsf{valid}} \qquad x \notin \mathsf{Dom}\,(\Gamma)$$

A statement, $M : \sigma$, is *derivable* from context Γ, notation $\Gamma \vdash M : \sigma$, if $\Gamma \vdash M : \sigma$ can be produced using the following rules.

$$\textsc{var} \qquad \frac{\Gamma \ \mathsf{valid}}{\Gamma \vdash x : \sigma} \qquad x{:}\sigma \in \Gamma$$

$$\textsc{lda} \qquad \frac{\Gamma, x{:}\sigma \vdash M : \tau}{\Gamma \vdash [x{:}\sigma]M : \sigma \to \tau}$$

$$\textsc{app} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\ N : \tau}$$

Such a presentation is usually considered formal enough for everyday reasoning. It can be implemented literally as a type synthesis algorithm for $\lambda{\to}$: to compute a type for a variable, look it up in the context; to compute a type for a lambda

* This work was supported by the ESPRIT Basic Research Actions on Logical Frameworks and Types for Proofs and Programs, and by grants from the British Science and Engineering Research Council.

** A version of this paper appears in *Types for Proofs and Programs: International Workshop TYPES'93, Nijmegen, May 1993, Selected Papers*, LNCS 806.

abstraction, compute a type for its body in an extended context; to compute a type for an application, compute types for its left and right components, and check that they match appropriately. Lets use the algorithm to compute a type for $a = [x{:}\tau][x{:}\sigma]x$.

$$\dfrac{\text{FAILURE: no rule applies because } x \in \mathsf{Dom}\,(x{:}\tau)}{\dfrac{\dfrac{\dfrac{x{:}\tau, x{:}\sigma\ \mathsf{valid}}{x{:}\tau, x{:}\sigma \vdash x : ?}}{x{:}\tau \vdash [x{:}\sigma]x : \sigma \to ?}}{\vdash [x{:}\tau][x{:}\sigma]x : \tau \to \sigma \to ?}} \tag{1}$$

This system fails to derive the intended type $\tau \to \sigma \to \sigma$. Notice that $\vdash [x{:}\tau][y{:}\sigma]y : \tau \to \sigma \to \sigma$ is derivable, but the system is not closed under alpha-conversion of subjects.

What went wrong? In directly implementing this system we are taking informal notation too literally: there is no "x" in $[x{:}\tau][x{:}\sigma]x$; the names of bound variables are not meant to be taken seriously. The rule LDA should be read as "in order to type $[x{:}\sigma]M$, choose some suitable alpha-representative of $[x{:}\sigma]M$, ...".

Formal systems with variable binding are implemented on machines as the basis of programming languages and proof checkers, among other applications. It is clear that the concrete syntax that users enter into such implementations, and see printed by the implementation in response, should be formally related to the implemented formal system. Further, users and implementors need an exact and concise description of such a system; informal explanation is not good enough.

The concrete syntax should have good properties. Users of such implemented systems will construct large formal objects with complex binding. The implementation should help in this task, and anamolies of naming such as the small example above make the job more difficult. What do you say to an ML implementation that claims `fn x => fn x => x` is not well typed?

There are several approaches to naming in implementations of formal systems. Perhaps the best known is the use of explicit names, and Curry-style renaming in the definition of substitution. This technique can (probably) be formalized. The difficulty arises when we ignore the distinction between alpha-convertible terms, and treat then as equal.

It is well known that one solution to the problems of alpha-conversion is the use of de Bruijn "nameless variables" [dB72]. Although nameless variables have their partisans for use in metatheoretic study, even those partisans admit that the explanation of substitution of a term for a given variable is painful in such a presentation, although it can be, and has been, carried out elegantly [Alt93,

Hue94][3]. However, the direct use of nameless variables is not a real possibility in pragmatic applications because human users find it difficult to write even small expressions using nameless variables. It is necessary to translate from named syntax to nameless, and then back again to named syntax for pretty printing, and this translation itself must be formalized.

I know of two recent proposals that take names seriously, but avoid the need for alpha-conversion. One proposal, by Coquand [Coq91], follows a style in logic to distinguish between free variables (parameters) and bound variables (variables). This idea has been used to formalize a large theory of Pure Type Systems, including reduction, conversion and typing [MP93]. This formalization does distinguish between alpha-convertible terms, and the typing judgement is indeed closed under alpha-conversion. The other proposal, by Martin-Löf [Tas93] goes much further, not only using explicit names, but also explicit substitutions, i.e. making the notion of substitution a part of the formal system (as originally proposed for nameless terms in [ACCL91]). Unfortunately the system of [Tas93], in its current formulation, is not closed under alpha-conversion; e.g. it fails to derive judgement (7) in section 3.

Finally, there is a recent proposal [Gor93] of a formalization mixing nameless terms and named variables in such a way that named terms are equal up to alpha-conversion.

1.1 The Constructive Engine

The Constructive Engine [Hue89] is an abstract machine for type checking the Calculus of Constructions. It is the basis for the proofcheckers Coq [DFH+93] and LEGO [LP92]. Among its interesting aspects are:

1. the non-deterministic rules of the underlying type theory are converted into a deterministic, syntax-directed program
2. this syntax-directed program implements a relation equivalent to the type theory, but with very much smaller derivations
3. external *concrete* syntax with explicit variable names is translated into internal *abstract* syntax of *locally nameless* terms, that is, local binding by de Bruijn indexes, and global binding by explicit names
4. an efficient technique for testing conversion of locally nameless terms (with special attention to the treatment of definitions)

The basic ideas of the first and second of these points appeared in early writing of Martin-Löf. The first point has been studied extensively [Pol92, vBJMP94] for the class of Pure Type Systems. The second point is addressed in [vBJMP94, Pol94]. The fourth point (which is clearly the remaining limiting factor in pragmatic implementations of proof checkers) has not received any theoretical attention to my knowledge, although [Hue89, dB85] suggest interesting ideas.

[3] For an example of metatheory where nameless variables are very inconvenient, see the discussion of the Thinning Lemma in [MP93]

In this note I focus on the third item above, the relationship between concrete syntax and abstract syntax in the Constructive Engine. The use of locally nameless style for internal representation of terms is one of the basic decisions of the Constructive Engine, but perhaps reflects more Huet's interest in experimenting with de Bruijn representation than any ultimate conviction that they are the "right" notation for implementing a type checker. I do not want to study the pros and cons of this representation for efficient typechecking, but only to muse over the relationship between concrete terms, their abstract representations, and their (abstract) types.

Plan of the paper. In the next section we discuss simply-typed lambda calculus, $\lambda\!\!\rightarrow$. After presenting several concrete and abstract presentations of its typing rules, we derive a Constructive Engine for $\lambda\!\!\rightarrow$, and consider some variations.

In section 3 we consider the same issue for Pure Type Systems (PTS). There is one new problem in the case of dependent types. We explain a constructive engine for dependent types, and show how to make it closed under alpha-conversion of terms.

Acknowledgement A careful and knowledgeable referee made many useful suggestions.

2 Simply typed lambda calculus

There is a crude way to close the relation $\vdash$ of the Introduction under alpha-conversion of subjects, by adding a rule

$$\textsc{alpha} \qquad \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash N : \sigma} \qquad M \stackrel{\alpha}{=} N$$

where $M \stackrel{\alpha}{=} N$, alpha-conversion, must also be defined by some inductive definition. Such solutions are heavy, and not ideal for either implementation or formal meta-reasoning. Instead of reasoning about three or five rules, we'll have to reason about all the rules for $\stackrel{\alpha}{=}$ as well. Further, rules such as ALPHA, that are not syntax-directed, are hard to reason about: being non-deterministic, they allow many derivations of the same judgement, which sometimes prevents proof by induction on the structure of derivations.

2.1 A concrete presentation closed under alpha-conversion

Another approach is to formalize the informal meaning of the LDA rule suggested above: choose a sufficiently fresh variable name to substitute for x. Informally, replace LDA by

$$\frac{\Gamma, y{:}\sigma \vdash_s M[y/x] : \tau}{\Gamma \vdash_s [x{:}\sigma]M : \sigma \rightarrow \tau} \qquad y \notin M$$

Substitution of y for x in M must still be defined, and the usual definition involves alpha-conversion. We give a formulation suggested in [Coq91] and formalized in detail in [MP93]. Let p, q, r, range over an infinite set of *parameters*, and x, y, z over variables as before. Parameters and variables are disjoint sets[4]. Define two operations of replacement. Replacing a parameter by a term is entirely textual:

$$\begin{aligned} x[M/p] &= x \\ q[M/p] &= \text{if } p = q \text{ then } M \text{ else } q \\ ([x{:}\sigma]N)[M/p] &= [x{:}\sigma]N[M/p] \\ (N_1\, N_2)[M/p] &= (N_1[M/p])\,(N_2[M/p]) \end{aligned}$$

Replacing a variable by a term does respect the scope of variable binding but does not rename variables to prevent capture:

$$\begin{aligned} x[M/y] &= \text{if } y = x \text{ then } M \text{ else } x \\ q[M/y] &= q \\ ([x{:}\sigma]N)[M/y] &= [x{:}\sigma](\text{if } y = x \text{ then } N \text{ else } N[M/y]) \\ (N_1\, N_2)[M/y] &= (N_1[M/y])\,(N_2[M/y]) \end{aligned}$$

Now replace LDA by

$$\text{S-LDA} \qquad \frac{\Gamma, p{:}\sigma \vdash_s M[p/x] : \tau}{\Gamma \vdash_s [x{:}\sigma]M : \sigma \to \tau} \qquad p \notin M$$

(where $p \in M$ means *textual* occurrence). In the side condition of this rule, it's not necessary to check that $p \notin \mathsf{Dom}(\Gamma)$ because failure of that condition prevents completing a derivation; just choose another parameter, since p does not occur in the conclusion of the rule. It *is* necessary to check $p \notin M$ so that the premiss doesn't bind instances of p that do not arise from x.

We could define beta-reduction, beta-conversion, prove Church-Rosser, subject reduction, etc. [MP93], but for our purposes alpha-conversion is enough:

$$\alpha\text{-REFL} \qquad M \stackrel{\alpha}{=} M$$

$$\alpha\text{-LDA} \qquad \frac{M[p/x] \stackrel{\alpha}{=} M'[p/y]}{[x{:}\sigma]M \stackrel{\alpha}{=} [y{:}\sigma]M'} \qquad p \notin M,\ p \notin M'$$

$$\alpha\text{-APP} \qquad \frac{M \stackrel{\alpha}{=} M' \qquad N \stackrel{\alpha}{=} N'}{M\, N \stackrel{\alpha}{=} M'\, N'}$$

Now we can state and prove $\vdash_s$ is closed under alpha-conversion

[4] Another informality! What is required is that the *terms* p and x be distinguishable, not necessarily that the underlying objects be. Depending on the formalization of terms we might use weaker conditions. In [MP93], where terms are an inductive type with distinct constructors for parameters and variables, we don't require parameters and variables to be distinct. In informal presentations of logic it is common for "term" to be an inductive *relation* on strings over some alphabet; in this case, of course, we require that (distinct) objects of the alphabet be distinct.

Lemma 1 Closure of $\vdash_s$ under alpha-conversion.
If $\Gamma \vdash_s M : \sigma$ and $M \stackrel{\alpha}{=} M'$ then $\Gamma \vdash_s M' : \sigma$

This can be proved following the same outline as a proof of subject reduction (closure under beta-reduction); in fact $\stackrel{\alpha}{=}$ is contained in parallel reduction, so this lemma is a corollary of subject reduction.

$\vdash_s$ still treats parameters seriously:

$$\vdash_s [x{:}\tau][x{:}\sigma]x : \tau \to \sigma \to \sigma \quad \text{but} \quad p{:}\tau, p{:}\sigma \not\vdash_s p : \sigma.$$

This is a different problem, if it is a problem at all. In $\vdash_s$ we have analysed the transition from local variable to global parameter, while the treatment of parameters themselves is the same as in $\vdash$.

An induction principle for $\vdash_s$. An interesting variation on the previous idea is to use generalized induction to truly remove the fresh name from derivations. Replace S-LDA by

$$\text{G-LDA} \qquad \frac{\forall p \notin \mathsf{Dom}(\Gamma) \,.\, \Gamma, p{:}\sigma \vdash_g M[p/x] : \tau}{\Gamma \vdash_g [x{:}\sigma]M : \sigma \to \tau}$$

Notice again the "side condition", this time appearing as an antecedent of the generalized premiss. We don't exclude those p that happen to occur in M, because we must derive $\Gamma, p{:}\sigma \vdash_g M[p/x] : \tau$ for infinitely many p (using that the class of variables is infinite), while M can contain only finitely many p. On the other hand, for $p \in \mathsf{Dom}(\Gamma)$, $\Gamma, p{:}\sigma \vdash_g M[p/x] : \tau$ is not derivable for reasons not having to do with M or τ, so this case must be excluded.

Both $\vdash_s$ and $\vdash_g$ derive more judgements than $\vdash$; in fact $\vdash_s$ and $\vdash_g$ are equivalent.

Lemma 2. *For all Γ, M and σ,*
$$\Gamma \vdash_s M : \sigma \quad \Leftrightarrow \quad \Gamma \vdash_g M : \sigma$$

This fact, to be proved, allows us to use structural induction over $\vdash_g$ as an induction principle for $\vdash_s$ that is stronger than structural induction over $\vdash_s$. Lemma 2 states that $\vdash_s$ and $\vdash_g$ are *extensionally* equivalent, i.e. have the same judgements. Consider proving some extensional property, $\mathcal{P}$, of $\Gamma \vdash_s M : \sigma$, by structural induction. In the case of S-LDA we must justify $\mathcal{P}(\Gamma \vdash_s [x{:}\sigma]M : \sigma \to \tau)$ from the induction hypothesis $\mathcal{P}(\Gamma, p{:}\sigma \vdash_s M[p/x] : \tau)$ for some *particular* p. Using lemma 2 it suffices to justify $\mathcal{P}(\Gamma \vdash_s [x{:}\sigma]M : \sigma \to \tau)$ from the stronger induction hypothesis $\forall p \notin \mathsf{Dom}(\Gamma) \,.\, \mathcal{P}(\Gamma, p{:}\sigma \vdash_s M[p/x] : \tau)$.

Informally, this strength comes from showing that the same judgements are derivable in a system with fewer derivations. It is clear that while there are infinitely many derivations of, for example, $\vdash_s [x{:}\sigma]x : \sigma \to \sigma$, each containing some particular parameter, $\vdash_g [x{:}\sigma]x : \sigma \to \sigma$ has only one derivation that does not contain any particular parameter. Loosely speaking, $\vdash_g$ has a "subformula property" that $\vdash_s$ lacks.

It is this induction principle which justifies our belief that in a judgement $\Gamma \vdash_s M : \sigma$, the occurrence of a parameter in Γ can be treated as binding its occurrences in M, hence the actual parameter used doesn't matter as long as it is sufficiently fresh. Such arguments are used many times in the formalization of named variables in [MP93, Pol94], and are used below in justifying the Constructive Engine.

Proof of lemma 2. Direction $\Leftarrow$ is by induction on the derivation of $\Gamma \vdash_g M : \sigma$.

Direction $\Rightarrow$ is not so easy. It can be proved by induction on the length of the subject, M, as, for every rule of $\vdash_g$, the subject of the conclusion is longer than the subject of any premiss. Such a proof does not extend to the case of Pure Type Systems to be considered in section 3. Here I give a proof that does extend, and is better than the proof given in [MP93]. [Pol94] describes these three different ways to prove such an equivalence in excrutiating detail.

We introduce the machinary of renaming. A *renaming* (ranged over by ϕ) is a function from parameters to parameters such that $\phi p = p$ for all but finitely many p. We extend the action of renamings compositionally to terms and contexts. It's easy to see that *bijective* renamings respect both $\vdash_s$ and $\vdash_g$; in particular, if ϕ is bijective and $\Gamma \vdash_g M : \sigma$, then $\phi\Gamma \vdash_g \phi M : \sigma$. It's a little difficult to construct bijective renamings in general, because not only the parameters that get moved have to be considered, but also those that are fixed; a combinatorial nightmare. However it's clear that any renaming that only swaps parameters, e.g. $\{q \mapsto p,\ p \mapsto q\}$, is bijective.

Now prove $\Gamma \vdash_g M : \sigma$ by structural induction on a derivation of $\Gamma \vdash_s M : \sigma$. All cases are trivial except the rule S-LDA

$$\text{S-LDA} \qquad \frac{\Gamma, p{:}\sigma \vdash_s M[p/x] : \tau}{\Gamma \vdash_s [x{:}\sigma]M : \sigma \to \tau} \qquad p \notin M$$

By induction hypothesis $\Gamma, p{:}\sigma \vdash_g M[p/x] : \tau$. To show $\Gamma \vdash_g [x{:}\sigma]M : \sigma \to \tau$ by G-LDA we need $\Gamma, r{:}\sigma \vdash_g M[r/x] : \tau$ for arbitrary parameter $r \notin \mathsf{Dom}(\Gamma)$.

Taking $\phi = \{r \mapsto p,\ p \mapsto r\}$, we have $\phi(\Gamma, p{:}\sigma) \vdash_g \phi(M[p/x]) : \tau$ is derivable by renaming the induction hypothesis. Thus we are finished if we can show

$$\phi(\Gamma, p{:}\sigma) = \Gamma, r{:}\sigma \qquad \text{and} \qquad \phi(M[p/x]) = M[r/x].$$

Notice $p \notin \mathsf{Dom}(\Gamma)$ (or the premiss of S-LDA could not be derivable), and we also know $r \notin \mathsf{Dom}(\Gamma)$. From these observations it's clear that the first equation holds. For the second equation, notice that if $r = p$ then ϕ is the identity renaming, and we are done, so assume $r \neq p$, and hence $r \notin M[p/x]$ (from the premiss of S-LDA and the assumption $r \notin \mathsf{Dom}(\Gamma)$). Now we use a lemma easily proved by structural induction on terms

$$\forall \phi, N, M, x \;.\; \phi M[\phi N/x] = \phi(M[N/x])$$

to reason

$$\begin{array}{rcll} \phi(M[p/x]) & = & \{p \mapsto r\}(M[p/x]) & (r \notin M[p/x]) \\ & = & (\{p \mapsto r\}M)[\{p \mapsto r\}p/x]) & (\text{lemma above}) \\ & = & M[r/x] & (p \notin M) \end{array}$$

as required. □

In the discussion before this proof I emphasised that lemma 2 is about extensional equivalence; we don't expect intensional properties of derivations, such as height, or occurrence of a particular parameter, to be preserved. However this proof is better than the statement of the lemma in the sense that it only renames derivations, leaving their shape intact; every one of the infinitely many branches above a G-LDA created by this proof has the same shape.

2.2 Some other systems for $\lambda\!\!\rightarrow$

We are now working towards a Constructive Engine for $\lambda\!\!\rightarrow$, and this will be a system for typing concrete terms with named variables that is closed under alpha-conversion. First we present an optimization that suggests a new idea for handling global variables. Following this idea we will see a presentation of $\lambda\!\!\rightarrow$ not mentioning substitution, that is closed under alpha-conversion.

An optimization. I think this optimization first appears in early writing of Martin-Löf, and it is also used in the Constructive Engine. It is interesting for our present purposes because it distinguishes between the variables that have always been global, i.e. bound by the context part of the judgement, and those that have only "locally" become global during construction of a derivation.

The inductive definition of $\vdash$ is very inefficient in duplicating the test that Γ valid on each branch of a derivation. For example

$$\dfrac{\dfrac{\vdots}{x{:}\sigma\to\sigma, y{:}\sigma \text{ valid}}}{x{:}\sigma\to\sigma, y{:}\sigma \vdash x : \sigma\to\sigma} \qquad \dfrac{\dfrac{\vdots}{x{:}\sigma\to\sigma, y{:}\sigma \text{ valid}}}{x{:}\sigma\to\sigma, y{:}\sigma \vdash y : \sigma}$$
$$x{:}\sigma\to\sigma, y{:}\sigma \vdash x\ y : \sigma$$

We can optimize $\vdash$ by moving the test for a valid context outside the typing derivation, that is, test once and for all that the given context is in fact valid, and then whenever a derivation extends the context (using the LDA rule), check that the extension preserves validity.

$$\text{O-VAR}\quad \Gamma \vdash_o x : \sigma \qquad x{:}\sigma \in \Gamma$$

$$\text{O-LDA}\quad \dfrac{\Gamma, x{:}\sigma \vdash_o M : \tau}{\Gamma \vdash_o [x{:}\sigma]M : \sigma\to\tau} \qquad x \notin \mathsf{Dom}(\Gamma)$$

$$\text{O-APP}\quad \dfrac{\Gamma \vdash_o M : \sigma\to\tau \qquad \Gamma \vdash_o N : \sigma}{\Gamma \vdash_o M\ N : \tau}$$

Comparing with $\vdash$, notice that O-VAR does not check that Γ is valid, but O-LDA does maintain this property during derivations.

The sense in which $\vdash_o$ is correct is given by

Lemma 3 Correctness of $\vdash_o$ for $\vdash$.
$\Gamma \vdash M : \sigma \quad \Leftrightarrow \quad (\Gamma \textsf{ valid and } \Gamma \vdash_o M : \sigma)$

Thus, in the tiny example above, we only need to check

$$\begin{array}{c}\vdots \\ x{:}\sigma \to \sigma, y{:}\sigma \textsf{ valid}\end{array} \quad \text{and} \quad \frac{x{:}\sigma \to \sigma, y{:}\sigma \vdash_o x : \sigma \to \sigma \qquad x{:}\sigma \to \sigma, y{:}\sigma \vdash_o y : \sigma}{x{:}\sigma \to \sigma, y{:}\sigma \vdash_o x\ y : \sigma}$$

Both directions of lemma 3 are easily proved by structural induction. However, there is a similar optimization, with a similar correctness lemma, for Pure Type Systems (see [vBJMP94, Pol94]). In that case, where correctness of a context and the typing judgement are mutually inductive, direction $\Rightarrow$ is trivial but direction $\Leftarrow$ is not. It is understandable that the latter is difficult, since it says there is a terminating algorithm for putting back all the redundant information removed from a $\vdash$-derivation[5].

The root of the problem We can view the systems presented so far as leaving unspecified how a context is searched for the type of a variable. This leaves open the possibility for various implementations, such as linear search, or hash-coding. The price we pay is the requirement for only one binding occurrence of a variable in a valid context. This is actually the root of our problem about closure under alpha-conversion, since we don't restrict to only one binding instance for a variable in a term. Informally our idea is to replace the rule O-VAR, which does not specify how to search Γ for an assignment to x, by

$$\Gamma \vdash x : \textsf{assoc}\, x\, \Gamma$$

which searches Γ linearly ($\textsf{assoc}\, x\, \Gamma$ returns the type of the *first* occurrence of x in Γ, viewing Γ as a list that conses on the right). More precisely, we replace O-VAR with two rules that search Γ from right to left.

$$\begin{array}{ll}
\textsc{I-START} & \Gamma, x{:}\sigma \vdash_i x : \sigma \\
\textsc{I-WEAK} & \dfrac{\Gamma \vdash_i x : \sigma}{\Gamma, y{:}\tau \vdash_i x : \sigma} \qquad x \neq y \\
\textsc{I-LDA} & \dfrac{\Gamma, x{:}\sigma \vdash_i M : \tau}{\Gamma \vdash_i [x{:}\sigma]M : \sigma \to \tau} \qquad x \notin \textsf{Dom}\,(\Gamma) \\
\textsc{I-APP} & \dfrac{\Gamma \vdash_i M : \sigma \to \tau \qquad \Gamma \vdash_i N : \sigma}{\Gamma \vdash_i M\ N : \tau}
\end{array}$$

$\vdash_i$ has fewer judgements than $\vdash_o$, for example $x{:}\sigma, x{:}\tau \vdash_o x : \sigma$ but $x{:}\sigma, x{:}\tau \not\vdash_i x : \sigma$. However only judgements of $\vdash_o$ that are incorrect for $\vdash$ are excluded: if Γ is valid, the order of search doesn't matter.

Lemma 4 Correctness of $\vdash_i$ for $\vdash$. *If Γ valid then*
$\Gamma \vdash_o M : \sigma \quad \Leftrightarrow \quad \Gamma \vdash_i M : \sigma$

[5] I owe this observation to Stefano Berardi.

Closure under alpha-conversion. The system $\vdash_i$ seems strange: why would we be more operational in presenting an abstract relation than required? The payoff is that now the side condition $x \notin \mathsf{Dom}(\Gamma)$ of rule I-LDA can also be dropped, giving the system of "liberal terms":

$$\text{LT-START} \qquad \Gamma, x{:}\sigma \vdash_{lt} x : \sigma$$

$$\text{LT-WEAK} \qquad \frac{\Gamma \vdash_{lt} x : \sigma}{\Gamma, y{:}\tau \vdash_{lt} x : \sigma} \qquad x \neq y$$

$$\text{LT-LDA} \qquad \frac{\Gamma, x{:}\sigma \vdash_{lt} M : \tau}{\Gamma \vdash_{lt} [x{:}\sigma]M : \sigma \to \tau}$$

$$\text{LT-APP} \qquad \frac{\Gamma \vdash_{lt} M : \sigma \to \tau \qquad \Gamma \vdash_{lt} N : \sigma}{\Gamma \vdash_{lt} M\, N : \tau}$$

We can consider two new relations now, $\vdash_{lt}$ and (Γ valid and $\vdash_{lt}$).

Notation 5. For relation $\vdash_x$, write $\Gamma \vdash^l_x M : \sigma$ for (Γ valid and $\Gamma \vdash_x M : \sigma$), the *local* version of $\vdash_x$

Clearly

$$\vdash \quad = \quad \vdash^l_i \quad \subset \quad \vdash^l_{lt} \quad \subset \quad \vdash_{lt}$$

where the containments are proper, as suggested by the following examples

$$y{:}\varsigma, x{:}\tau \vdash ([z{:}\tau][w{:}\xi]w)\, x : \xi \to \xi \tag{2}$$

$$y{:}\varsigma, x{:}\tau \vdash^l_{lt} ([x{:}\tau][x{:}\xi]x)\, x : \xi \to \xi \tag{3}$$

$$x{:}\varsigma, x{:}\tau \vdash_{lt} ([x{:}\tau][x{:}\xi]x)\, x : \xi \to \xi \tag{4}$$

In $\vdash$ a variable may not be bound twice on one branch, so judgement (3) is not $\vdash$-derivable. In $\vdash^l_{lt}$, a context must be valid, so (4) is not $\vdash^l_{lt}$-derivable, but there is no other restriction on variable re-use. In $\vdash_{lt}$ there is no restriction on variable re-use at all.

Notice that $\Gamma \vdash_{lt} M : \sigma$ iff $\vdash_{lt} \Gamma M : \Gamma\sigma$ (where if $\Gamma = x{:}\varsigma, y{:}\tau, \ldots$, then $\Gamma M = [x{:}\varsigma][y{:}\tau]\ldots M$ and $\Gamma\sigma = \varsigma \to \tau \to \ldots \sigma$), while $\vdash^l_{lt}$ doesn't have this property, which is why $\vdash^l_{lt}$ is called the local system of liberal terms.

With $\vdash_{lt}$ and $\vdash^l_{lt}$ we have systems for typing $\lambda\!\to$ which are closed under alpha-conversion and require no notion of substitution. (But of course we are also interested in reduction and conversion on the typed terms, and these require substitution.)

A criticism of $\vdash_{lt}$. As I suggested above, the non-operational abstraction "$x \in \Gamma$" that requires x is bound at most once in a valid context, is not well matched to our goal of presenting the typing relation, for the informal notion of term allows the same variable name to be bound more than once, and implicitly contains the idea of "linearly" searching from a variable instance through enclosing scopes to find the one binding that variable instance.

In fact, presentations of type systems in the style of our presentation of $\vdash$, where validity of a context means any variable name is bound at most once, are very common in the literature (for example [Bar92, Luo90, HHP92]). Why do type theory designers not use presentations in the style of $\vdash_{lt}$? The problem is that $\vdash_{lt}$ has bad properties of weakening. If $\Gamma \vdash M : \sigma$ and Γ' contains all the bindings of Γ, and is also valid, then $\Gamma' \vdash M : \sigma$ but $\vdash_{lt}$ doesn't have this property. This is a logical property which shows that global bindings should not be treated the same as local bindings. Both $\vdash$ and $\vdash_{lt}$ treat local and global bindings uniformly: $\vdash$ is unsatisfactory because it is too restrictive with local bindings, so is not closed under alpha-conversion; $\vdash_{lt}$ is unsatisfactory because it is too liberal with global bindings, so is not closed under weakening. Is $\vdash^{l}_{lt}$ just right?

2.3 $\lambda\!\!\rightarrow$ with nameless variables

A well-known technique to avoid questions of variable names is the use of de Bruijn nameless variables.

Pure nameless terms. Here is a presentation of $\lambda\!\!\rightarrow$ for pure nameless (de Bruijn) terms.

$$\textsc{db-start} \qquad \Gamma, \sigma \vdash_{db} 0 : \sigma$$

$$\textsc{db-weak} \qquad \frac{\Gamma \vdash_{db} n : \sigma}{\Gamma, \tau \vdash_{db} n+1 : \sigma}$$

$$\textsc{db-lda} \qquad \frac{\Gamma, \sigma \vdash_{db} M : \tau}{\Gamma \vdash_{db} [\sigma]M : \sigma \rightarrow \tau}$$

$$\textsc{db-app} \qquad \frac{\Gamma \vdash_{db} M : \sigma \rightarrow \tau \qquad \Gamma \vdash_{db} N : \sigma}{\Gamma \vdash_{db} M\ N : \tau}$$

Notice that there are no real choices to be made: there are no restrictions on the context, and, since there is no ambiguity possible, how we search the context is immaterial.

Locally nameless terms. Now consider terms whose local binding is by de Bruijn indexes, but whose global binding is by named variables. As before, x, y range over a class of variables that will be used for global, or free, variables. As usual, we define two operations of "substitution" (see [Hue89])

$M[N/k]$ replaces the k^{th} *free* index with appropriately lifted instances of N, and lowers all free indexes higher than k since there is no longer a "hole" at k.

$M[k/x]$ replaces name x with the k^{th} *free* index, lifting indexes greater or equal to k to make room for a new free index.

Here is a system for $\lambda\!\!\rightarrow$ typing of locally nameless terms.

$$\text{LN-VAR}\quad \frac{\Gamma\ \mathsf{valid}}{\Gamma \vdash_{ln} x : \sigma} \qquad x{:}\sigma \in \Gamma$$

$$\text{LN-LDA}\quad \frac{\Gamma, x{:}\sigma \vdash_{ln} M[x/0] : \tau}{\Gamma \vdash_{ln} [\sigma]M : \sigma \rightarrow \tau} \qquad x \notin M$$

$$\text{LN-APP}\quad \frac{\Gamma \vdash_{ln} M : \sigma \rightarrow \tau \qquad \Gamma \vdash_{ln} N : \sigma}{\Gamma \vdash_{ln} M\ N : \tau}$$

This system is very similar in spirit to $\vdash_s$ of section 2.1. Its handling of global names is identical to that of $\vdash_s$ (and $\vdash$), and its central feature, the analysis of how a local variable becomes global, is very reminiscent of $\vdash_s$. In particular, any sufficiently fresh variable can be used for x in the LN-LDA rule, and we can formalize this observation by proving, exactly as in lemma 2, that the judgements of $\vdash_{ln}$ are not changed by replacing LN-LDA with

$$\frac{\forall x \notin \mathsf{Dom}\ (\Gamma)\ .\ \Gamma, x{:}\sigma \vdash_{ln} M[x/0] : \tau}{\Gamma \vdash_{ln} [\sigma]M : \sigma \rightarrow \tau}$$

From this observation, we have a stronger induction principle for $\vdash_{ln}$ than structural induction, as discussed in relation to lemma 2.

Of course $\vdash_{ln}$ is closed under alpha-conversion, because alpha-conversion and identity are the same for locally nameless terms. The Constructive Engine uses $\vdash_{ln}$ as the "kernel" of a system for typing conventional named terms that inherits closure under alpha-conversion from $\vdash_{ln}$.

Locally closed terms. We call a term *locally closed* if it has no free index occurrences. It is easy to see that if $\Gamma \vdash_{ln} M : \sigma$, then M is locally closed, and all the type systems for locally nameless terms used below have this property.

2.4 A Constructive Engine for $\lambda\!\!\rightarrow$

Since $\vdash_{ln}$ treats global names just as $\vdash$ does, we may use the optimization and transformations of section 2.2 on $\vdash_{ln}$. Analogous to $\vdash_i$ we have $\vdash_{iln}$

$$\textsc{iln-start} \qquad \Gamma, x{:}\sigma \vdash_{iln} x : \sigma$$

$$\textsc{iln-weak} \qquad \frac{\Gamma \vdash_{iln} x : \sigma}{\Gamma, y{:}\tau \vdash_{iln} x : \sigma} \qquad x \neq y$$

$$\textsc{iln-lda} \qquad \frac{\Gamma, x{:}\sigma \vdash_{iln} M[x/0] : \tau}{\Gamma \vdash_{iln} [\sigma]M : \sigma \rightarrow \tau} \qquad x \notin M,\ x \notin \mathsf{Dom}(\Gamma)$$

$$\textsc{iln-app} \qquad \frac{\Gamma \vdash_{iln} M : \sigma \rightarrow \tau \qquad \Gamma \vdash_{iln} N : \sigma}{\Gamma \vdash_{iln} M\ N : \tau}$$

Similar to lemmas 3 and 4,

$$\Gamma \vdash_{ln} M : \sigma \quad \Leftrightarrow \quad (\Gamma \,\mathsf{valid} \text{ and } \Gamma \vdash_{iln} M : \sigma)$$

Again as in section 2.2 we define a system of "liberal terms" by replacing ILN-LDA with

$$\textsc{ltln-lda} \qquad \frac{\Gamma, x{:}\sigma \vdash_{ltln} M[x/0] : \tau}{\Gamma \vdash_{ltln} [\sigma]M : \sigma \rightarrow \tau} \qquad x \notin M$$

In section 2.2 the step from $\vdash_i$ to $\vdash_{lt}$ changed the derivable judgements; in fact $\vdash_{lt}$ is closed under alpha-conversion, while $\vdash_i$ is not. In the present case $\vdash_{iln}$ is already closed under alpha-conversion, so the step to $\vdash_{ltln}$ does not change the derivable judgements. The main point is that x occurs in the conclusion of I-LDA but not in the conclusion of ILN-LDA.

Lemma 6 Correctness of $\vdash_{ltln}$***.***
$\Gamma \vdash_{ln} M : \sigma \quad \Leftrightarrow \quad (\Gamma$ *valid and* $\Gamma \vdash_{ltln} M : \sigma)$

Proof. It suffices to show $\Gamma \vdash_{iln} M : \sigma \ \Leftrightarrow\ \Gamma \vdash_{ltln} M : \sigma$. Direction $\Rightarrow$ is trivial. Prove direction $\Leftarrow$ by induction on a derivation of $\Gamma \vdash_{ltln} M : \sigma$ using the strong induction principle derived from the fact that LTLN-LDA can be replaced by

$$\frac{\forall x \notin \mathsf{Dom}(\Gamma)\ .\ \Gamma, x{:}\sigma \vdash_{ltln} M[x/0] : \tau}{\Gamma \vdash_{ltln} [\sigma]M : \sigma \rightarrow \tau}$$

without changing the derivable judgements, as in the proof of lemma 2. □

The Constructive Engine. We are almost ready to present the Constructive Engine for $\lambda\!\!\rightarrow$. It is (a system of rules for) an inductive relation of the shape $\Gamma \vdash M \Rightarrow \overline{M} : \sigma$. Γ and M are concrete objects with named variables, which we think of as inputs to the engine. $\overline{M}$ and σ are the outputs, respectively the translation of M into locally nameless form, and the Γ-type of $\overline{M}$ in $\vdash_{ltln}$. For example, the rule for application terms is

$$\textsc{ce-app} \qquad \frac{\Gamma \vdash M \Rightarrow \overline{M} : \sigma \to \tau \qquad \Gamma \vdash N \Rightarrow \overline{N} : \sigma}{\Gamma \vdash M\,N \Rightarrow \overline{M}\,\overline{N} : \tau}$$

This is read "to translate and compute a type for the named term $M\,N$ in context Γ (i.e. to evaluate the conclusion of the rule given its inputs), translate and compute types for M and N (i.e. evaluate the premisses of the rule, whose inputs are computed from the given inputs to the conclusion), and return a result computed from the results of the premisses". We have called such systems *translation systems* [Pol90].

There is one difficulty remaining, with the rule for lambda terms. Following the CE-APP example, it should be

$$\frac{\Gamma, x{:}\sigma \vdash M \Rightarrow \overline{M}[x/0] : \tau}{\Gamma \vdash [x{:}\sigma]M \Rightarrow [\sigma]\overline{M} : \sigma \to \tau} \qquad x \notin \overline{M}$$

but it is not clear how to compute $[\sigma]\overline{M}$ from $\overline{M}[x/0]$. Reading LTLN-LDA algorithmically the term $[\sigma]M$ is the input: to compute its type, strip off the lambda, put a variable x in the hole thus created, (i.e. $M[x/0]$) and compute a type for this in an extended context. In the translation system this is dualized: given the named term $[x{:}\sigma]M$, translate the named term M to locally nameless term $\overline{M}$, and then somehow construct a locally nameless version of $[x{:}\sigma]M$. To fix this problem, we consider one more system, the same as $\vdash_{ltln}$ except that LTLN-LDA is replaced by

$$\textsc{pce-lda} \qquad \frac{\Gamma, x{:}\sigma \vdash_{pce} N : \tau}{\Gamma \vdash_{pce} [\sigma](N[0/x]) : \sigma \to \tau}$$

(PCE is for *pre-constructive-engine*) and claim:

Lemma 7 Correctness of $\vdash_{pce}$***.***
$\Gamma \vdash_{ln} M : \sigma \quad \Leftrightarrow \quad (\Gamma$ *valid and* $\Gamma \vdash_{pce} M : \sigma)$

Proof. It suffices to show $\Gamma \vdash_{ltln} M : \sigma \;\Leftrightarrow\; \Gamma \vdash_{pce} M : \sigma$. First we have the equations

$$M[0/x][x/0] = M \qquad \text{if } M \text{ is locally closed} \tag{5}$$
$$M[x/0][0/x] = M \qquad \text{if } x \notin M \tag{6}$$

For direction $\Leftarrow$, if a PCE-derivation ends with

$$\textsc{pce-lda} \qquad \frac{\Gamma, x{:}\sigma \vdash_{pce} N : \tau}{\Gamma \vdash_{pce} [\sigma](N[0/x]) : \sigma \to \tau}$$

(hence N is locally closed) apply LTLN-LDA with $M = N[0/x]$, using equation (5) and the fact that $x \notin N[0/x]$ no matter what N is.

Conversely, assume a LTLN-derivation ends with

$$\textsc{ltln-lda} \qquad \frac{\Gamma, x{:}\sigma \vdash_{ltln} M[x/0] : \tau}{\Gamma \vdash_{ltln} [\sigma]M : \sigma \to \tau} \qquad x \notin M$$

Since $x \notin M$, using equation (6), apply PCE-LDA with $N = M[x/0]$. □

Now we can give the Constructive Engine for $\lambda\!\to$:

$$\textsc{ce-start} \qquad \Gamma, x{:}\sigma \vdash x \Rightarrow \overline{x} : \sigma$$

$$\textsc{ce-weak} \qquad \frac{\Gamma \vdash x \Rightarrow \overline{x} : \sigma}{\Gamma, y{:}\tau \vdash x \Rightarrow \overline{x} : \sigma} \qquad x \neq y$$

$$\textsc{ce-lda} \qquad \frac{\Gamma, x{:}\sigma \vdash M \Rightarrow \overline{M} : \tau}{\Gamma \vdash [x{:}\sigma]M \Rightarrow [\sigma](\overline{M}[0/x]) : \sigma \to \tau}$$

$$\textsc{ce-app} \qquad \frac{\Gamma \vdash M \Rightarrow \overline{M} : \sigma \to \tau \qquad \Gamma \vdash N \Rightarrow \overline{N} : \sigma}{\Gamma \vdash M\,N \Rightarrow \overline{M}\,\overline{N} : \tau}$$

This system has a clear operational reading. Further, it has a soundness property: if $\Gamma \vdash N \Rightarrow \overline{N} : \sigma$ is derivable then $\Gamma \vdash_{pce} \overline{N} : \sigma$ is derivable, for given a derivation of the former, just erase the named terms to get a derivation of the latter. If we show that the translation from named terms to locally nameless terms is correct (I will not do so) the correctness of this engine is established.

3 Dependent Types

We will work with the familiar class of Pure Type Systems [Bar91, Bar92, GN91, Ber90, MP93, vBJMP94, vBJ93], which, without further ado, we present as the system of rules in Table 1.

A new difficulty arises with alpha-conversion in dependent types: the binding dependency of a term and its type may be different. For example, we expect to be able to derive

$$A{:}*, P{:}A \to * \vdash [x{:}A][x{:}Px]x : \{x{:}A\}\{y{:}Px\}Px \tag{7}$$

but not to derive

$$A{:}*, P{:}A \to * \vdash [x{:}A][x{:}Px]x : \{x{:}A\}\{x{:}Px\}Px$$

This example suggests that the rule

$$\textsc{lda} \qquad \frac{\Gamma[x{:}A] \vdash M : B \qquad \Gamma \vdash \{x{:}A\}B : s}{\Gamma \vdash [x{:}A]M : \{x{:}A\}B}$$

$$\textsc{ax}\qquad \bullet \vdash s_1 : s_2 \qquad\qquad \mathsf{Ax}(s_1{:}s_2)$$

$$\textsc{start}\qquad \frac{\Gamma \vdash A : s}{\Gamma[x{:}A] \vdash x : A} \qquad x \notin \mathsf{Dom}\,(\Gamma)$$

$$\textsc{weak}\qquad \frac{\Gamma \vdash \sigma : C \qquad \Gamma \vdash A : s}{\Gamma[x{:}A] \vdash \sigma : C} \qquad \sigma \text{ is a sort or a variable, } x \notin \mathsf{Dom}\,(\Gamma)$$

$$\textsc{pi}\qquad \frac{\Gamma \vdash A : s_1 \qquad \Gamma[x{:}A] \vdash B : s_2}{\Gamma \vdash \{x{:}A\}B : s_3} \qquad \mathsf{Rule}(s_1, s_2, s_3)$$

$$\textsc{lda}\qquad \frac{\Gamma[x{:}A] \vdash M : B \qquad \Gamma \vdash \{x{:}A\}B : s}{\Gamma \vdash [x{:}A]M : \{x{:}A\}B}$$

$$\textsc{app}\qquad \frac{\Gamma \vdash M : \{x{:}A\}B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B[N/x]}$$

$$\textsc{conv}\qquad \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s \qquad A \simeq B}{\Gamma \vdash M : B}$$

Table 1. The typing judgement of a PTS.

using the same bound variable for the term and its type is not exactly what we intend.

A formalization of PTS that distinguishes between parameters and variables, along the lines discussed in section 2.1, is described in [MP93]. We use the following LDA rule:

$$\textsc{lda}\qquad \frac{\Gamma, p{:}A \vdash M[p/x] : B[p/y] \qquad \Gamma \vdash \{y{:}A\}B : s}{\Gamma \vdash [x{:}A]M : \{y{:}A\}B}\ p \notin M,\ p \notin B$$

This system has closure under parallel reduction [Pol94]; writing $\twoheadrightarrow$ for parallel reduction we have

Lemma 8. *If $\Gamma \vdash M : A$, $\Gamma \twoheadrightarrow \Gamma'$, $M \twoheadrightarrow M'$ and $A \twoheadrightarrow A'$ then $\Gamma' \vdash M' : A'$*

Since alpha-conversion is contained in parallel reduction, we have as a corollary

Corollary 9. *If $\Gamma \vdash M : A$, $\Gamma \stackrel{\alpha}{=} \Gamma'$, $M \stackrel{\alpha}{=} M'$ and $A \stackrel{\alpha}{=} A'$ then $\Gamma' \vdash M' : A'$*

However this proof, instantiating closure under parallel reduction with the degenerate case of alpha-conversion, is not intensionally satisfactory as it uses the rule CONV only for alpha-conversion. A presentation of PTS using nameless terms will never use CONV for alpha-conversion, and I would like to know $\vdash$ is the structurally isomorphic to such a nameless presentation, not just that it has isomorphic judgements.

VC-SRT	$\Gamma \vdash_{vc} s_1 : s_2$	$\mathsf{Ax}(s_1{:}s_2)$
VC-VAR	$\Gamma \vdash_{vc} x : \mathsf{assoc}\ x\ \Gamma$	
VC-PI	$\dfrac{\Gamma \vdash_{vc} A : s_1 \qquad \Gamma, x{:}A \vdash_{vc} B : s_2}{\Gamma \vdash_{vc} \{x{:}A\}B : s_3}$	$\mathsf{Rule}(s_1, s_2, s_3)$ $x \notin \mathsf{Dom}\,(\Gamma)$
VC-LDA	$\dfrac{\Gamma, x{:}A \vdash_{vc} b : B \qquad \Gamma \vdash_{vc} \{x{:}A\}B : s}{\Gamma \vdash_{vc} [x{:}A]b : \{x{:}A\}B}$	$x \notin \mathsf{Dom}\,(\Gamma)$
VC-APP	$\dfrac{\Gamma \vdash_{vc} a : \{x{:}B\}A \qquad \Gamma \vdash_{vc} b : B}{\Gamma \vdash_{vc} a\,b : A[b/x]}$	
VC-CNV	$\dfrac{\Gamma \vdash_{vc} a : A \qquad \Gamma \vdash_{vc} B : s \qquad A \simeq B}{\Gamma \vdash_{vc} a : B}$	
NIL-VC	$\bullet \vdash_{vc}$	
CONS-VC	$\dfrac{\Gamma \vdash_{vc} \qquad \Gamma \vdash_{vc} A : s}{\Gamma, x{:}A \vdash_{vc}}$	$x \notin \mathsf{Dom}\,(\Gamma)$

Table 2. The system of valid contexts.

In section 2.2 we derived a system, $\vdash_{lt}$, for $\lambda\!\!\rightarrow$ without any variable renaming that was closed under alpha-conversion. I don't think we can do the same for PTS. If we follow the transformations of section 2.2, first optimizing to only check context validity once, then linearizing context search, we arrive at the system of Table 2. This system is correct, in the sense

$$\Gamma \vdash M : A \Leftrightarrow (\Gamma \vdash_{vc} \text{ and } \Gamma \vdash_{vc} M : A)$$

If we now try to drop the side conditions $x \notin \mathsf{Dom}\,(\Gamma)$ from VC-PI and VC-LDA, as in section 2.2, (call this system $\vdash_{bad}$) we find the following incorrect derivation

$$\dfrac{\dfrac{}{\Gamma, x{:}A, x{:}Px \vdash_{bad} x : Px}}{\dfrac{\Gamma, x{:}A \vdash_{bad} [x{:}Px]x : \{x{:}Px\}Px}{\Gamma \vdash_{bad} [x{:}A][x{:}Px]x : \{x{:}A\}\{x{:}Px\}Px}}$$

3.1 The Constructive Engine

I remind you that this paper is not addressing the issue of making PTS syntax directed; the Constructive Engine we will derive now is not yet a program for typechecking PTS (in particular, the non-syntax-directed conversion rule remains), but does explain the interaction between named and nameless variables of an operational Constructive Engine.

Table 3 is a correct presentation of PTS using locally nameless terms, corresponding to $\vdash_{iln}$ of section 2.4. Now, as in section 2.4, we may drop the side

ILN-SRT	$\Gamma \vdash_{iln} s_1 : s_2$	$\mathsf{Ax}(s_1{:}s_2)$
ILN-VAR	$\Gamma \vdash_{iln} x : \mathsf{assoc}\ x\ A$	
ILN-PI	$\dfrac{\Gamma \vdash_{iln} A : s_1 \qquad \Gamma, x{:}A \vdash_{iln} B[x/0] : s_2}{\Gamma \vdash_{iln} \{A\}B : s_3}$	$\mathsf{Rule}(s_1, s_2, s_3)$ $x \notin B,\ x \notin \mathsf{Dom}\,(\Gamma)$
ILN-LDA	$\dfrac{\Gamma, x{:}A \vdash_{iln} b[x/0] : B[x/0] \qquad \Gamma \vdash_{iln} \{A\}B : s}{\Gamma \vdash_{iln} [A]b : \{A\}B}$	$x \notin b,\ x \notin B$ $x \notin \mathsf{Dom}\,(\Gamma)$
ILN-APP	$\dfrac{\Gamma \vdash_{iln} a : \{B\}A \qquad \Gamma \vdash_{iln} b : B}{\Gamma \vdash_{iln} a\,b : A[b/0]}$	
ILN-CNV	$\dfrac{\Gamma \vdash_{iln} a : A \qquad \Gamma \vdash_{iln} B : s \qquad A \simeq B}{\Gamma \vdash_{iln} a : B}$	
ILN-NIL	$\bullet \vdash_{iln}$	
ILN-CONS	$\dfrac{\Gamma \vdash_{iln} \qquad \Gamma \vdash_{iln} A : s}{\Gamma, x{:}A \vdash_{iln}}$	$x \notin \mathsf{Dom}\,(\Gamma)$

Table 3. The intermediate system of locally nameless terms.

condition $x \notin \mathsf{Dom}\,(\Gamma)$ from rules ILN-PI and ILN-LDA (getting the system $\vdash_{ltln}$), just as in lemma 6. Here we are using the locally nameless representation in an essential way for dependent types!

Continuing as in section 2.4, we use the argument of lemma 7 to replace LTLN-PI and LTLN-LDA by

PCE-PI	$\dfrac{\Gamma \vdash_{pce} A : s_1 \qquad \Gamma, x{:}A \vdash_{pce} B : s_2}{\Gamma \vdash_{pce} \{A\}(B[0/x]) : s_3}$	$\mathsf{Rule}(s_1, s_2, s_3)$
PCE-LDA	$\dfrac{\Gamma, x{:}A \vdash_{pce} b : B \qquad \Gamma \vdash_{pce} \{A\}B : s}{\Gamma \vdash_{pce} [A](b[0/x]) : \{A\}(B[0/x])}$	

giving a system, $\vdash_{pce}$, that can be made into a Constructive Engine as in section 2.4. This Constructive Engine (Table 4) is closed under alpha-conversion.

CE-SRT $\quad \Gamma \vdash s_1 \Rightarrow s_1 : s_2 \qquad \mathsf{Ax}(s_1{:}s_2)$

CE-VAR $\quad \Gamma \vdash x \Rightarrow x : \mathsf{assoc}\, x\, A$

CE-PI $\quad \dfrac{\Gamma \vdash A \Rightarrow \overline{A} : s_1 \qquad \Gamma, x{:}A \vdash B \Rightarrow \overline{B} : s_2}{\Gamma \vdash \{x{:}A\}B \Rightarrow \{\overline{A}\}(\overline{B}[0/x]) : s_3} \qquad \mathsf{Rule}(s_1, s_2, s_3)$

CE-LDA $\quad \dfrac{\Gamma, x{:}A \vdash b \Rightarrow \overline{b} : \overline{B} \qquad \Gamma \vdash \{x{:}A\}B \Rightarrow \{\overline{A}\}\overline{B} : s}{\Gamma \vdash [x{:}A]b \Rightarrow [\overline{A}](\overline{b}[0/x]) : \{\overline{A}\}(\overline{B}[0/x])}$

CE-APP $\quad \dfrac{\Gamma \vdash a \Rightarrow \overline{a} : \{\overline{B}\}\overline{A} \qquad \Gamma \vdash b \Rightarrow \overline{b} : \overline{B}}{\Gamma \vdash a\, b \Rightarrow \overline{a}\,\overline{b} : \overline{A}[\overline{b}/0]}$

CE-CNV $\quad \dfrac{\Gamma \vdash a \Rightarrow \overline{a} : \overline{A} \qquad \Gamma \vdash B \Rightarrow \overline{B} : s \qquad \overline{A} \simeq \overline{B}}{\Gamma \vdash a \Rightarrow \overline{a} : \overline{B}}$

CE-NIL $\quad \bullet \vdash_{iln}$

CE-CONS $\quad \dfrac{\Gamma \vdash_{iln} \qquad \Gamma \vdash A \Rightarrow \overline{A} : s}{\Gamma, x{:}A \vdash_{iln}} \qquad x \notin \mathsf{Dom}\,(\Gamma)$

Table 4. The Constructive Engine for PTS.

References

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Levy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, October 1991.

[Alt93] Thorsten Altenkirch. A formalization of the strong normalization proof for System F in LEGO. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications, TLCA'93*. Springer-Verlag, LNCS 664, March 1993.

[Bar91] Henk Barendregt. Introduction to Generalised Type Sytems. *J. Functional Programming*, 1(2):125–154, April 1991.

[Bar92] Henk Barendregt. Lambda calculi with types. In Abramsky, Gabbai, and Maibaum, editors, *Handbook of Logic in Computer Science*, volume II. Oxford University Press, 1992.

[Ber90] Stefano Berardi. *Type Dependence and Constructive Mathematics*. PhD thesis, Dipartimento di Informatica, Torino, Italy, 1990.

[Coq91] Thierry Coquand. An algorithm for testing conversion in type theory. In G. Huet and G. D. Plotkin, editors, *Logical Frameworks*. Cambridge University Press, 1991.

[dB72] Nicolas G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indag. Math.*, 34(5), 1972.

[dB85] Nicolas G. de Bruijn. Generalizing automath by means of a lambda-typed

lambda calculus. In *Proceedings of the Maryland 1984-1985 Special Year in Mathematical Logic and Theoretical Computer Science*, 1985.

[DFH+93] Dowek, Felty, Herbelin, Huet, Murthy, Parent, Paulin-Mohring, and Werner. The Coq proof assistant user's guide, version 5.8. Technical report, INRIA-Rocquencourt, February 1993.

[GN91] Herman Geuvers and Mark-Jan Nederhof. A modular proof of strong normalization for the calculus of constructions. *Journal of Functional Programming*, 1(2):155–189, April 1991.

[Gor93] Andrew Gordon. A mechanism of name-carrying syntax up to alpha-conversion. In *Proceedings of the 1993 HOL User's Meeting, Vancouver*. Springer-Verlag, 1993. LNCS.

[HHP92] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1992. Preliminary version in LICS'87.

[Hue89] Gérard Huet. The constructive engine. In R. Narasimhan, editor, *A Perspective in Theoretical Computer Science*. World Scientific Publishing, 1989. Commemorative Volume for Gift Siromoney.

[Hue94] Gérard Huet. Residual theory in λ-calculus: A formal development. *Journal of Functional Programming*, 4(3):371–394, July 1994.

[LP92] Zhaohui Luo and Robert Pollack. LEGO proof development system: User's manual. Technical Report ECS-LFCS-92-211, LFCS, Computer Science Dept., University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, May 1992. Updated version. See `http://www.dcs.ed.ac.uk/packages/lego/`

[Luo90] Zhaohui Luo. *An Extended Calculus of Constructions*. PhD thesis, Department of Computer Science, University of Edinburgh, June 1990.

[MP93] James McKinna and Robert Pollack. Pure Type Sytems formalized. In M.Bezem and J.F.Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications, TLCA'93, Utrecht*, pages 289–305. Springer-Verlag, LNCS 664, March 1993.

[Pol90] Robert Pollack. Implicit syntax. Informal Proceedings of First Workshop on Logical Frameworks, Antibes, May 1990.

[Pol92] R. Pollack. Typechecking in Pure Type Sytems. In *Informal Proceedings of the 1992 Workshop on Types for Proofs and Programs, Båstad, Sweden*, pages 271–288, June 1992. Available by ftp.

[Pol94] Robert Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1994. Available by anonymous ftp from `ftp.cs.chalmers.se` in directory `pub/users/pollack`.

[Tas93] A. Tasistro. Formulation of Martin-Löf's theory of types with explicit substitutions. Master's thesis, Chalmers Tekniska Högskola and Göteborgs Universitet, May 1993.

[vBJ93] L.S. van Benthem Jutting. Typing in Pure Type Sytems. *Information and Computation*, 105(1):30–41, July 1993.

[vBJMP94] L.S. van Benthem Jutting, James McKinna, and Robert Pollack. Checking algorithms for Pure Type Systems. In Henk Barendregt and Tobias Nipkow, editors, *Types for Proofs and Programs: International Workshop TYPES'93, Nijmegen, May 1993, Selected Papers*, volume 806 of *LNCS*, pages 19–61. Springer-Verlag, 1994.

This article was processed using the LaTeX macro package with LLNCS style