Expansion Postponement for Normalising Pure Type Systems

Erik Poll

Computing Laboratory, University of Kent, Canterbury, England

Abstract

Expansion Postponement is a tantalisingly simple conjecture about Pure Type Systems, which has so far resisted all attempts to prove it for any interesting class of systems. We prove the property for all normalising Pure Type Systems, and discuss the connection with typechecking.

Pure Type Systems (PTSs) (Barendregt 1991, 1992) provide a general framework for describing a large class of type theories (or typed lambda calculi). Typically PTSs are expressive type theories, where we not only have β-reduction on terms, but also on types. To derive a type for term it may then be necessary to perform some β-conversions of types.

Expansion Postponement (EP) is the conjecture that to typecheck terms we only ever have to β-reduce types, and never have to β-expand them. This is clearly a desirable property: β-expansion a very non-deterministic relation, and there is no sensible strategy for β-expanding terms. As we will explain later, EP is a necessary condition for correctness of the natural typechecking algorithm for PTSs proposed by R. Pollack (1992) (but, unfortunately, not a sufficient one).

For a more precise definition of EP, we have to consider the PTS type inference rule for converting types:

$$\textsf{conversion} \quad \frac{\Gamma \vdash b : B \qquad \Gamma \vdash B' : s \qquad B =_\beta B'}{\Gamma \vdash b : B'}$$

It allows any type B to be replaced by a β-convertible one B', provided this new type B' is a well-formed type-expression, which is guaranteed by the premise $\Gamma \vdash B' : s$.

This rule for converting types can be split into two rules, one for *reducing* and one for *expanding* types. And because reducing a well-formed expression always produces a well-formed expression, the premise $\Gamma \vdash B' : s$ can safely be dropped in the former rule. This results in the following rules:

$$\textsf{reduction} \quad \frac{\Gamma \vdash b : B \qquad B \twoheadrightarrow_\beta B'}{\Gamma \vdash b : B'}$$

$$\textsf{expansion} \quad \frac{\Gamma \vdash b : B \qquad \Gamma \vdash B' : s \qquad B \twoheadleftarrow_\beta B'}{\Gamma \vdash b : B'}$$

It is not difficult to show that replacing conversion with the two rules above does not change the typing relation (Lemma 16 in (Benthem Jutting *et al.*, 1993)).

But is expansion really needed ? When we look at some type derivations, it seems that we can always get by with just reduction. The most obvious place where conversion is needed is to check if the types of a function and its argument match, and for this just reduction would be sufficient (since β-reduction is Church-Rosser).

Expansion Postponement is the conjecture that any typing judgement $\Gamma \vdash a : A$ can be derived by first deriving $\Gamma \vdash a : A'$ for some A' without using expansion, and then possibly using expansion just once to β-expand A' to A. In other words, EP says that the use of expansion can always be *postponed* to the end of a type derivation. It would follow from EP that if we are interested in finding just any type for a given term we can forget about the expansion rule altogether.

Definition of PTSs and Expansion Postponement

We quickly recall the definition of Pure Type Systems (PTSs). For more information on PTSs see for instance (Barendregt 1991, 1992).

Definition 1
A *Pure Type System (PTS)* is a triple $(\mathsf{S}, \mathsf{A}, \mathsf{R})$, with S a set of symbols called the *sorts*, $\mathsf{A} \subseteq \mathsf{S} \times \mathsf{S}$ a set of *axioms*, and $\mathsf{R} \subseteq \mathsf{S} \times \mathsf{S} \times \mathsf{S}$, a set of *rules*.

The *terms* of a PTS are generated by

$$\mathsf{T} \quad ::= \quad \mathsf{Var} \mid \ \mathsf{S} \mid \ (\mathsf{T}\mathsf{T}) \mid \ (\lambda \mathsf{Var}{:}\mathsf{T}.\ \mathsf{T}) \mid \ (\Pi \mathsf{Var}{:}\mathsf{T}.\ \mathsf{T}),$$

where Var is a set of variables.

We use the following conventions: $s, s_1, \ldots$ range over sorts; x, y range over variables; $a, A, b, B, \ldots$ range over terms. Terms equal up to the names of bound variables are identified, and $\equiv$ denotes syntactic equality. We write $b[x := a]$ for the capture-free substitution of a for x in b, $\rightarrow_\beta$ for one-step β-reduction, $\twoheadrightarrow_\beta$ for its reflexive and transitive closure, and $=_\beta$ for β-equality.

The *typing relation* $\vdash$ of a PTS is the smallest relation closed under the rules:

axiom	$\epsilon \vdash s : s'$	$(s : s') \in \mathsf{A}$
variable	$\dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$	
weakening	$\dfrac{\Gamma \vdash b : B \quad \Gamma \vdash A : s}{\Gamma, x : A \vdash b : B}$	
formation	$\dfrac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x{:}A.\ B) : s_3}$	$(s_1, s_2, s_3) \in \mathsf{R}$
abstraction	$\dfrac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x{:}A.\ B) : s}{\Gamma \vdash (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B)}$	
application	$\dfrac{\Gamma \vdash b : (\Pi x{:}A.\ B) \quad \Gamma \vdash a : A}{\Gamma \vdash ba : B[x := a]}$	
reduction	$\dfrac{\Gamma \vdash b : B \quad B \twoheadrightarrow_\beta B'}{\Gamma \vdash b : B'}$	
expansion	$\dfrac{\Gamma \vdash b : B \quad \Gamma \vdash B' : s \quad B \twoheadleftarrow_\beta B'}{\Gamma \vdash b : B'}$	

where we assume that no variable is declared twice in a context.

Two of the most important properties of PTSs are:

Theorem 2

- Subject Reduction (SR). If $\Gamma \vdash a : A$ and $a \twoheadrightarrow_\beta a'$, then $\Gamma \vdash a' : A$.
- Correctness of Types (CT). If $\Gamma \vdash a : A$, then $\Gamma \vdash A : s$ or $A \in \mathsf{S}$.

We now consider the system with only *reduction* of types:

Definition 3
The typing relation $\vdash_r$ is the smallest relation closed under all the rules above *except* expansion.

It is obvious that all $\vdash_r$-judgements are also $\vdash$-judgements:

Theorem 4 ($\vdash_r \subseteq \vdash$)
If $\Gamma \vdash_r a : A$ then $\Gamma \vdash a : A$.

Expansion postponement is essentially the reverse implication. However, $\vdash \subseteq \vdash_r$ will not always be true. For example, it may be possible that $\Gamma, x : A \vdash x : A'$ for some $A' \twoheadrightarrow_\beta A$, whereas it is impossible that $\Gamma, x : A \vdash_r x : A'$ for some $A' \twoheadrightarrow_\beta A$, because in $\vdash_r$ types cannot be expanded as in $\vdash$. So the best we can hope for is $\vdash \subseteq \vdash_r$ "modulo" β-reduction of types:

Open Problem 5 (Expansion Postponement (EP))
If $\Gamma \vdash a : A$ then $\Gamma \vdash_r a : A'$ for some $A \twoheadrightarrow_\beta A'$?

We will abuse notation when writing inclusions between relations, and for instance (somewhat incorrectly) write $\vdash \subseteq \vdash_r$ for EP.

The rest of this section illustrates some of the problems that arise when we try to prove EP. None of this is relevant to the rest of the paper, so the reader who already knows or believes that EP is not easy to prove may choose to skip it.

The obvious way to prove EP – induction on the derivation – fails in the case that the last step is abstraction:

- Suppose the last step in the derivation is:

$$\textsf{abstraction}\quad \frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash (\Pi x{:}A.\ B) : s}{\Gamma \vdash (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B)}$$

  By the induction hypothesis $\Gamma, x : A \vdash_r b : B'$(i) for some $B \twoheadrightarrow_\beta B'$ and $\Gamma \vdash_r (\Pi x{:}A.\ B) : S$ for some $s \twoheadrightarrow_\beta S$. And since s cannot be reduced, this means that $\Gamma \vdash_r (\Pi x{:}A.\ B) : s$ (ii). Now to derive $\Gamma \vdash_r (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B')$ from (i) using the abstraction rule, we would need $\Gamma \vdash_r (\Pi x{:}A.\ B') : s$ as a second premise. This is frustratingly close to (ii), but not exactly the same !

  The root of the problem here is that in the abstraction rule $(\Pi x : A.\ B)$ occurs to the *right* of ':' in the conclusion but to the *left* of ':' in the premise.

The natural thing to try now is proving some useful properties of $\vdash_r$. In particular, SR would provide the missing link in the attempted proof above, as it would allow us to deduce $\Gamma \vdash_r (\Pi x{:}A.\ B') : s$ from (ii). But it is surprisingly difficult to prove *any* of the usual meta-theoretic properties for $\vdash_r$:

Open Problem 6

- Weak Subject Reduction (WSR) for $\vdash_r$.
  If $\Gamma \vdash_r a : A$ and $a \twoheadrightarrow_\beta a'$ then $\Gamma \vdash_r a' : A'$ for some $A \twoheadrightarrow_\beta A'$?
  (We do *not* have SR for $\vdash_r$; see (Pollack, 1994) for a counterexample).
- Correctness of Types (CT) for $\vdash_r$.
  If $\Gamma \vdash_r a : A$ then $\Gamma \vdash_r A : s$ or $A \in \mathsf{S}$?

It is easily shown that EP is equivalent with WSR for $\vdash_r$, and that, for the so-called functional PTSs, EP is also equivalent with CT for $\vdash_r$.

Expansion Postponement for Normalising PTSs

From now on we restrict ourselves to PTSs *with normalising types*, i.e. the PTSs for which for all $\Gamma \vdash a : A$ the type A has a normal form. This clearly subsumes all *normalising* PTSs, i.e. the PTSs for which for all $\Gamma \vdash a : A$ both a and A have normal forms.

The trick is that instead of $\vdash_r$ we consider an even more restricted system, $\vdash_{nf}$, and we prove EP ($\vdash \subseteq \vdash_r$) by proving the stronger property $\vdash \subseteq \vdash_{nf}$. The system $\vdash_{nf}$ gives types in normal form:

Definition 7
The typing relation $\vdash_{nf}$ is the smallest relation closed under the following rules:

axiom	$\epsilon \vdash_{nf} s : s'$	$(s : s') \in \mathsf{A}$
nf-variable	$\dfrac{\Gamma \vdash_{nf} A : s}{\Gamma, x : A \vdash_{nf} x : nf(A)}$	
weakening	$\dfrac{\Gamma \vdash_{nf} b : B \quad \Gamma \vdash_{nf} A : s}{\Gamma, x : A \vdash_{nf} b : B}$	
formation	$\dfrac{\Gamma \vdash_{nf} A : s_1 \quad \Gamma, x : A \vdash_{nf} B : s_2}{\Gamma \vdash_{nf} (\Pi x{:}A.\ B) : s_3}$	$(s_1, s_2, s_3) \in \mathsf{R}$
nf-abstraction	$\dfrac{\Gamma, x : A \vdash_{nf} b : B \quad \Gamma \vdash_{nf} (\Pi x{:}A.\ B) : s}{\Gamma \vdash_{nf} (\lambda x{:}A.\ b) : (\Pi x{:}nf(A).\ B)}$	
nf-application	$\dfrac{\Gamma \vdash_{nf} b : (\Pi x{:}A.\ B) \quad \Gamma \vdash_{nf} a : A}{\Gamma \vdash_{nf} ba : nf(B[x := a])}$	

Here $nf(A)$ denotes the β-normal form of A.

It is easy to see that all $\vdash_{nf}$-judgements are also $\vdash_r$-judgements:

Theorem 8 ($\vdash_{nf} \subseteq \vdash_r$)
If $\Gamma \vdash_{nf} a : A$ then $\Gamma \vdash_r a : A$.

Proof
Easy induction on $\Gamma \vdash_{nf} a : A$. In fact, it suffices to observe that all $\vdash_{nf}$-rules are derivable rules for $\vdash_r$. For instance, nf-abstraction can be derived by composing abstraction and reduction. □

So, by Theorems 4 and 8, $\vdash_{nf} \subseteq \vdash_r \subseteq \vdash$. (Note that it immediately follows from this that if $\vdash$ produces normalising types, then so do $\vdash_r$ and $\vdash_{nf}$.) The crucial difference between $\vdash_{nf}$ and $\vdash_r$ is that for $\vdash_{nf}$ we can prove SR. For this the following two lemmas are needed:

Lemma 9 (Generation lemma for $\vdash_{nf}$)
If $\Gamma \vdash_{nf} (\lambda x{:}A'.\ b) : (\Pi x{:}A.\ B)$ then $\Gamma, x : A' \vdash_{nf} b : B$.

Proof
Induction on the derivation of $\Gamma \vdash_{nf} (\lambda x{:}A'.\ b) : (\Pi x{:}A.\ B)$. The last step can only be nf-abstraction or weakening, so these are the only cases we have to consider. □

Lemma 10 (Substitution lemma for $\vdash_{nf}$)
For PTSs with normalising types: if $\Gamma, x : A, \Delta \vdash_{nf} b : B$ and $\Gamma \vdash_{nf} a : \mathit{nf}(A)$, then $\Gamma, \Delta[x := a] \vdash_{nf} b[x := a] : \mathit{nf}(B[x := a])$.

Proof
This can be proved as the substitution lemma for $\vdash$ (see for instance (Barendregt, 1992)), by induction on the derivation of $\Gamma, x : A, \Delta \vdash_{nf} b : B$. □

Theorem 11 (Subject Reduction (SR) for $\vdash_{nf}$)
For PTSs with normalising types: if $\Gamma \vdash_{nf} a : A$ and $a \twoheadrightarrow_\beta a'$ then $\Gamma \vdash_{nf} a' : A$.

Proof
This can be proved as SR for $\vdash$ (see for instance (Barendregt, 1992)). In fact, the proof for $\vdash_{nf}$ is a bit simpler. We simultaneously prove the following two properties by induction on the derivation:

(1) if $\Gamma \vdash_{nf} c : C$ and $\Gamma \to_\beta \Gamma'$ then $\Gamma' \vdash_{nf} c : C$.
(2) if $\Gamma \vdash_{nf} c : C$ and $c \to_\beta c'$ then $\Gamma \vdash_{nf} c' : C$.

All the cases are very boring, except the case that c is the redex that is contracted, which is where the substitution lemma and the generation lemma are needed:

Suppose the last step in the derivation of $\Gamma \vdash_{nf} c : C$ is

$$\mathsf{nf-application} \quad \frac{^{(\mathrm{i})}\Gamma \vdash_{nf} b : (\Pi x{:}A.\ B) \qquad ^{(\mathrm{ii})}\Gamma \vdash_{nf} a : A}{\Gamma \vdash_{nf} ba : \mathit{nf}(B[x := a])}$$

i.e. $c \equiv ba$ and $C \equiv \mathit{nf}(B[x := a])$.

(1) To prove: $\Gamma' \vdash_{nf} c : \mathit{nf}(B[x := a])$ for $\Gamma \to_\beta \Gamma'$.
If $\Gamma \to_\beta \Gamma'$, then by the IH $\Gamma' \vdash_{nf} b : (\Pi x{:}A.\ B)$ and $\Gamma' \vdash_{nf} a : A$, and by the $\vdash_{nf}$-application rule $\Gamma' \vdash_{nf} ba : \mathit{nf}(B[x := a])$.
(2) To prove: $\Gamma \vdash_{nf} c' : \mathit{nf}(B[x := a])$ for $ba \to_\beta c'$.
We distinguish two possibilities for the reduction $ba \to_\beta c'$:

— $ba \to_\beta c'$ is a reduction $ba \to_\beta b'a'$, with $a \to_\beta a'$ and $b \equiv b'$, or $b \to_\beta b'$ and $a \equiv a'$. Then by the IH $\Gamma \vdash_{nf} b' : (\Pi x{:}A.\ B)$ and $\Gamma \vdash_{nf} a' : A$, and by the $\vdash_{nf}$-application rule $\Gamma \vdash_{nf} b'a' : \mathit{nf}(B[x := a'])$.

— $ba \to_\beta c'$ is the reduction $(\lambda x{:}A'.\ b')a \to_\beta b'[x := a]$, i.e. $b \equiv (\lambda x{:}A'.\ b')$.
To prove: $\Gamma \vdash_{nf} b'[x := a] : nf(B[x := a])$.
Now (i) is $\Gamma \vdash_{nf} (\lambda x{:}A'.\ b') : (\Pi x{:}A.\ B)$, and so by the generation lemma $\Gamma, x : A' \vdash_{nf} b' : B$ (iii). By the substitution lemma it follows from (ii) and (iii) that $\Gamma \vdash_{nf} b'[x := a] : nf(B[x := a])$.

$\square$

Theorem 12 ($\vdash \subseteq \vdash_{nf}$)
For PTSs with normalising types: if $\Gamma \vdash c : C$ then $\Gamma \vdash_{nf} c : nf(C)$.

Proof
Induction on the derivation of $\Gamma \vdash c : C$. Almost all the cases are trivial, except abstraction, which is where SR for $\vdash_{nf}$ is needed:

Suppose the last rule in the derivation of $\Gamma \vdash c : C$ is

$$\frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash (\Pi x{:}A.\ B) : s}{\Gamma \vdash (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B)}\ \mathsf{abstraction}$$

i.e. $c \equiv (\lambda x{:}A.\ b)$ and $C \equiv (\Pi x{:}A.\ B)$.

To prove: $\Gamma \vdash_{nf} (\lambda x{:}A.\ b) : nf(\mathit{\Pi} x{:}A.\ B)$. By the induction hypothesis we have $\Gamma, x : A \vdash_{nf} b : nf(B)$ (i) and $\Gamma \vdash_{nf} (\Pi x{:}A.\ B) : s$ (ii). By SR for $\vdash_{nf}$ it follows from (ii) that $\Gamma \vdash_{nf} (\Pi x{:}A.\ nf(B)) : s$ (iii). Now by applying nf-abstraction to (i) and (iii) we are done, since obviously $nf(\mathit{\Pi} x{:}A.\ nf(B)) \equiv nf(\mathit{\Pi} x{:}A.\ B)$. $\square$

Now by Theorems 8 and 12 ($\vdash_{nf} \subseteq \vdash_r$ and $\vdash \subseteq \vdash_{nf}$) it immediately follows that $\vdash \subseteq \vdash_r$:

Corollary 13 (Expansion Postponement)
For PTSs with normalising types: if $\Gamma \vdash a : A$ then $\Gamma \vdash_r a : A'$ for some $A' \twoheadrightarrow_\beta A$.

Expansion Postponement and Typechecking

To conclude, we say a few words about the connection between Expansion Postponement and typechecking, and leave the reader with an interesting open problem.

By a typechecking algorithm we mean an algorithm that, given a term a and a context Γ, answers the question $\Gamma \vdash a : ?$, i.e. that returns a type A such that $\Gamma \vdash a : A$, or reports failure if no such A exists. Several generic typechecking algorithms for different classes of PTSs are known (see (Benthem Jutting *et al.*, 1993) and (Poll, 1993)).

However, correctness of the most natural typechecking algorithm, defined in Pollack (1992), remains an open problem. This algorithm tries to answer $\Gamma \vdash a : ?$ in the obvious way, by trying to construct a type derivation for a in context Γ guided by the shape a (and Γ), effectively trying to derive a type of a term from the types of its sub-terms.

Before considering its formal definition, we can already give an informal explanation of why EP is a necessary condition for correctness of this algorithm. It is not difficult to see that the algorithm will never use the expansion rule. To use expansion it would have to guess a β-expansion of a type, and there is no sensible strategy for doing this. But if the algorithm tries to answer $\Gamma \vdash a : ?$ without using $\mathsf{expansion}$, then it really tries to answer $\Gamma \vdash_r a : ?$ instead of $\Gamma \vdash a : ?$, and can only be correct if these questions are equivalent.

A formal definition of algorithm is given below by the system $\vdash_{sd}$. The set of inference rules for $\vdash_{sd}$ are *syntax-directed*, which means that there is at most one type derivation for a given term a in a given context Γ, which is completely determined by the syntax of a and Γ (at least, for the so-called functional PTSs). So the rules for $\vdash_{sd}$ effectively provide a type-checking algorithm.†

Definition 14 (Pollack, 1992)
We write $\Gamma \vdash_{sd} a :\rightarrow_\rho A$ for $(\exists_{A'} \Gamma \vdash_{sd} a : A' \wedge A' \rightarrow_\rho A)$, where $\rightarrow_\rho$ is some reduction relation.

The typing relation $\vdash_{sd}$ is the smallest relation closed under the rules:

axiom	$\epsilon \vdash_{sd} s : s'$	$(s : s') \in \mathsf{A}$
sd-variable	$\dfrac{\Gamma \vdash_{sd} A :\twoheadrightarrow_\beta s}{\Gamma, x : A \vdash_{sd} x : A}$	
sd-weakening	$\dfrac{\Gamma \vdash_{sd} b : B \qquad \Gamma \vdash_{sd} A :\twoheadrightarrow_\beta s}{\Gamma, x : A \vdash_{sd} b : B}$	$b \in \mathsf{Var} \cup \mathsf{S}$
sd-formation	$\dfrac{\Gamma \vdash_{sd} A :\twoheadrightarrow_\beta s_1 \qquad \Gamma, x : A \vdash_{sd} B :\twoheadrightarrow_\beta s_2}{\Gamma \vdash_{sd} (\Pi x{:}A.\ B) : s_3}$	$(s_1, s_2, s_3) \in \mathsf{R}$
sd-abstraction	$\dfrac{\Gamma, x : A \vdash_{sd} b : B \qquad \Gamma \vdash_{sd} (\Pi x{:}A.\ B) :\twoheadrightarrow_\beta s}{\Gamma \vdash_{sd} (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B)}$	
sd-application	$\dfrac{\Gamma \vdash_{sd} b :\twoheadrightarrow_{wh} (\Pi x{:}A'.\ B) \qquad \Gamma \vdash_{sd} a : A \qquad A =_\beta A'}{\Gamma \vdash_{sd} ba : B[x := a]}$	

Here $\twoheadrightarrow_{wh}$ denotes *weak-head* reduction. In sd-application weak-head reduction is the quickest way to test if the type of b is β-convertible to a Π-type. In sd-weakening the side-condition $b \in \mathsf{Var} \cup \mathsf{S}$ fixes a strategy for weakening, namely as high up in the derivation tree as possible.

It is easy to see that the algorithm given by $\vdash_{sd}$ is *sound* (i.e. if it finds a type then this type is correct):

Theorem 15 (Soundness, $\vdash_{sd} \subseteq \vdash$)
If $\Gamma \vdash_{sd} a : A$ then $\Gamma \vdash a : A$.

Proof
Easy induction on $\Gamma \vdash_{sd} a : A$. Just observe that all the $\vdash_{nf}$-rules are derivable rules for $\vdash_r$ (and hence $\vdash$). □

However, it is not so simple to prove that the algorithm is *complete* (i.e. if a term is typable then it will find a type):

† In the same way, the rules for $\vdash_{nf}$ also provide a type-checking algorithm, but reducing all types to normal form is unacceptably inefficient for all but the simplest PTSs.

Open Problem 16 (Completeness, $\vdash \subseteq \vdash_{sd}$)
If $\Gamma \vdash a : A$ then $\Gamma \vdash_{sd} a : A'$ for some $A' =_\beta A$?

In fact, this problem has to be restricted to the so-called functional PTSs; a counterexample for a non-functional PTS is given in (Pollack, 1992).

Since $\vdash_{sd} \subseteq \vdash_r \subseteq \vdash$ it is clear that EP ($\vdash \subseteq \vdash_r$) is a necessary condition for $\vdash \subseteq \vdash_{sd}$. Like EP, proving $\vdash \subseteq \vdash_{sd}$ by induction on the derivation also fails in the case that the last step is abstraction.

There are two ways in which EP might help us to solve the problem above. Firstly, having proved EP the proof obligation $\vdash \subseteq \vdash_{sd}$ above can be reduced to $\vdash_r \subseteq \vdash_{sd}$. Unfortunately, this is still an open problem. Secondly, the problems we encounter when trying to prove completeness of $\vdash_{sd}$ and EP are similar, so a method for proving EP might also useful for proving completeness of $\vdash_{sd}$. But for the method we used this is not the case: $\vdash_{nf} \subseteq \vdash_{sd}$ does not seem any easier to prove than $\vdash \subseteq \vdash_{sd}$.

References

Barendregt, H.P. (1991). Introduction to Generalised Type Systems. *Journal of Functional Programming*, **1**(2), 124–154.

Barendregt, H.P. (1992). Lambda calculi with types. Gabbai, D.M., Abramsky, S., and Maibaum, T.S.E. (eds), *Handbook of Logic in Computer Science*, vol. 1. Oxford University Press.

Benthem Jutting, B. van, McKinna, J., and Pollack, R. (1993). Checking Algorithms for Pure Type Systems. *Pages 19–62 of: Types for programs and proofs.* LNCS, vol. 806. Springer.

Poll, E. (1993). *A Typechecker for Bijective Pure Type Systems.* Computing Science Note (93/22). Eindhoven University of Technology.

Pollack, R. (1992). Typechecking in Pure Type Systems. *Pages 271–288 of: Informal proceedings of the 1992 Workshop on Types for Programs and Proofs, Båstad, Sweden.*

Pollack, R. (1994). *The theory of LEGO.* Ph.D. thesis, University of Edinburgh.