A typechecker for bijective pure type systems

Eindhoven University of Technology

Department of Mathematics and Computing Science

A Typechecker for Bijective Pure Type Systems

by

Erik Poll

93/22

Computing Science Note 93/22
Eindhoven, June 1993

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. M. Philips
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

A Typechecker for Bijective Pure Type Systems

Erik Poll*

Abstract. A type inference algorithm is given, which closely follows to the type derivation rules for Pure Type Systems. Soundness and completeness of the algorithm are proved for a large class of Pure Type Systems, which includes all systems in Barendregt's λ-cube.

* supported by the Dutch organization for scientific research (NWO).

1 Introduction

For a Pure Type System (PTS) (see [Bar92]) the central notion is that of type assignment. Every PTS comes with its typing relation $\vdash$, defined by a set of inference rules. Type judgements are of the form $\Gamma \vdash a : A$ – in context Γ the term a has type A – where Γ is a list of type assignments to variables.

For an implementation of a PTS a reasonably efficient type-checker is needed. (If a type-checker reduces types to their normal forms we do not consider it to be 'reasonably efficient'.) It is not difficult to define a type-checker that closely follows the type inference rules (see [Pol92]). Proving soundness of this algorithm is easy, but all attempts at proving completeness have so far failed.

Instead of type judgements of the form

$$x_1 : A_1, \ldots, x_n : A_n \ \vdash \ a : A$$

we consider more informative judgements of the form

$$x_1 : A_1 : s_1, \ldots, x_n : A_n : s_n \ \Vdash \ a : A : s$$

These judgements do not only involve the types of terms but also the types of these types. In this new typing relation a term has exactly the same types as in the old typing relation.

As for the original typing relation, for the new typing relation $\Vdash$ there is a natural typechecking algorithm which closely follows the type inference rules. For this typechecking algorithm soundness and completeness can be proved by straightforward induction proofs for a large class of PTS, which we call the *bijective* PTS (definition 3.5 on page 9). The bijective PTS are a subclass of the functional PTS. It includes all functional PTS with rules of the form (s_1, s_2, s_2), and hence all systems in Barendregt's λ-cube.

In the next section we recall the definition of a PTS given in [Bar92] and discuss the problem of typechecking PTS.

Then, in section 3, a new typing relation is introduced, with judgements of the form $x_1 : A_1 : s_1, \ldots, x_n : A_n : s_n \ \Vdash \ a : A : s$. Two versions of this typing relation are given: one that can be used for all functional PTS ($\Vdash_f$) and one that can only be used for the bijective PTS ($\Vdash_b$).

In section 4 a syntax-directed version $\Vdash_{bsd}$ of the typing relation $\Vdash_b$ is defined (definition 4.1). This typing relation gives a sound and complete typechecking algorithm for all strongly normalising bijective PTS.

In section 5 a syntax-directed version of the typing relation $\Vdash_f$ for functional PTS is defined, for which proving completeness is a problem unfortunately.

Finally, in the last section, the main steps leading from the original typing relation to the syntax-directed one are outlined, and we sketch how the typechecker can be extended to include other primitives, such as Σ, $+$, or $\times$-types.

All the proofs that the different typing relations are equivalent are by induction on derivations and are given in the appendices.

2 Pure Type Systems

Definition 2.1. [Pure Type System]
A Pure Type System (PTS) is a triple $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ with

- $\boldsymbol{S}$ is a set of symbols called the *sorts*
- $\boldsymbol{A} \subseteq \boldsymbol{S} \times \boldsymbol{S}$, a set of *axioms.*
- $\boldsymbol{R} \subseteq \boldsymbol{S} \times \boldsymbol{S} \times \boldsymbol{S}$, a set of *rules*

Definition 2.2. [terms and contexts]
Given a PTS $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$, the collection of *terms* M and *contexts* Γ is given by

$$\begin{array}{l} M ::= x \mid s \mid (MM) \mid (\lambda x{:}M.\ M) \mid (\Pi x{:}M.\ M) \\ \Gamma ::= \epsilon \mid \Gamma, x{:}M \end{array}$$

where x is a variable and s is a sort.

Convention s, s_1, s', etc. range over $\boldsymbol{S}$; a, b, A, B, a', etc. range over terms.

Definition 2.3. [typing relation $\Gamma \vdash b : B$]
For each PTS $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$, a typing relation $\vdash$ is defined. For type judgements of the form $\Gamma \vdash b : B$ we have the following inference rules

$$(axiom) \qquad \epsilon \vdash s : s' \qquad \text{if } (s : s') \in \boldsymbol{A}$$

$$(variable) \qquad \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \qquad x \text{ fresh}$$

$$(weakening) \qquad \frac{\Gamma \vdash b : B \quad \Gamma \vdash A : s}{\Gamma, x : A \vdash b : B} \qquad x \text{ fresh}$$

$$(formation) \qquad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x{:}A.\ B) : s_3} \qquad \text{if } (s_1, s_2, s_3) \in \boldsymbol{R}$$

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

$$(application) \qquad \frac{\Gamma \vdash b : (\Pi x{:}A.\ B) \quad \Gamma \vdash a : A}{\Gamma \vdash ba : B[x := a]}$$

$$(conversion) \qquad \frac{\Gamma \vdash b : B \quad \Gamma \vdash B' : s \quad B \simeq B'}{\Gamma \vdash b : B'}$$

where s ranges over sorts, i.e. $s \in \boldsymbol{S}$, and $\simeq$ is β-equality.

The most important subclass of PTS is that of the *functional* PTS. Most – if not all – PTS that are of practical interest are functional PTS.

Definition 2.4. A PTS $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ is called *functional* (or singly sorted) iff

$$\begin{array}{rcl} (s : s') \in \boldsymbol{A} \wedge (s : s'') \in \boldsymbol{A} & \Longrightarrow & s' \equiv s'' \\ (s_1, s_2, s_3) \in \boldsymbol{R} \wedge (s_1, s_2, s_3') \in \boldsymbol{R} & \Longrightarrow & s_3 \equiv s_3' \end{array}$$

The distinguishing property of these PTS is that the type of a term is unique up to β-equality:

Lemma. [unicity of types for $\vdash : UT_{\vdash}$]
For functional PTS: if $\Gamma \vdash b : B$ and $\Gamma \vdash b : B'$, then $B \simeq B'$.

2.1 Typechecking Pure Type Systems

Two typechecking problems can be distinguished

- $\Gamma \vdash b : B?$ – the type *checking* problem –
  given a context Γ and terms b and B decide whether $\Gamma \vdash b : B$ derivable or not.
- $\Gamma \vdash b : ?$ – the type *inference* problem –
  given a context Γ and a term b, find a term B such that $\Gamma \vdash b : B$, or report failure if no such B exists.
  For functional PTS finding a single type for b suffices, as it is unique modulo β. For non-functional PTS there may be several β-equivalence classes of types for b, in which case we would want to find a representative of each one.

The second is the more general of the two problems: if we can solve $\Gamma \vdash b : ?$ we can also solve $\Gamma \vdash b : B?$. On the other hand, it is difficult to imagine a solution for $\Gamma \vdash b : B?$ which does not also provide a way to solve $\Gamma \vdash b : ?$.

The remainder of this section concerns our motivation for considering a new typing relation. We discuss the natural type inference algorithm for functional PTS defined in [Pol92], and the problem encountered for this algorithm, which is that nobody has been able to prove completeness. For the new typing relation introduced in the next section, we will not have this problem.

The obvious way to solve $\Gamma \vdash b : ?$ is to try to construct a type derivation for b guided by the shape of b. Here we run into the problem that that the PTS inference rules are very non-deterministic: there can be many derivations for $\Gamma \vdash b : B$, let alone for $\Gamma \vdash b : ?$. The source of this non-determinism is the conversion rule, which can be used at any point in a derivation. (The weakening rule also introduces non-determinism, but this can easily be eliminated by restricting the weakening rule to variables and sorts, as in the $\vdash_{\mathrm{sd}}$-weakening rule below).

Despite this non-determinism, it is not difficult to define a natural type inference algorithm which tries to construct a particular type derivation, by only doing conversions

- to test if a type reduces to a sort
- to test if the types of a function and its argument match

But then we are implementing the following syntax-directed system $\vdash_{\mathrm{sd}}$ ($_{\mathrm{sd}}$ for syntax-directed) : instead of $\vdash$

$$(\textit{axiom}) \qquad \epsilon \vdash_{\mathrm{sd}} s : s' \qquad \text{if } (s : s') \in \boldsymbol{A}$$

$$(\textit{variable}) \qquad \frac{\Gamma \vdash_{\mathrm{sd}} A : S \quad S \twoheadrightarrow_\beta s}{\Gamma, x : A \vdash_{\mathrm{sd}} x : A} \qquad x \text{ fresh}$$

$$(\textit{weakening}) \qquad \frac{\Gamma \vdash_{\mathrm{sd}} b : B \quad \Gamma \vdash_{\mathrm{sd}} A : S \quad S \twoheadrightarrow_\beta s}{\Gamma, x : A \vdash_{\mathrm{sd}} b : B} \quad x \text{ fresh and } b \text{ a variable or sort}$$

$$(\textit{formation}) \qquad \frac{\Gamma \vdash_{\mathrm{sd}} A : S_1 \qquad S_1 \twoheadrightarrow_\beta s_1 \qquad \Gamma, x : A \vdash_{\mathrm{sd}} B : s_2 \qquad S_2 \twoheadrightarrow_\beta s_2}{\Gamma \vdash_{\mathrm{sd}} (\Pi x : A.\ B) : s_3} \quad \text{if } (s_1, s_2, s_3) \in \boldsymbol{R}$$

$$(\textit{abstraction}) \qquad \frac{\Gamma, x : A \vdash_{\mathrm{sd}} b : B \qquad \Gamma \vdash_{\mathrm{sd}} (\Pi x : A.\ B) : S \quad S \twoheadrightarrow_\beta s}{\Gamma \vdash_{\mathrm{sd}} (\lambda x : A.\ b) : (\Pi x : A.\ B)}$$

$$(\textit{application}) \qquad \frac{\Gamma \vdash_{\mathrm{sd}} b : C \quad C \twoheadrightarrow_{\mathrm{wh}} (\Pi x : A'.\ B) \qquad \Gamma \vdash_{\mathrm{sd}} a : A \quad A \simeq A'}{\Gamma \vdash_{\mathrm{sd}} ba : B[x := a]}$$

Here $\twoheadrightarrow_\beta$ denotes β-reduction and $\twoheadrightarrow_{\mathrm{wh}}$ denotes weak-head reduction. Note that the weakening rule is restricted to variables and sorts. This means that in derivations weakening is postponed as much as possible. The shape of a term b determines a unique $\vdash_{\mathrm{sd}}$-type derivation for b.

If our algorithm implements $\vdash_{sd}$ instead of $\vdash$, we want to know if $\vdash_{sd}$ and $\vdash$ are equivalent. This means we have to prove

$$\begin{array}{rl} \text{soundness}: & \Gamma \vdash_{sd} c : C \Longrightarrow \Gamma \vdash c : C \\ \text{completeness}: & \Gamma \vdash c : C \Longrightarrow (\exists C' \simeq C.\ \Gamma \vdash_{sd} c : C') \end{array}$$

It is easy to prove soundness. However, proving completeness is a problem. This means it is not known if for a functional PTS the algorithm can always find a type for a typeable term. (For non-functional PTS completeness fails; see [Pol92] for a counterexample.)

A proof of completeness by induction on the derivation $\Gamma \vdash c : C$ fails in the abstraction rule:

> Suppose the last step in the derivation of $\Gamma \vdash c : C$ is the abstraction rule. Then $c \equiv (\lambda x : A.\ b)$, $C \equiv (\Pi x : A.\ B)$ and the last step in the derivation is
>
> $$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x : A.\ B) : s}{\Gamma \vdash (\lambda x : A.\ b) : (\Pi x : A.\ B)}$$
>
> By the induction hypothesis $\Gamma, x : A \vdash_{sd} b : B'$ for some $B' \simeq B$ and $\Gamma \vdash_{sd} (\Pi x : A.\ B) : S$ for some $S \simeq s$. However, the type B' found for b can be different from B (but β-equivalent to B), and we do not know if $\Gamma \vdash_{sd} (\Pi x : A.\ B') : S'$ for some S'.

Basically, the problem is the way in which the subformula property fails for the abstraction rule: a type derivation for $(\lambda x : A.\ b)$ in context Γ contains a sub-derivation of a term – $(\Pi x : A.\ B)$ – which is *not* a subterm of $(\lambda x : A.\ b)$.

An alternative abstraction rule, which is equivalent with the original abstraction rule, is

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash b : B \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\lambda x : A.\ b) : (\Pi x : A.\ B)} \text{ if } (s_1, s_2, s_3) \in \boldsymbol{R}$$

For this abstraction rule we have the same problem, this time caused by the premiss $\Gamma, x : A \vdash B : s_2$ instead of the premiss $\Gamma \vdash (\Pi x : A.\ B) : s$.

In section 6 two type checking algorithms are mentioned which successfully avoid this problem by relaxing the condition $\Gamma \vdash (\Pi x : A.\ B) : s$, or rather the condition $\Gamma, x : A \vdash B : s_2$ in the alternative abstraction rule given above.

We avoid the problem by considering derivations for judgements of the form $\Gamma \vdash a : A : s$, as explained in the next section.

3 Type-kind derivations

Instead of judgements of the form

$$x_1 : A_1, \ldots, x_n : A_n \vdash a : A$$

we consider judgements of the form

$$x_1 : A_1 : s_1, \ldots, x_n : A_n : s_n \vdash a : A : s$$

These judgements not only give the type of a term, but also the type of its type, which we call its *kind*. For example, in the judgement above s is the kind of a and each s_i is the kind of x_i.

The notion of kind is related to the notion of *degree* – defined in [dB80] for the AUTOMATH languages and in [Bar92] for the λ-cube – in that terms with the same kind have the same degree.

Contexts are now of the form

$$\Delta ::= \epsilon \mid \Delta, x : A : s$$

In contexts both the type and the kind of a variable are registered.

We have to be careful for non-functional PTS! Here a variable x of type A may have more than one kind s. It may be essential for different occurrences of x to have different kinds, so that by fixing the kind for x some terms are no longer typeable. Later an example will be given that shows that for non-functional PTS our new typing relation is *not* equivalent to the original typing relation.

It turns out that, when deriving the type of a term, we can derive its kind at the same time at little extra cost. Some $\vdash$-inference rules that derive a type B for a term b, could also produce the type of this B:

$$(\textit{variable}) \qquad \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$$

$$(\textit{abstraction}) \ \frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x : A.\ B) : s_3}{\Gamma \vdash (\lambda x : A.\ b) : (\Pi x : A.\ B)}$$

but here the type s of A and the type s_3 of $(\Pi x : A.\ B)$ are not passed on in the conclusion. Also, kinds cannot be large terms; by the following lemma the kind of a term is always a sort.

Lemma 3.1. [correctness of types]
If $\Gamma \vdash b : B$ then $\Gamma \vdash B : s$ for some $s \in \boldsymbol{S}$ or $B \in \boldsymbol{S}$.

So the extra amount of information carried around in $\vdash$-derivations in comparison to $\vdash$-derivations is small.

A problem is not all terms that have a type also have a kind. If $\Gamma \vdash b : B$ and $B \in \boldsymbol{S}$, then there may not be a type for B, and hence no kind for b. This is solved by introducing a new "sort" $\top$, which acts as the type for hitherto untypeable sorts:

Definition 3.2. $[\boldsymbol{S}_\top, \boldsymbol{A}_\top]$
Given a PTS $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ we define

$$\boldsymbol{S}_\top = \boldsymbol{S} \cup \{\top\}$$
$$\boldsymbol{A}_\top = \boldsymbol{A} \cup \{(s : \top) \mid s \in \boldsymbol{S} \wedge \neg(\exists_{s'}(s : s') \in \boldsymbol{A})\}$$

So for all $s \in \boldsymbol{S}$ there is a $s' \in \boldsymbol{S}_\top$ such that $(s : s') \in \boldsymbol{A}_\top$. For functional PTS this s' is unique, i.e. $\boldsymbol{A}_\top$ is a function from $\boldsymbol{S}$ to $\boldsymbol{S}_\top$. For functional PTS we write $\boldsymbol{A}_\top(s)$ for the $s' \in \boldsymbol{S}_\top$ for which $(s : s') \in \boldsymbol{A}_\top$.

Convention s, s_1, s', etc. range over $\boldsymbol{S}_\top$ (and not just $\boldsymbol{S}$).

Replacing all judgements of the form $\Gamma \vdash b : B$ by judgements $\Delta \vdash b : B : s$ in the PTS inference rules, we get the following typing relation $\vdash_{\mathrm{f}}$. $\vdash_{\mathrm{f}}$ is defined for arbitrary PTS, but only for functional PTS will it be equivalent to the original typing relation $\vdash$ (hence the subscript f for functional):

Definition 3.3. $[\Delta \vdash_{\mathrm{f}} b : B : s]$

$$(axiom) \qquad \epsilon \vdash_{\mathrm{f}} s : s' : \boldsymbol{A}_{\mathsf{T}}(s') \qquad \text{if } (s : s') \in \boldsymbol{A}$$

$$(variable) \qquad \frac{\Delta \vdash_{\mathrm{f}} A : s_1 : s_1'}{\Delta, x : A : s_1 \vdash_{\mathrm{f}} x : A : s_1} \qquad x \text{ fresh}$$

$$(weakening) \qquad \frac{\Delta \vdash_{\mathrm{f}} b : B : s \quad \Delta \vdash_{\mathrm{f}} A : s_1 : s_1'}{\Delta, x : A : s_1 \vdash_{\mathrm{f}} b : B : s} \qquad x \text{ fresh}$$

$$(formation) \qquad \frac{\Delta \vdash_{\mathrm{f}} A : s_1 : s_1' \quad \Delta, x : A : s_1 \vdash_{\mathrm{f}} B : s_2 : s_2'}{\Delta \vdash_{\mathrm{f}} (\Pi x : A.\ B) : s_3 : \boldsymbol{A}_{\mathsf{T}}(s_3)} \qquad \text{if } (s_1, s_2, s_3) \in \boldsymbol{R}$$

$$(abstraction) \qquad \frac{\Delta \vdash_{\mathrm{f}} A : s_1 : s_1' \quad \Delta, x : A : s_1 \vdash_{\mathrm{f}} b : B : s_2}{\Delta \vdash_{\mathrm{f}} (\lambda x : A.\ b) : (\Pi x : A.\ B) : s_3} \qquad \text{if } (s_1, s_2, s_3) \in \boldsymbol{R}$$

$$(application) \qquad \frac{\begin{array}{l}\Delta \vdash_{\mathrm{f}} b : (\Pi x : A.\ B) : s_3 \\ \Delta \vdash_{\mathrm{f}} a : A : s_1 \\ \Delta \vdash_{\mathrm{f}} B[x := a] : s_2 : s_2'\end{array}}{\Delta \vdash_{\mathrm{f}} ba : B[x := a] : s_2}$$

$$(conversion) \qquad \frac{\Delta \vdash_{\mathrm{f}} b : B : s \quad \Delta \vdash_{\mathrm{f}} B' : s : s' \quad B \simeq B'}{\Delta \vdash_{\mathrm{f}} b : B' : s}$$

Compared with the original typing rules, the major differences are in the abstraction and the application rule. These are discussed below. For all other rules there is a one-to-one correspondence between the premisses in the $\vdash_{\mathrm{f}}$-rule and the corresponding $\vdash$-rule.

Note that the conversion rule changes the type of a term, but not its kind. This is because in functional PTS the kind of a term is unique (not just unique up to β-equality).

For functional PTS, $\vdash_{\mathrm{f}}$ is equivalent with $\vdash$:

Theorem 3.4. [soundness and completeness $\vdash_{\mathrm{f}}$]
For all functional PTS

$$\begin{array}{rcl} \Gamma \vdash b : B & \Longrightarrow & (\exists s \in \boldsymbol{S}_{\mathsf{T}}.\ \tilde{\Gamma} \vdash_{\mathrm{f}} b : B : s) \\ \Delta \vdash_{\mathrm{f}} b : B : s & \Longrightarrow & |\Delta| \vdash b : B \ \wedge\ (|\Delta| \vdash B : s \ \vee\ (B : s) \in \boldsymbol{A}_{\mathsf{T}} \backslash \boldsymbol{A}) \end{array}$$

where $|\Delta|$ *is defined by* $\begin{cases} |\epsilon| &= \epsilon \\ |\Delta, x : A : s| &= |\Delta|, x : A \end{cases}$

and $\tilde{\Gamma}$ *is defined by* $\begin{cases} \tilde{\epsilon} &= \epsilon \\ \widetilde{\Gamma, x : A} &= \tilde{\Gamma}, x : A : s \quad \text{if } \Gamma \vdash A : s. \end{cases}$

Proof. See appendix D: lemmas D.1, D.7 and D.8. □

For non-functional PTS there can be terms that are typeable using $\vdash$, but that are not typeable using $\vdash_{\mathrm{f}}$, as shown in the following example.

Consider the non-functional PTS

$$\boldsymbol{S} = \{*, \Box_1, \Box_2\}, \quad \boldsymbol{A} = \{* : \Box_1,\ * : \Box_2\}, \quad \boldsymbol{R} = \{(\Box_1, \Box_2, \Box_2)\}$$

$\epsilon \vdash (\lambda x{:}*.\ x) : (\Pi x{:}*.\ *)$ is derivable. To derive this we have to derive $\epsilon \vdash (\Pi x{:}*.\ *) : \Box_2$. To derive $\epsilon \vdash (\Pi x{:}*.\ *) : \Box_2$, the first occurrence of $*$ has to be typed $\Box_1$ and the second occurrence of $*$ has to be typed $\Box_2$:

$$\frac{x : * \vdash x : * \qquad \dfrac{\vdash * : \Box_1 \quad x : * \vdash * : \Box_2}{\vdash (\Pi x{:}*.\ *) : \Box_2}\ (\Box_1, \Box_2, \Box_2) \in \boldsymbol{R}}{\vdash (\lambda x{:}*.\ x) : (\Pi x{:}*.\ *)}$$

Both occurrences of $*$ in $(\Pi x{:}*.\ *)$ stem from occurrences of x in $(\lambda x{:}*.\ x)$. If the kind of x is fixed in a context $x : * : \Box_i$, then both occurrences of $*$ will have the same type $\Box_i$. As a consequence, $(\lambda x{:}*.\ x)$ cannot be typed using $\vdash_f$:

$$\frac{\epsilon \vdash_f * : \Box_i : \top \quad x{:}*{:}\Box_i \vdash_f x{:}*{:}\Box_i}{\epsilon \vdash_f (\lambda x{:}*.\ x) : (\Pi x{:}*.\ *) : s}\ (\Box_i, \Box_i, s) \in \boldsymbol{R}????$$

This type derivation fails because there is no kind $\Box_i$ for x such that $(\Box_i, \Box_i, s) \in \boldsymbol{R}$ for some s.

As mentioned earlier, the main differences between $\vdash$ and $\vdash_f$ are in the abstraction and the application rule.

The abstraction rule

In the $\vdash_f$-abstraction rule we save work. Unfolding the $\vdash$-formation rule in the $\vdash$-abstraction rule we get

$$\frac{\Gamma, x : A \vdash \mathbf{b} : B \qquad \dfrac{\Gamma \vdash \mathbf{A} : s_1 \quad \Gamma, x : A \vdash \mathbf{B} : s_2}{\Gamma \vdash (\Pi x{:}A.\ B) : s_3}\ (s_1, s_2, s_3) \in \boldsymbol{R}}{\Gamma \vdash (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B)}$$

In this derivation, in order to type $(\lambda x{:}A.\ b)$, the terms b, A and B are typed.

In the $\vdash_f$-abstraction rule

$$\frac{\Delta, x{:}A{:}s_1 \vdash_f \mathbf{b} : B : s_2 \qquad \Delta \vdash_f \mathbf{A} : s_1 : s_1'}{\Delta \vdash_f (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B) : s_3}(s_1, s_2, s_3) \in \boldsymbol{R}$$

Here only b and A have to be typed. A separate type derivation for B is no longer needed, because by typing b we also obtain the type of B. Remember that in section 2.1 it was the premiss $\Gamma, x : A \vdash B : s_2$ in the abstraction rule that caused the problem in the unsuccessful completeness proof.

The application rule

In the $\vdash_f$-application rule more work has to be done to produce not only the type but also the kind. The original application rule is

$$\frac{\Gamma \vdash b : (\Pi x{:}A.\ B) \qquad \Gamma \vdash a : A}{\Gamma \vdash ba : B[x := a]}$$

and the new application rule is

$$\frac{\Delta \vdash_f b : (\Pi x{:}A.\ B) : s_3 \qquad \Delta \vdash_f a : A : s_1 \qquad \Delta \vdash_f B[x := a] : s_2 : s_2'}{\Delta \vdash_f ba : B[x := a] : s_2}$$

The extra work is a type derivation for $B[x := a]$, which is needed to determine the kind of the application ba. (It turns out that, for a syntax-directed version of $\vdash_f$, the premiss $\Delta \vdash_f B[x := a] : s_2 : s_2'$ causes the same problem in the proof of completeness that we sketched earlier for $\vdash$. This is discussed in section 5.)

Fortunately, this extra work can be avoided for a large class of PTS, namely those where given s_1 and s_3 there is at most one s_2 such that $(s_1, s_2, s_3) \in \boldsymbol{R}$. In these PTS the kind of an application is determined by the kind of the function and the kind of its argument.

Definition 3.5. A PTS is called *bijective* iff it is functional and for all $s_1, s_2, s_2', s_3 \in \boldsymbol{S}$

$$(s_1, s_2, s_3) \in \boldsymbol{R} \wedge (s_1, s_2', s_3) \in \boldsymbol{R} \implies s_2 \equiv s_2' \ .$$

So a PTS is bijective iff

$$\begin{array}{rl} (s, s') \in \boldsymbol{A} \wedge (s, s'') \in \boldsymbol{A} & \implies s' \equiv s'' \\ (s_1, s_2, s_3) \in \boldsymbol{R} \wedge (s_1, s_2, s_3') \in \boldsymbol{R} & \implies s_3 \equiv s_3' \\ (s_1, s_2, s_3) \in \boldsymbol{R} \wedge (s_1, s_2', s_3) \in \boldsymbol{R} & \implies s_2 \equiv s_2' \end{array}$$

All functional PTS that only have rules of the form (s_1, s_2, s_2) are bijective. This means that all systems in Barendregt's λ-cube are bijective.

For bijective PTS, the kind of a term is uniquely determined by the kinds of it subterms. As a consequence, the premiss $\Delta \vdash_{\mathrm{f}} B[x := a] : s_2 : s_2'$ can be omitted in the application rule.

Definition 3.6. $[\Delta \vdash_{\mathrm{b}} b : B : s]$
$\vdash_{\mathrm{b}}$ is defined by the same inference rules as $\vdash_{\mathrm{f}}$, except for the application rule.

$$\begin{array}{lll} (\mathit{axiom}) & \epsilon \vdash_{\mathrm{b}} s : s' : \boldsymbol{A}_{\mathsf{T}}(s') & \text{if } (s : s') \in \boldsymbol{A} \\ (\mathit{variable}) & \dfrac{\Delta \vdash_{\mathrm{b}} A : s_1 : s_1'}{\Delta, x : A : s_1 \vdash_{\mathrm{b}} x : A : s_1} & x \text{ fresh} \\ (\mathit{weakening}) & \dfrac{\Delta \vdash_{\mathrm{b}} b : B : s \quad \Delta \vdash_{\mathrm{b}} A : s_1 : s_1'}{\Delta, x : A : s_1 \vdash_{\mathrm{b}} b : B : s} & x \text{ fresh} \\ (\mathit{formation}) & \dfrac{\Delta \vdash_{\mathrm{b}} A : s_1 : s_1' \quad \Delta, x : A : s_1 \vdash_{\mathrm{b}} B : s_2 : s_2'}{\Delta \vdash_{\mathrm{b}} (\Pi x : A.\ B) : s_3 : \boldsymbol{A}_{\mathsf{T}}(s_3)} & \text{if } (s_1, s_2, s_3) \in \boldsymbol{R} \\ (\mathit{abstraction}) & \dfrac{\Delta \vdash_{\mathrm{b}} A : s_1 : s_1' \quad \Delta, x : A : s_1 \vdash_{\mathrm{b}} b : B : s_2}{\Delta \vdash_{\mathrm{b}} (\lambda x : A.\ b) : (\Pi x : A.\ B) : s_3} & \text{if } (s_1, s_2, s_3) \in \boldsymbol{R} \\ \textbf{(application)} & \dfrac{\Delta \vdash_{\mathrm{b}} b : (\Pi x : A.\ B) : s_3 \quad \Delta \vdash_{\mathrm{b}} a : A : s_1}{\Delta \vdash_{\mathrm{b}} ba : B[x := a] : s_2} & \text{if } (s_1, s_2, s_3) \in \boldsymbol{R} \\ (\mathit{conversion}) & \dfrac{\Delta \vdash_{\mathrm{b}} b : B : s \quad \Delta \vdash_{\mathrm{b}} B' : s : s' \quad B \simeq B'}{\Delta \vdash_{\mathrm{b}} b : B' : s} & \end{array}$$

For bijective PTS, $\vdash_{\mathrm{b}}$ is equivalent with $\vdash$:

Theorem 3.7. [soundness and completeness $\vdash_{\mathrm{b}}$]
For all bijective PTS

$$\begin{array}{rcl} \Gamma \vdash b : B & \implies & (\exists s \in \boldsymbol{S}_{\mathsf{T}}.\ \tilde{\Gamma} \vdash_{\mathrm{b}} b : B : s) \\ \Delta \vdash_{\mathrm{b}} b : B : s & \implies & |\Delta| \vdash b : B \ \wedge \ (|\Delta| \vdash B : s \ \vee \ (B : s) \in \boldsymbol{A}_{\mathsf{T}} \backslash \boldsymbol{A}) \end{array}$$

where $|\Delta|$ and $\tilde{\Gamma}$ are defined as in theorem 3.4.

Proof. See appendix B: lemmas B.1, B.6, and B.7. □

The proofs of theorems 3.4 and 3.7 are by induction on type derivations, and hence are identical except when the application rule is involved. An alternative to proving theorem 3.7 is proving that for all bijective PTS $\Delta \vdash_{\mathrm{f}} b : B : s \iff \Delta \vdash_{\mathrm{b}} b : B : s$.

4 A type-checker for bijective PTS

In this section a syntax-directed version of $\vdash_b$ is defined, by removing the weakening and conversion rule.

$\vdash_b$ deals with contexts in the same inefficient way as $\vdash$: in every branch of a derivation the context is broken down to check that it is well-formed. To avoid this a notion of well-formedness for contexts is introduced and the original axiom and variable rule are replaced by the following more powerful axiom and variable rules

$$(axiom) \qquad \Delta \vdash_{bsd} s : s' : \boldsymbol{A}_T(s') \quad \text{if } s : s' \in \boldsymbol{A}$$

$$(variable) \quad \Delta, x : A : s, \Delta' \vdash_{bsd} x : A : s$$

The conversion rule is a source of non-determinism in $\vdash_b$. It is distributed over the other rules. Conversion is only used to test if

- a type reduces to a sort
- the types of a function and its argument match

The resulting system is syntax-directed: the shape of a term determines which inference rule is used in the last step of its type derivation, and so there is a *unique* type derivation for a given term. As a consequence, the system provides a type inference algorithm.

Definition 4.1. $[\Delta \vdash_{bsd} b : B : s,\ WF_{bsd}\Delta]$
For $\Delta \vdash_{bsd} b : B : s$ we have the following inference rules

$$(axiom) \qquad \Delta \vdash_{bsd} s : s' : \boldsymbol{A}_T(s') \qquad \text{if } s : s' \in \boldsymbol{A}$$

$$(variable) \qquad \Delta, x : A : s, \Delta' \vdash_{bsd} x : A : s$$

$$(formation) \quad \frac{\Delta, x : A : s_1 \vdash_{bsd} B : S_2 : s_2' \qquad S_2 \twoheadrightarrow_\beta s_2 \qquad \Delta \vdash_{bsd} A : S_1 : s_1' \qquad S_1 \twoheadrightarrow_\beta s_1}{\Delta \vdash_{bsd} (\Pi x : A.\ B) : s_3 : \boldsymbol{A}_T(s_3)} \quad \text{if } (s_1, s_2, s_3) \in \boldsymbol{R} \text{ and } x \text{ fresh}$$

$$(abstraction) \quad \frac{\Delta, x : A : s_1 \vdash_{bsd} b : B : s_2 \qquad \Delta \vdash_{bsd} A : S_1 : s_1' \qquad S_1 \twoheadrightarrow_\beta s_1}{\Delta \vdash_{bsd} (\lambda x : A.\ b) : (\Pi x : A.\ B) : s_3} \quad \text{if } (s_1, s_2, s_3) \in \boldsymbol{R} \text{ and } x \text{ fresh}$$

$$(application) \quad \frac{\Delta \vdash_{bsd} b : C : s_3 \qquad C \twoheadrightarrow_{wh} (\Pi x : A'.\ B) \qquad \Delta \vdash_{bsd} a : A : s_1 \qquad A \simeq A'}{\Delta \vdash_{bsd} ba : B[x := a] : s_2} \quad \text{if } (s_1, s_2, s_3) \in \boldsymbol{R}$$

A context Δ is $\vdash_{bsd}$-well-formed – written $WF_{bsd}\Delta$ – if it can be derived using the rules

$$(empty) \qquad WF_{bsd}\epsilon$$

$$(weakening) \quad \frac{WF_{bsd}\Delta \qquad \Delta \vdash_{bsd} A : S : s' \qquad S \twoheadrightarrow_\beta s}{WF_{bsd}\Delta, x : A : s} \quad x \text{ fresh}$$

Note that for $\vdash_{bsd}$-derivations the subformula property holds, i.e. the derivation of $\Delta \vdash_{bsd} b : B : s$ only contains type-kind derivations for subterms of b.

Theorem 4.2. [soundness and completeness $\vdash_{bsd}$]
For all bijective PTS:

$$\Delta \vdash_{bsd} b : B : s \ \wedge\ WF_{bsd}\Delta \implies \Delta \vdash_b b : B : s$$
$$\Delta \vdash_b b : B : s \implies WF_{bsd}\Delta \ \wedge\ (\exists B' \simeq B.\ \Delta \vdash_{bsd} b : B' : s)$$

Proof. See appendix C: lemmas C.1, C.5 and C.6. □

Joining theorems 3.7 and 4.2 yields

Corollary 4.3. [soundness and completeness of $\vdash_{\mathrm{bsd}}$ w.r.t. $\vdash$]
For all bijective PTS:

$$\begin{array}{rcl} \Delta \vdash_{\mathrm{bsd}} b : B : s \ \wedge \ WF_{\mathrm{bsd}}\Delta & \Longrightarrow & |\Delta| \vdash b : B \ \wedge \ (|\Delta| \vdash B : s \ \vee \ (B : s) \in \boldsymbol{A}_{\mathsf{T}} \backslash \boldsymbol{A}) \\ \Gamma \vdash b : B & \Longrightarrow & (\exists s \in \boldsymbol{S}_{\mathsf{T}}, B' \simeq B.\ \tilde{\Gamma} \vdash_{\mathrm{bsd}} b : B' : s) \end{array}$$

It is easy to define an algorithm which, given a context Δ and a term a, checks $WF_{\mathrm{bsd}}\Delta$ and tries to constructs the type derivation for a in context Δ, thus finding the type (and the kind) of a. This does require strong normalisation of β-reduction for types, i.e.

$$\Gamma \vdash a : A \ \Rightarrow \ A \text{ is } \beta\text{-strongly normalising}$$

(or, equivalently, $WF_{\mathrm{bsd}}\Delta \wedge \Delta \vdash a : A : s \ \Rightarrow \ A$ is β-strongly normalising). This is needed to ensure that β normal forms and weak head normal forms of types can be computed, and that convertibility of types is decidable, so that the premisses $S_i \twoheadrightarrow_\beta s_i$, $C \twoheadrightarrow_{\mathrm{wh}} (\Pi x{:}A.\ B)$ and $A \simeq A'$ are decidable.

The algorithm has to choose a correct order in which to check the different premisses of the rules, so that it nevers attempts to construct a type derivation in a context which may be incorrect. If $\Delta \vdash_{\mathrm{bsd}} a : A : s$ and *not* $WF_{\mathrm{bsd}}\Delta$, then we do not know if A is strongly normalising. This means that in the formation and abstraction rules the premisses $\Delta \vdash_{\mathrm{bsd}} A : S_1 : s'$ and $S_1 \twoheadrightarrow_\beta s_1$ have to be checked first, and in the the weakening rule the premiss $WF_{\mathrm{bsd}}\Delta$ has to be checked first.

5 Functional PTS

For functional PTS that are not bijective a syntax-directed system can be defined in the same way as was done in the previous section, this time taking $\vdash_{\mathrm{f}}$ instead of $\vdash_{\mathrm{b}}$ as the starting point.

Definition 5.1. $[\Delta \vdash_{\mathrm{fsd}} b : B : s,\ WF_{\mathrm{fsd}}\Delta]$
$\vdash_{\mathrm{fsd}}$ and WF_{fsd} are defined as $\vdash_{\mathrm{bsd}}$ and WF_{bsd}, except for the application rule, where we have the following rule instead

$$\frac{\begin{array}{ll} \Delta \vdash_{\mathrm{fsd}} b : C : s_3 & C \twoheadrightarrow_{\mathrm{wh}} (\Pi x{:}A'.\ B) \\ \Delta \vdash_{\mathrm{fsd}} a : A : s_1 & A \simeq A' \\ \Gamma \vdash_{\mathrm{fsd}} B[x := a] : s_2 : s_2' & \end{array}}{\Gamma \vdash_{\mathrm{fsd}} ba : B[x := a] : s_2}$$

Soundness of this system can be proved for all functional PTS:

$$\Delta \vdash_{\mathrm{fsd}} b : B : s \wedge WF_{\mathrm{fsd}} \ \Longrightarrow \ \Delta \vdash_{\mathrm{f}} b : B : s.$$

However, proving completeness for this system,

$$\Delta \vdash_{\mathrm{f}} b : B : s \ \Longrightarrow (\exists B' \simeq B.\ \Delta \vdash_{\mathrm{fsd}} b : B' : s),$$

is a problem. In fact, we now have the same problem with the premiss $\Gamma \vdash_{\mathrm{f}} B[x := a] : s_2 : s_2'$ in the application rule that we sketched in section 2.1 for the premiss $\Gamma \vdash (\Pi x{:}A.\ B) : s$ in the abstraction rule.

6 Related work

Other type-checking algorithms for Pure Type Systems all involve a way of avoiding the premiss $\Gamma, x : A \vdash B : s_2$ in the abstraction rule

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash b : B \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\lambda x : A.\ b) : (\Pi x : A.\ B)} \text{ if } (s_1, s_2, s_3) \in \boldsymbol{R}$$

One possibility is to consider only PTS where $\boldsymbol{R}$ is such a large subset of $\boldsymbol{S} \times \boldsymbol{S} \times \boldsymbol{S}$ that we do not have to know a type s_2 of B to decide if $(\Pi x : A.\ B)$ can be formed. In [Pol92] Pollack gives a sound and complete type inference algorithm for all *semi-full* PTS, which is a generalisation of the type inference algorithm used in Huet's Constructive Engine for the Calculus of Constructions. A PTS is semi-full if

$$\forall_{s_1} \ (\exists_{s_2, s_3}(s_1, s_2, s_3) \in \boldsymbol{R}) \Longrightarrow (\forall_{s_2} \exists_{s_3}(s_1, s_2, s_3) \in \boldsymbol{R})$$

For these PTS, only a type s_1 of A and a type B of b – but not the type of this B – have to be known to decide if the λ-abstraction $(\lambda x : A.\ b)$ is allowed: the rule

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash b : B \quad B \notin \{s \in \boldsymbol{S} \mid \neg \exists_{s'}(s : s') \in \boldsymbol{A}\}}{\Gamma \vdash (\lambda x : A.\ b) : (\Pi x : A.\ B)} \text{ if } (s_1, s_2, s_3) \in \boldsymbol{R}$$

is equivalent with the abstraction rules given above.

For example, for the Calculus of Constructions, the PTS with

$$\boldsymbol{S} = \{*, \Box\} \quad \boldsymbol{A} = \{(* : \Box)\} \quad \boldsymbol{R} = \{(s_1, s_2, s_2) \mid s_1, s_2 \in \boldsymbol{S}\}$$

this abstraction rule becomes

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash b : B \quad B \neq \Box}{\Gamma \vdash (\lambda x : A.\ b) : (\Pi x : A.\ B)}$$

which is in fact the original abstraction rule given in [CH88] for the Calculus of Constructions.

Jutting proved soundness and completeness for a type inference algorithm for all functional PTS [vBJ92]. He considers an abstraction rule of the form

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash b : B \quad \Gamma, x : A \vdash_1 B : s_2}{\Gamma \vdash (\lambda x : A.\ b) : (\Pi x : A.\ B)} \text{ if } (s_1, s_2, s_3) \in \boldsymbol{R}$$

where $\vdash_1$ is a typing relation which is more liberal than $\vdash$. This result has been extended to *all* PTS in [vBJMP93].

$\vdash_{\text{bsd}}$ does not give a type inference algorithm for all PTS, but it has the advantage that it is closer to the original PTS type derivation rules than Jutting's algorithm. This makes it easy to incorporate other type constructors, as is sketched in the next section.

7 Conclusions

For *bijective* PTS the following equivalences have been proved

$$\vdash \overset{3.7}{\Longleftrightarrow} \vdash_{\mathsf{b}} \overset{4.2}{\Longleftrightarrow} \vdash_{\mathsf{bsd}}$$

For *functional* PTS we have the following relationships between the different systems

$$\vdash \overset{3.4}{\Longleftrightarrow} \vdash_{\mathsf{f}} \Longleftarrow \vdash_{\mathsf{fsd}}$$

The steps leading from the original typing relation $\vdash$ to the syntax-directed $\vdash_{\mathsf{bsd}}$ are

1. from $\vdash$ to $\vdash_{\mathsf{b}}$: type-*kind* derivations instead of type derivations, i.e. judgements of the form

   $$x_1 : A_1 : s_1, \ldots, x_n : A_n : s_n \vdash a : A : s$$

   instead of the usual judgements of the form

   $$x_1 : A_1, \ldots, x_n : A_n \vdash a : A$$

2. from $\vdash_{\mathsf{b}}$ to $\vdash_{\mathsf{bsd}}$:
   - making the weakening rule redundant, by introducing a notion of well-formedness for contexts and more powerful axiom and variable rules, and
   - getting rid of the conversion rule by distributing it over the other rules.

In the same way sound and complete type-checkers for PTS's extended with more primitives, such as Cartesian product types, disjoint sum types, Σ or existential types, can be found. For all these extensions we run into the same problem as for the Π-types, namely that we do not know how to prove completeness. By considering type-*kind* derivations this problem is avoided.

For example, for Cartesian products the formation and introduction rules are

$$(\times - \mathit{formation}) \qquad \frac{\Gamma \vdash A : s_1 \quad \Gamma \vdash B : s_2}{\Gamma \vdash A \times B : s_3} \qquad \text{if } (s_1, s_2, s_3) \in \boldsymbol{R}_\times$$

$$(\times - \mathit{introduction}) \; \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B \quad \Gamma \vdash A \times B : s_3}{\Gamma \vdash (a, b) : A \times B}$$

where $\boldsymbol{R}_\times \subseteq \boldsymbol{S}^3$ controls the formation of Cartesian products. If we want to prove completeness of a type-checker for PTS extended with products, the premiss $\Gamma \vdash A \times B : s_3$ in the $\times$-introduction rule poses the same problem as the premiss $\Gamma \vdash (\Pi x{:}A.\ B) : s_3$ in the abstraction rule. Fortunately, this premiss is exactly the one that can be omitted in the $\times$-introduction rule for type-kind derivations:

$$(\times - \mathit{introduction}) \; \frac{\Delta \vdash a : A : s_1 \quad \Delta \vdash b : B : s_2}{\Delta \vdash (a, b) : A \times B : s_3} \text{ if } (s_1, s_2, s_3) \in \boldsymbol{R}_\times$$

However, the restriction of "bijectivity" has to be extended to $\boldsymbol{R}_\times$: the kind of a pair (a, b) has to be uniquely determined by the kinds of of the two components, i.e.

$$(s_1, s_2, s_3) \in \boldsymbol{R}_\times \wedge (s_1, s_2, s_3') \in \boldsymbol{R}_\times \implies s_3 \equiv s_3'$$

and the kinds of the first and second projections of a pair have to be uniquely determined by the kind of that pair, i.e.

$$(s_1, s_2, s_3) \in \boldsymbol{R}_\times \wedge (s_1', s_2', s_3) \in \boldsymbol{R}_\times \implies s_1 \equiv s_1' \wedge s_2 \equiv s_2'$$

Under these restrictions, removing the weakening and conversion rules as was done for $\vdash_{\mathsf{b}}$ produces a sound and complete type-checker for bijective PTS's with Cartesian products. This includes all bijective PTS with $\boldsymbol{R}_\times \subseteq \{(s, s, s) \mid s \in \boldsymbol{S}\}$, the most sensible choices for $\boldsymbol{R}_\times$.

APPENDICES : proofs of equivalences

Most proofs are uneventful proofs by induction on derivations. The most interesting parts are in the proofs of lemmas B.1 and B.6, the cases where the last step is the application rule.

A Properties of PTS

Convention α-convertible terms are identified.

The following lemmas will be needed. With the exception of lemma A.7 they are all basic properties of PTS given in [Bar92].

Lemma A.1. [subject reduction for $\vdash : SR_\vdash$]
If $\Gamma \vdash b : B$ *and* $b \twoheadrightarrow_\beta b'$, *then* $\Gamma \vdash b' : B$.

Lemma A.2. [start lemma for $\vdash$]
(i) If $\Gamma \vdash b : B$ *and* $(s : s') \in \boldsymbol{A}$, *then* $\Gamma \vdash s : s'$.
(ii) If $\Gamma, x : A, \Gamma' \vdash b : B$, *then* $\Gamma, x : A, \Gamma' \vdash x : A$.

Lemma A.3. [Π-generation lemma for $\vdash$]
Suppose $\Gamma \vdash (\Pi x : A.\ B) : s_3$.
Then $\Gamma \vdash A : s_1$ *and* $\Gamma, x : A \vdash B : s_2$ *for some* $(s_1, s_2, s_3) \in \boldsymbol{R}$.

Lemma A.4. [sort-generation lemma for $\vdash$]
Suppose $\Gamma \vdash s : A$.
Then $A \simeq s'$ *for some* $(s : s') \in \boldsymbol{A}$.

Lemma A.5. [substitution lemma for $\vdash$]
Suppose $\Gamma, x : C, \Gamma' \vdash b : B$ *and* $\Gamma \vdash c : C$.
Then $\Gamma, \Gamma'[x := c] \vdash b[x := c] : B[x := c]$.

Lemma A.6. [unicity of types for $\vdash : UT_\vdash$]
For functional PTS: if $\Gamma \vdash b : B$ *and* $\Gamma \vdash b : B'$, *then* $B \simeq B'$.

Lemma A.7.
Suppose $A' \simeq A$ *and* $\Gamma \vdash A' : C$.
Then $(A : s) \notin \boldsymbol{A}_\top \setminus \boldsymbol{A}$.

Proof. Suppose towards a contradiction that $(A : s) \in \boldsymbol{A}_\top \setminus \boldsymbol{A}$ (so $s = \top$).
Then A is a sort, so $A' \twoheadrightarrow_\beta A$, and by $SR_\vdash$ it follows from $\Gamma \vdash A' : C$ that $\Gamma \vdash A : C$.
By the lemma A.4 there then is a sort s' such that $s' \simeq C$ and $(A : s') \in \boldsymbol{A}$.
But $(A : s) \in \boldsymbol{A}_\top \setminus \boldsymbol{A}$, and we have a contradiction. □

B Theorem 3.7 : $\vdash \iff \Vdash_{\mathrm{b}}$

We now prove theorem 3.7: for all bijective PTS

$$\begin{array}{lll} (3.7.1) & \Gamma \vdash b : B & \Rightarrow \quad (\exists s \in \mathcal{S}_{\mathsf{T}}.\ \tilde{\Gamma} \Vdash_{\mathrm{b}} b : B : s) \\ (3.7.2) & \Delta \Vdash_{\mathrm{b}} b : B : s & \Rightarrow \quad |\Delta| \vdash b : B \ \wedge\ (\ |\Delta| \vdash B : s \ \vee\ (B : s) \in \mathcal{A}_{\mathsf{T}} \backslash \mathcal{A}\ ) \end{array}$$

where $|\Delta|$ and $\tilde{\Gamma}$ are defined as in theorem 3.4.

The first part follows from lemmas B.6 and B.7; the second part is proved in the next lemma.

Lemma B.1. *For all bijective PTS*
if $\Delta \Vdash_{\mathrm{b}} b : B : s$ *then* *(i)* $|\Delta| \vdash b : B$, *and*
(ii) $|\Delta| \vdash B : s \ \vee\ (B : s) \in \mathcal{A}_{\mathsf{T}} \backslash \mathcal{A}$.

Proof. By induction on the derivation of $\Delta \Vdash_{\mathrm{b}} b : B : s$. Last step:
Axiom. The last step in the derivation in $\epsilon \Vdash_{\mathrm{b}} s : s' : \mathcal{A}_{\mathsf{T}}(s')$ where $s : s' \in \mathcal{A}$.
To prove : (i) $\epsilon \vdash s : s'$
(ii) $\epsilon \vdash s : \mathcal{A}_{\mathsf{T}}(s') \ \vee\ (s' : \mathcal{A}_{\mathsf{T}}(s')) \in \mathcal{A}_{\mathsf{T}} \backslash \mathcal{A}$
$s : s' \in \mathcal{A}$, so $\epsilon \vdash s : s'$.
If $(s' : \mathcal{A}_{\mathsf{T}}(s')) \notin \mathcal{A}_{\mathsf{T}} \backslash \mathcal{A}$ then $(s' : \mathcal{A}_{\mathsf{T}}(s')) \in \mathcal{A}$ and then $\epsilon \vdash s' : \mathcal{A}_{\mathsf{T}}(s')$
Variable. The last step in the derivation is

$$\frac{\Delta \Vdash_{\mathrm{b}} A : s_1 : s_1'}{\Delta, x : A : s_1 \Vdash_{\mathrm{b}} x : A : s_1}$$

To prove : (i) $|\Delta|, x : A \vdash x : A$
(ii) $|\Delta|, x : A \vdash A : s_1 \ \vee\ (A : s_1) \in \mathcal{A}_{\mathsf{T}} \backslash \mathcal{A}$
By the IH $|\Delta| \vdash A : s_1$. The $\vdash$-variable rule gives (i):

$$\frac{|\Delta| \vdash A : s_1}{|\Delta|, x : A \vdash x : A}$$

The $\vdash$-weakening rule gives (ii):

$$\frac{|\Delta| \vdash A : s_1 \quad |\Delta| \vdash A : s_1}{|\Delta|, x : A \vdash A : s_1}$$

Weakening. The last step in the derivation is

$$\frac{\Delta \Vdash_{\mathrm{b}} b : B : s \quad \Delta \Vdash_{\mathrm{b}} A : s_1 : s_1'}{\Delta, x : A : s_1 \Vdash_{\mathrm{b}} b : B : s}$$

To prove : (i) $|\Delta|, x : A \vdash b : B$
(ii) $|\Delta|, x : A \vdash B : s \ \vee\ (B : s) \in \mathcal{A}_{\mathsf{T}} \backslash \mathcal{A}$
By the IH $|\Delta| \vdash A : s_1$ and $|\Delta| \vdash b : B$ so the $\vdash$-weakening rule gives (i) :

$$\frac{|\Delta| \vdash b : B \quad |\Delta| \vdash A : s_1}{|\Delta|, x : A \vdash b : B}$$

By the IH $|\Delta| \vdash A : s_1$ and $(|\Delta| \vdash B : s \vee (B : s) \in \mathcal{A}_{\mathsf{T}} \backslash \mathcal{A})$.
If $(B : s) \notin \mathcal{A}_{\mathsf{T}} \backslash \mathcal{A}$ then $|\Delta| \vdash B : s$ and the $\vdash$-weakening rule gives (ii)

$$\frac{|\Delta| \vdash B : s \quad |\Delta| \vdash A : s_1}{|\Delta|, x : A \vdash B : s}$$

Conversion. The last step in the derivation is

$$\frac{\Delta \vdash_{\mathrm{b}} b : B : s \quad \Delta \vdash_{\mathrm{b}} B' : s : s' \quad B \simeq B'}{\Delta \vdash_{\mathrm{b}} b : B' : s}$$

To prove : (i) $|\Delta| \vdash b : B$
(ii) $|\Delta| \vdash B' : s' \ \vee \ (B' : s') \in \boldsymbol{A}_{\mathsf{T}} \backslash \boldsymbol{A}$
By the IH $|\Delta| \vdash B' : s$, so (ii).
By the IH $|\Delta| \vdash b : B$ and $|\Delta| \vdash B' : s$, so the $\vdash$-conversion rule gives (i):

$$\frac{|\Delta| \vdash b : B \quad |\Delta| \vdash B' : s}{|\Delta| \vdash b : B'}$$

Formation. The last step in the derivation is

$$\frac{\Delta \vdash_{\mathrm{b}} A : s_1 : s_1' \quad \Delta, x : A : s_1 \vdash_{\mathrm{b}} B : s_2 : s_2'}{\Delta \vdash_{\mathrm{b}} (\Pi x : A.\ B) : s_3 : \boldsymbol{A}_{\mathsf{T}}(s_3)} \quad (s_1, s_2, s_3) \in \boldsymbol{R}$$

To prove : (i) $|\Delta| \vdash (\Pi x : A.\ B) : s_3$
(ii) $|\Delta| \vdash s_3 : \boldsymbol{A}_{\mathsf{T}}(s_3) \ \vee \ (s_3 : \boldsymbol{A}_{\mathsf{T}}(s_3)) \in \boldsymbol{A}_{\mathsf{T}} \backslash \boldsymbol{A}$
By the IH $|\Delta| \vdash A : s_1$ and $|\Delta|, x : A \vdash B : s_2$, so the $\vdash$-formation rule gives (i):

$$\frac{|\Delta| \vdash A : s_1 \quad |\Delta|, x : A \vdash B : s_2}{|\Delta| \vdash (\Pi x : A.B) : s_3}$$

$(s_3 : \boldsymbol{A}_{\mathsf{T}}(s_3)) \in \boldsymbol{A}$ or $(s_3 : \boldsymbol{A}_{\mathsf{T}}(s_3)) \in \boldsymbol{A}_{\mathsf{T}} \backslash \boldsymbol{A}$.
If $(s_3 : \boldsymbol{A}_{\mathsf{T}}(s_3)) \in \boldsymbol{A}_{\mathsf{T}} \backslash \boldsymbol{A}$ then (ii).
If $(s_3 : \boldsymbol{A}_{\mathsf{T}}(s_3)) \in \boldsymbol{A}$ then by the $\vdash$-start lemma(A.2) $|\Delta| \vdash s_3 : \boldsymbol{A}_{\mathsf{T}}(s_3)$ and hence (ii).
Abstraction. The last step in the derivation is

$$\frac{\Delta \vdash_{\mathrm{b}} A : s_1 : s_1' \quad \Delta, x : A : s_1 \vdash_{\mathrm{b}} b : B : s_2}{\Delta \vdash_{\mathrm{b}} (\lambda x : A.\ b) : (\Pi x : A.\ B) : s_3} \quad (s_1, s_2, s_3) \in \boldsymbol{R}$$

Clearly $((\Pi x : A.\ B) : s_3) \notin \boldsymbol{A}_{\mathsf{T}} \backslash \boldsymbol{A}$, so to prove : (i) $|\Delta| \vdash (\lambda x : A.\ b) : (\Pi x : A.\ B)$
(ii) $|\Delta| \vdash (\Pi x : A.\ B) : s_3$
By the IH $|\Delta|, x : A \vdash b : B$ and $|\Delta| \vdash A : s_1$.
Also by the IH $|\Delta|, x : A \vdash B : s_2 \ \vee \ (B : s_2) \in \boldsymbol{A}_{\mathsf{T}} \backslash \boldsymbol{A}$. If $(B : s_2) \in \boldsymbol{A}_{\mathsf{T}} \backslash \boldsymbol{A}$ then $s_2 \equiv \top$. But $(s_1, s_2, s_3) \in \boldsymbol{R} \subseteq \boldsymbol{A}^3$, so $s_2 \not\equiv \top$. Hence $|\Delta|, x : A \vdash B : s_2$.
The $\vdash$ formation rule gives (i):

$$\frac{|\Delta| \vdash A : s_1 \quad |\Delta|, x : A \vdash B : s_2}{|\Delta| \vdash (\Pi x : A.\ B) : s_3}$$

and then the $\vdash$-abstraction rule gives (ii):

$$\frac{|\Delta|, x : A \vdash b : B \quad |\Delta| \vdash (\Pi x : A.\ B) : s_3}{|\Delta| \vdash (\lambda x : A.\ b) : (\Pi x : A.\ B)}$$

Application. The last step in the derivation is

$$\frac{\Delta \vdash_b b : (\Pi x{:}A.\ B) : s_3 \qquad \Delta \vdash_b a : A : s_1}{\Delta \vdash_b ba : B[x := a] : s_2}$$

for some $(s_1, s_2, s_3) \in \boldsymbol{R}$.
To prove : (i) $|\Delta| \vdash ba : B[x := a]$
(ii) $|\Delta| \vdash B[x := a] : s_2 \ \vee\ (B[x := a] : s_2) \in \boldsymbol{A_T \backslash A}$
By the IH

$$|\Delta| \vdash a : A \tag{1}$$
$$|\Delta| \vdash b : (\Pi x{:}A.\ B) \tag{2}$$

so the $\vdash$ application rule gives (i):

$$\frac{|\Delta| \vdash b : (\Pi x{:}A.\ B) \qquad |\Delta| \vdash a : A}{|\Delta| \vdash ba : B[x := a]}$$

By the IH also

$$|\Delta| \vdash A : s_1 \vee (A : s_1) \in \boldsymbol{A_T \backslash A} \tag{3}$$
$$|\Delta| \vdash (\Pi x{:}A.\ B) : s_3 \tag{4}$$

since clearly $((\Pi x{:}A.\ B) : s_3) \notin \boldsymbol{A_T \backslash A}$. Then by the Π-generation lemma for $\vdash$(A.3) it follows from (4) that

$$|\Delta| \vdash A : t_1 \tag{5}$$
$$|\Delta|, x : A \vdash B : t_2 \tag{6}$$

for some $(t_1, t_2, s_3) \in \boldsymbol{R}$. By lemma A.7 it follows from (3) and (5) that $|\Delta| \vdash A : s_1$. By $UT_{\vdash}$ then $s_1 \equiv t_1$, and since the PTS is bijective, $s_2 \equiv t_2$.

Finally, by the $\vdash$-substitution lemma(A.5) it follows from (1) and (6) that $|\Delta| \vdash B[x := a] : s_2$, and hence (ii). □

To prove (3.7.1) – "$\vdash \Rightarrow \vdash_b$" – we need the Π-generation lemma for $\vdash_b$ (lemma B.5) and unicity of types for $\vdash_b$.

Lemma B.2. [unicity of types for $\vdash_b$:$UT_{\vdash_b}$]
For bijective PTS :
(i) if $\Delta \vdash_b b : B : s$ *and* $\Delta \vdash_b b : B' : s'$ *then* $B \simeq B'$ *and* $s \equiv s'$.
(ii) if $\Delta \vdash_b b : B : s$ *and* $\Delta \vdash_b B : S : s'$ *then* $s \simeq S$.

Proof. (i) Suppose $\Delta \vdash_b b : B : s$ and $\Delta \vdash_b b : B' : s'$.
Then by the previous lemma $|\Delta| \vdash b : B$ and $|\Delta| \vdash b : B'$, so by $UT_{\vdash}$ $B \simeq B'$.
Also, by the previous lemma

$$|\Delta| \vdash B : s \ \vee\ (B : s) \in \boldsymbol{A_T \backslash A}$$
$$|\Delta| \vdash B' : s' \ \vee\ (B' : s') \in \boldsymbol{A_T \backslash A}$$

If $|\Delta| \vdash B : s$, then by lemma A.7 $|\Delta| \vdash B' : s'$ and by $SR_{\vdash}$ and $UT_{\vdash}$ $s \equiv s'$.
In the same way, if $|\Delta| \vdash B' : s'$, then also $s \equiv s'$.
Finally, if both $(B : s) \in \boldsymbol{A_T \backslash A}$ and $(B' : s') \in \boldsymbol{A_T \backslash A}$ then $s \equiv s' \equiv \top$.
(ii) Suppose $\Delta \vdash_b b : B : s$ and $\Delta \vdash_b B : S : s'$.
Then by the previous lemma

$$|\Delta| \vdash B : s \ \vee\ (B : s) \in \boldsymbol{A_T \backslash A} \tag{1}$$
$$|\Delta| \vdash B : S \tag{2}$$

It then follows from lemma A.7 that $|\Delta| \vdash B : s$, and hence by $UT_{\vdash}$ $s \simeq S$. □

For the Π-generation lemma the thinning lemma is required, which in turn requires the following start lemma.

Lemma B.3. [start lemma for $\vdash_b$]
If $\Delta \vdash_b b : B : s$ *and* $(s : s') \in \boldsymbol{A}$, *then* $\Delta \vdash_b s : s' : \boldsymbol{A}_T(s')$.
If $\Delta, x:A:s, \Delta' \vdash_b b : B : s$ *then* $\Delta, x:A:s, \Delta' \vdash_b x : A : s$.

Proof. Induction on the derivation of $\Delta \vdash_b b : B : s$ and $\Delta, x:A:s, \Delta' \vdash_b b : B : s$. □

Lemma B.4. [thinning lemma for $\vdash_b$]
Suppose $\Delta' \vdash_b b : B : s$, $\Delta' \subseteq \Delta$ *and* $\Delta \vdash_b a : A : s'$.
Then $\Delta \vdash_b b : B : s$.

Proof. Induction on the derivation of $\Delta \vdash_b a : A : s$. The start lemma is used if the last rule applied is the axiom or variable rule. □

Lemma B.5. [Π-generation lemma for $\vdash_b$]
Let $\Delta \vdash_b (\Pi x:A.\ B) : s_3 : s_3'$. *Then there are* s_1, s_1', s_2 *and* s_2' *such that* $(s_1, s_2, s_3) \in \boldsymbol{R}$, $\Delta \vdash_b A : s_1 : s_1'$ *and* $\Delta, x : A : s_1 \vdash_b B : s_2 : s_2'$.

Proof. Any derivation of $\Delta \vdash_b (\Pi x:A.\ B) : s_3 : s_3'$ ends with the formation rule followed by zero or more uses of the conversion rule and/or the weakening rule:

$$\left.\begin{array}{c} \vdots \\ \dfrac{\Delta', x : A : s_1 \vdash_b B : s_2 : s_2' \quad \Delta' \vdash_b A : s_1 : s_1'}{\Delta' \vdash_b (\Pi x:A.\ B) : s : s'} \\ \vdots \\ \Delta \vdash_b (\Pi x:A.\ B) : s_3 : s_3' \end{array}\right\} \quad (\textit{weakening/conversion})$$

where $(s_1, s_2, s) \in \boldsymbol{R}$.

The weakening rule does not affect the type s of $(\Pi x : A.\ B)$, and the conversion rule only β-converts it. So $s_3 \simeq s$, and since s and s_3 are both sorts, $s_3 \equiv s$.

$\Delta' \subseteq \Delta$, so by the thinning lemma for $\vdash_b$: $\Delta, x : A : s_1 \vdash_b B : s_2 : s_2'$ and $\Delta \vdash_b A : s_1 : s_1'$. □

To prove (3.7.1) : for all bijective PTS

$$\Gamma \vdash b : B \ \Rightarrow\ (\exists s \in \boldsymbol{S}_T.\ \tilde{\Gamma} \vdash_b b : B : s)$$

where $\tilde{\Gamma}$ is defined by $\begin{cases} \tilde{\epsilon} &= \epsilon \\ \widetilde{\Gamma, x:A} &= \tilde{\Gamma}, x:A:s \quad \text{if } \Gamma \vdash A : s \end{cases}$

we first prove

Lemma B.6. *For bijective PTS*

$$\Gamma \vdash b : B \ \Rightarrow\ (\exists s \in \boldsymbol{S}_T.\ \overline{\Gamma} \vdash_b b : B : s)$$

where $\overline{\Gamma}$ *is defined by* $\begin{cases} \overline{\epsilon} &= \epsilon \\ \overline{\Gamma, x:A} &= \overline{\Gamma}, x : A : s \quad \textit{if } \overline{\Gamma} \vdash_b A : s : s' \end{cases}$.

By $UT_{\vdash_b}$(lemma B.2) this defines a unique $\overline{\Gamma}$. We have to use $\overline{\Gamma}$ instead of $\tilde{\Gamma}$ as defined in theorem 3.4 for the proof to run smoothly. In the next lemma lemmas B.1 and B.6 will be used to prove that $\tilde{\Gamma}$ and $\overline{\Gamma}$ are the same.

Proof. By induction on the derivation of $\Gamma \vdash b : B$. Last step:
Axiom. So $\epsilon \vdash s : s'$, where $s : s' \in \boldsymbol{A}$. Then by the $\vdash_{\mathrm{b}}$-axiom rule $\overline{\epsilon} \vdash_{\mathrm{b}} s : s' : \boldsymbol{A}_{\mathsf{T}}(s')$.
Variable. The last step in the derivation is

$$\frac{\Gamma \vdash A : s_1}{\Gamma, x : A \vdash x : A}$$

To prove : $(\exists s \in \boldsymbol{S}_{\mathsf{T}}.\ \overline{\Gamma, x : A} \vdash_{\mathrm{b}} x : A : s)$.
By the IH there is a s_1' such that $\overline{\Gamma} \vdash_{\mathrm{b}} A : s_1 : s_1'$.
Then $\overline{\Gamma, x : A} \equiv \overline{\Gamma}, x : A : s_1$, and the $\vdash_{\mathrm{b}}$-variable rule gives

$$\frac{\overline{\Gamma} \vdash_{\mathrm{b}} A : s_1 : s_1'}{\overline{\Gamma}, x : A : s_1 \vdash_{\mathrm{b}} x : A : s_1}$$

Weakening. The last step in the derivation is

$$\frac{\Gamma \vdash b : B \quad \Gamma \vdash A : s_1}{\Gamma, x : A \vdash b : B}$$

To prove : $(\exists s \in \boldsymbol{S}_{\mathsf{T}}.\ \overline{\Gamma, x : A} \vdash_{\mathrm{b}} b : B : s)$.
By the IH there are s and s_1' such that $\overline{\Gamma} \vdash_{\mathrm{b}} b : B : s$ and $\overline{\Gamma} \vdash_{\mathrm{b}} A : s_1 : s_1'$.
Then $\overline{\Gamma, x : A} \equiv \overline{\Gamma}, x : A : s_1$, and the $\vdash_{\mathrm{b}}$-weakening rule gives

$$\frac{\overline{\Gamma} \vdash_{\mathrm{b}} b : B : s \quad \overline{\Gamma} \vdash_{\mathrm{b}} A : s_1 : s_1'}{\overline{\Gamma}, x : A : s_1 \vdash_{\mathrm{b}} b : B : s}$$

Conversion. The last step in the derivation is

$$\frac{\Gamma \vdash b : B \quad \Gamma \vdash B' : s' \quad B \simeq B'}{\Gamma \vdash b : B'}$$

To prove : $(\exists s \in \boldsymbol{S}_{\mathsf{T}}.\ \overline{\Gamma} \vdash_{\mathrm{b}} b : B' : s)$.
By the IH there are s and s'' such that $\overline{\Gamma} \vdash_{\mathrm{b}} b : B : s$ and $\overline{\Gamma} \vdash_{\mathrm{b}} B' : s' : s''$. To use the $\vdash_{\mathrm{b}}$-conversion rule we have to prove $s' \equiv s$.
By lemma B.1 $(\Gamma \vdash B : s \ \vee\ (B : s) \in \boldsymbol{A}_{\mathsf{T}} \backslash \boldsymbol{A})$. Since also $\Gamma \vdash B' : s'$, it follows by lemma A.7 that $\Gamma \vdash B : s$. Then by $SR_{\vdash}$(lemma A.1) a common reduct of B and B' has both type s and s', so by $UT_{\vdash}$(lemma A.6) $s' \equiv s$.
The $\vdash_{\mathrm{b}}$-conversion rule now gives

$$\frac{\overline{\Gamma} \vdash_{\mathrm{b}} b : B : s \quad \overline{\Gamma} \vdash_{\mathrm{b}} B' : s : s'' \quad B \simeq B'}{\overline{\Gamma} \vdash_{\mathrm{b}} b : B' : s}$$

Formation. The last step in the derivation is

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x : A.B) : s_3} \quad (s_1, s_2, s_3) \in \boldsymbol{R}$$

To prove : $(\exists s \in \boldsymbol{S}_{\mathsf{T}}.\ \overline{\Gamma} \vdash_{\mathrm{b}} (\Pi x : A.B) : s_3 : s)$.
By the IH $\overline{\Gamma} \vdash_{\mathrm{b}} A : s_1 : s_1'$ and $\overline{\Gamma, x : A} \vdash_{\mathrm{b}} B : s_2 : s_2'$ for some s_1' and s_2'.
Then $\overline{\Gamma, x : A} \equiv \overline{\Gamma}, x : A : s_1$, and the $\vdash_{\mathrm{b}}$-formation rule gives

$$\frac{\overline{\Gamma} \vdash_{\mathrm{b}} A : s_1 : s_1' \quad \overline{\Gamma}, x : A : s_1 \vdash_{\mathrm{b}} B : s_2 : s_2'}{\overline{\Gamma} \vdash_{\mathrm{b}} (\Pi x : A.\ B) : s_3 : \boldsymbol{A}_{\mathsf{T}}(s_3)}$$

Abstraction. The last step in the derivation is

$$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x : A.\ B) : s_3}{\Gamma \vdash (\lambda x : A.\ b) : (\Pi x : A.\ B)}$$

To prove : $(\exists s \in \boldsymbol{S_T}.\ \overline{\Gamma} \Vdash_{\mathrm{b}} (\lambda x : A.\ b) : (\Pi x : A.\ B) : s)$.

By the IH there are s_2 and s_3' such that

$$\overline{\Gamma, x : A} \Vdash_{\mathrm{b}} b : B : s_2 \tag{1}$$

$$\overline{\Gamma} \Vdash_{\mathrm{b}} (\Pi x : A.\ B) : s_3 : s_3' \tag{2}$$

By the Π-generation lemma for $\Vdash_{\mathrm{b}}$ (lemma B.5) it follows from (2) that there are s_1, s_1', t_2 and t_2' with $(s_1, t_2, s_3) \in \boldsymbol{R}$ such that

$$\overline{\Gamma} \Vdash_{\mathrm{b}} A : s_1 : s_1' \tag{3}$$

$$\overline{\Gamma}, x : A : s_1 \Vdash_{\mathrm{b}} B : t_2 : t_2' \tag{4}$$

From (3) it follows that $\overline{\Gamma, x : A} \equiv \overline{\Gamma}, x : A : s_1$.
By $UT_{\Vdash_{\mathrm{b}}}$(lemma B.2) it follows from (1) and (4) that $t_2 \equiv s_2$.
Now the $\Vdash_{\mathrm{b}}$-abstraction rule gives

$$\frac{\overline{\Gamma} \Vdash_{\mathrm{b}} A : s_1 : s_1' \quad \overline{\Gamma}, x : A : s_1 \Vdash_{\mathrm{b}} b : B : s_2}{\overline{\Gamma} \Vdash_{\mathrm{b}} (\lambda x : A.\ b) : (\Pi x : A.\ B) : s_3}$$

Application. The last step in the derivation is

$$\frac{\Gamma \vdash b : (\Pi x : A.\ B) \quad \Gamma \vdash a : A}{\Gamma \vdash ba : B[x := a]}$$

To prove : $(\exists s \in \boldsymbol{S_T}.\ \overline{\Gamma} \Vdash_{\mathrm{b}} ba : B[x := a] : s)$.

By the IH there are s_3 and s_1 such that

$$\overline{\Gamma} \Vdash_{\mathrm{b}} a : A : s_1 \tag{1}$$

$$\overline{\Gamma} \Vdash_{\mathrm{b}} b : (\Pi x : A.\ B) : s_3 \tag{2}$$

If there is an s_2 such that $(s_1, s_2, s_3) \in \boldsymbol{R}$, then we can use the $\Vdash_{\mathrm{b}}$-application rule to derive $\overline{\Gamma} \Vdash_{\mathrm{b}} ba : B[x := a] : s_2$ from (1) and (2).

So, to prove : $\exists_{s_2}(s_1, s_2, s_3) \in \boldsymbol{R}$.

By lemma B.1 ($\Vdash_{\mathrm{b}} \Rightarrow \vdash$) it follows from (1) and (2) that

$$\Gamma \vdash A : s_1 \ \vee \ (A : s_1) \in \boldsymbol{A_T} \backslash \boldsymbol{A} \tag{3}$$

$$\Gamma \vdash (\Pi x : A.\ B) : s_3 \tag{4}$$

since clearly $((\Pi x : A.\ B) : s_3) \notin \boldsymbol{A_T} \backslash \boldsymbol{A}$.

By the Π-generation lemma for $\vdash$(A.3) it follows from (4) that

$$\Gamma \vdash A : t_1 \tag{5}$$

$$\Gamma, x : A \vdash B : s_2 \tag{6}$$

for some $(t_1, s_2, s_3) \in \boldsymbol{R}$. If we can prove $s_1 \equiv t_1$ then we are done.

By lemma A.7 it follows from (3) and (5) that $\Gamma \vdash A : s_1$. By $UT_{\vdash}$(lemma A.6) then $t_1 \equiv s_1$.

□

The two ways to extend $\vdash$-contexts to $\vdash_{\mathrm{b}}$-contexts defined in theorem 3.4 and the previous lemma are the same:

Lemma B.7. $\overline{\Gamma} \equiv \tilde{\Gamma}$

Proof. Induction on Γ. The base case – $\Gamma \equiv \epsilon$ – is trivial.

Suppose $\Gamma \equiv \Gamma', x : A$. Then $\widetilde{\Gamma', x : A} \equiv \tilde{\Gamma}', x : A : s$ if $\Gamma \vdash A : s$

$\overline{\Gamma', x : A} \equiv \overline{\Gamma}', x : A : s$ if $\exists_{s'} \overline{\Gamma} \vdash_{\mathrm{b}} A : s : s'$

By the IH $\overline{\Gamma} \equiv \tilde{\Gamma}$, and by lemmas B.1 and B.6

$$\Gamma \vdash A : s \iff \exists_{s'} \overline{\Gamma} \vdash_{\mathrm{b}} A : s : s'$$

□

C Theorem 4.4 : $\vdash_b \Longleftrightarrow \vdash_{bsd}$

We now prove theorem 4.2: for all bijective PTS

$$\Delta \vdash_{bsd} b : B : s \ \wedge \ WF_{bsd}\Delta \iff \Delta \vdash_b b : B : s$$

Lemma C.1. *For bijective PTS*

$$\Delta \vdash_b b : B : s \ \Rightarrow \ (\exists B' \simeq B.\ \Delta \vdash_{bsd} b : B' : s)$$

Proof. By induction on the derivation of $\Delta \vdash_b b : B : s$.
Last step:
Axiom, Variable. The conclusions of the $\vdash_b$-axiom and variable rule are instances of the corresponding $\vdash_{bsd}$-inference rules, so the induction step is trivial.
Conversion. The last step in the derivation is

$$\frac{\Delta \vdash_b b : B : s \quad \Delta \vdash_b B' : s : s' \quad B \simeq B'}{\Delta \vdash_b b : B' : s}$$

To prove : $\exists B'' \simeq B'.\ \Delta \vdash_{bsd} b : B'' : s$.
By IH($\Delta \vdash_b b : B : s$) there is a $B'' \simeq B$ such that $\Delta \vdash_{bsd} b : B'' : s$.
But then also $B'' \simeq B'$, so we are done.
Formation. The last step in the derivation is

$$\frac{\Delta \vdash_b A : s_1 : s_1' \quad \Delta, x : A : s_1 \vdash_b B : s_2 : s_2'}{\Delta \vdash_b (\Pi x{:}A.\ B) : s_3 : \boldsymbol{A}_T(s_3)} \qquad (s_1, s_2, s_3) \in \boldsymbol{R}$$

By the IH there are $S_1 \simeq s_1$ and $S_2 \simeq s_2$ such that
$\Delta, x : A : s_1 \vdash_{bsd} B : S_2 : s_2'$ and $\Delta \vdash_{bsd} A : S_1 : s_1'$. So by the $\vdash_{bsd}$-formation rule

$$\frac{\begin{array}{ll}\Delta, x : A : s_1 \vdash_{bsd} B : S_2 : s_2' & S_2 \twoheadrightarrow_\beta s_2 \\ \Delta \vdash_{bsd} A : S_1 : s_1' & S_1 \twoheadrightarrow_\beta s_1\end{array}}{\Delta \vdash_{bsd} (\Pi x{:}A.\ B) : s_3 : \boldsymbol{A}_T(s_3)}$$

Abstraction. The last step in the derivation is

$$\frac{\Delta \vdash_b A : s_1 : s_1' \quad \Delta, x : A : s_1 \vdash_b b : B : s_2}{\Delta \vdash_b (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B) : s_3} \qquad (s_1, s_2, s_3) \in \boldsymbol{R}$$

By the IH there are $S_1 \simeq s_1$ and $B' \simeq B$ such that
$\Delta, x : A : s_1 \vdash_{bsd} b : B' : s_2$ and $\Delta \vdash_{bsd} A : S_1 : s_1'$. So by the $\vdash_{bsd}$-abstraction rule

$$\frac{\begin{array}{ll}\Delta, x : A : s_1 \vdash_{bsd} b : B' : s_2 & \\ \Delta \vdash_{bsd} A : S_1 : s_1' & S_1 \twoheadrightarrow_\beta s_1\end{array}}{\Delta \vdash_{bsd} (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B') : s_3}$$

and $(\Pi x{:}A.\ B') \simeq (\Pi x{:}A.\ B)$.
Application. The last step in the derivation is

$$\frac{\begin{array}{l}\Delta \vdash_b b : (\Pi x{:}A.\ B) : s_3 \\ \Delta \vdash_b a : A : s_1\end{array}}{\Delta \vdash_b ba : B[x := a] : s_2} \qquad (s_1, s_2, s_3) \in \boldsymbol{R}$$

By the IH there are $C \simeq (\Pi x{:}A.\ B)$ and $A'' \simeq A$ such that
$\Delta \vdash_{bsd} b : C : s_3$ and $\Delta \vdash_{bsd} a : A'' : s_1$.
$C \simeq (\Pi x{:}A.\ B)$, so there exist terms A' and B' such that $A' \simeq A$, $B' \simeq B$ and $C \twoheadrightarrow_{wh} (\Pi x{:}A'.\ B')$. Then by the $\vdash_{bsd}$-application rule

$$\frac{\begin{array}{ll}\Delta \vdash_{bsd} b : C : s_3 & C \twoheadrightarrow_{wh} (\Pi x{:}A'.\ B') \\ \Delta \vdash_{bsd} a : A'' : s_1 & A'' \simeq A'\end{array}}{\Delta \vdash_{bsd} ba : B'[x := a] : s_2}$$

and $B'[x := a] \simeq B[x := a]$. □

In order to prove

$$\Delta \vdash_{\mathrm{bsd}} b : B : s \ \wedge \ WF_{\mathrm{bsd}}\Delta \ \Rightarrow \ \Delta \vdash_{\mathrm{b}} b : B : s \tag{1}$$

we first prove

$$\Delta \vdash_{\mathrm{bsd}} b : B : s \ \wedge \ WF_{\mathrm{b}}\Delta \ \Rightarrow \ \Delta \vdash_{\mathrm{b}} b : B : s \tag{2}$$

where $WF_{\mathrm{b}}\Delta$ is defined as follows:

$$(\mathit{empty}) \qquad WF_{\mathrm{b}}\epsilon$$

$$(\mathit{weaken}) \quad \frac{WF_{\mathrm{b}}\Delta \quad \Delta \vdash_{\mathrm{b}} A : s : s'}{WF_{\mathrm{b}}\Delta, x : A : s} \ \text{if } x \text{ is } \Delta\text{-fresh}$$

Once we have proved (2), proving $WF_{\mathrm{b}}\Delta \Rightarrow WF_{\mathrm{bsd}}\Delta$ – and hence (1) – will be simple.

Before we prove (2), some work has to be done.

Lemma C.2. *For all bijective PTS*

$$\Delta \vdash_{\mathrm{b}} b : B : s \ \Rightarrow \ WF_{\mathrm{b}}\Delta$$

Proof. By induction on the derivation of $\Delta \vdash_{\mathrm{b}} b : B : s$. Last step:
Axiom. The last step in the derivation is $\epsilon \vdash_{\mathrm{b}} s : s' : \boldsymbol{A}_{\mathrm{T}}(s')$, and the empty context is well-formed : $WF_{\mathrm{b}}\epsilon$.
Variable. The last step in the derivation is

$$\frac{\Delta \vdash_{\mathrm{b}} A : s_1 : s_1'}{\Delta, x : A : s_1 \vdash_{\mathrm{b}} x : A : s_1}$$

By the IH $WF_{\mathrm{b}}\Delta$, so by WF_{b}-weakening rule

$$\frac{WF_{\mathrm{b}}\Delta \quad \Delta \vdash_{\mathrm{b}} A : s_1 : s_1'}{WF_{\mathrm{b}}\Delta, x : A : s_1}$$

Weakening. The last step in the derivation is

$$\frac{\Delta \vdash_{\mathrm{b}} b : B : s \quad \Delta \vdash_{\mathrm{b}} A : s_1 : s_1'}{\Delta, x : A : s_1 \vdash_{\mathrm{b}} b : B : s}$$

By the IH $WF_{\mathrm{b}}\Delta$, so by WF_{b}-weakening rule

$$\frac{WF_{\mathrm{b}}\Delta \quad \Delta \vdash_{\mathrm{b}} A : s_1 : s_1'}{WF_{\mathrm{b}}\Delta, x : A : s_1}$$

Conversion, Formation, Abstraction, Application In these cases the induction step is trivial: none of these rules extend the context, so $WF_{\mathrm{b}}\Delta$ follows immediately from the induction hypothesis. □

Lemma C.3. *For bijective PTS :* $\Delta \vdash_{\mathrm{b}} b : B : s \ \Rightarrow \ \overline{|\Delta|} \equiv \Delta$, *with* $\bar{}$ *as defined in lemma B.6.*
(Recall that by lemma B.7 $\overline{|\Delta|} \equiv \widetilde{|\Delta|}$, *with* $\tilde{}$ *as defined in theorem 3.4)*

Proof. In view of lemma C.2 it suffices to prove that for bijective PTS : $WF_{\mathrm{b}}\Delta \ \Rightarrow \ \overline{|\Delta|} \equiv \Delta$.
This is proved by induction on Δ. The base case – $\Delta \equiv \epsilon$ – is trivial.

Suppose $\Delta \equiv \Delta', x : A : s$. Then

$WF_{\mathrm{b}}\Delta', x : A : s$

$\Rightarrow$ {def WF_{b}}

$\exists_{s'} WF_{\mathrm{b}}\Delta' \wedge \Delta' \vdash_{\mathrm{b}} A : s : s'$

$\Rightarrow$ $\{IH\}$

$\exists_{s'} WF_{\mathrm{b}}\Delta' \wedge \Delta' \vdash_{\mathrm{b}} A : s : s' \wedge \overline{|\Delta'|} \equiv \Delta'$

$\Rightarrow$

$\exists_{s'} \Delta' \vdash_{\mathrm{b}} A : s : s' \wedge \overline{|\Delta'|} \equiv \Delta'$

$\Rightarrow$ {def $\overline{\Gamma}$ }

$\overline{|\Delta'|}, x : A \equiv \overline{|\Delta'|}, x : A : s \wedge \overline{|\Delta'|} \equiv \Delta'$

$\Rightarrow$ {def $|\ |$ }

$\overline{|\Delta', x : A : s|} \equiv \Delta', x : A : s$ □

Using this lemma, SR can now be siphoned over from $\vdash$ to $\vdash_{\mathrm{b}}$:

Lemma C.4. [type reduction for $\vdash_{\mathrm{b}}$: $TR_{\vdash_{\mathrm{b}}}$]
For bijective PTS : if $\Delta \vdash_{\mathrm{b}} b : B : s$ and $B \twoheadrightarrow_\beta B'$ then $\Delta \vdash_{\mathrm{b}} b : B' : s$.

Proof. Suppose $B \twoheadrightarrow_\beta B'$ and $\Delta \vdash_{\mathrm{b}} b : B : s$.

First we prove $\Delta \vdash_{\mathrm{b}} b : B' : s'$ for some s':

$\Delta \vdash_{\mathrm{b}} b : B : s$

$\Rightarrow$ {lemma B.1 : $\vdash_{\mathrm{b}} \Rightarrow \vdash$}

$|\Delta| \vdash b : B \ \wedge \ (|\Delta| \vdash B : s \vee B \in \boldsymbol{S})$

$\Rightarrow$ {$B \twoheadrightarrow_\beta B'$, so $B \notin \boldsymbol{S}$ }

$|\Delta| \vdash b : B \wedge |\Delta| \vdash B : s$

$\Rightarrow$ {$SR_{\vdash}$(lemma A.1)}

$|\Delta| \vdash b : B \wedge |\Delta| \vdash B' : s$

$\Rightarrow$ {$\vdash$-conversion rule}

$|\Delta| \vdash b : B'$

$\Rightarrow$ { lemma B.6: $\vdash \Rightarrow \vdash_{\mathrm{b}}$ }

$\overline{|\Delta|} \vdash_{\mathrm{b}} b : B' : s'$ for some s'

By lemma C.3, $\Delta \vdash_{\mathrm{b}} b : B : s \ \Rightarrow \ \overline{|\Delta|} \equiv \Delta$.

So it follows from $\Delta \vdash_{\mathrm{b}} b : B : s$ that $\Delta \vdash_{\mathrm{b}} b : B' : s'$ for some s'.

Finally, by $UT_{\vdash_{\mathrm{b}}}$ (lemma B.2) , part (i), it follows that $s' \equiv s$. □

Using $TR_{\vdash_{\mathrm{b}}}$ we can now prove

Lemma C.5. *For bijective PTS*

$$\Delta \vdash_{\mathrm{bsd}} b : B : s \ \wedge \ WF_{\mathrm{b}}\Delta \ \Rightarrow \ \Delta \vdash_{\mathrm{b}} b : B : s$$

Proof. By induction on the derivation of $\Delta \vdash_{\mathrm{bsd}} b : B : s$ we prove

$$\Delta \vdash_{\mathrm{bsd}} b : B : s \ \Rightarrow \ (WF_{\mathrm{b}}\Delta \ \Rightarrow \ \Delta \vdash_{\mathrm{b}} b : B : s)$$

Last step:

Axiom. The last step in the derivation is $\Delta \vdash_{\mathrm{bsd}} s : s' : \boldsymbol{A}_{\mathrm{T}}(s')$, where $s : s' \in \boldsymbol{A}$.

By the start lemma for $\vdash_{\mathrm{b}}$ (lemma B.3) and lemma C.2 : $WF_{\mathrm{b}}\Delta \ \Rightarrow \ \Delta \vdash_{\mathrm{b}} s : s' : \boldsymbol{A}_{\mathrm{T}}(s')$.

Variable. The last step in the derivation is $\Delta, x : A : s, \Delta' \vdash_{\mathrm{bsd}} x : A : s$.

By the start lemma for $\vdash_{\mathrm{b}}$ (lemma B.3) and lemma C.2 :

$WF_{\mathrm{b}}\Delta, x : A : s, \Delta' \ \Rightarrow \ \Delta, x : A : s, \Delta' \vdash_{\mathrm{b}} x : A : s$.

Formation. The last step in the derivation is

$$\frac{\Delta, x : A : s_1 \vdash_{\mathrm{bsd}} B : S_2 : s_2' \qquad S_2 \twoheadrightarrow_\beta s_2 \\ \Delta \vdash_{\mathrm{bsd}} A : S_1 : s_1' \qquad\qquad S_1 \twoheadrightarrow_\beta s_1}{\Delta \vdash_{\mathrm{bsd}} (\Pi x : A.B) : s_3 : \boldsymbol{A}_{\mathrm{T}}(s_3)} \qquad (s_1, s_2, s_3) \in \boldsymbol{R}$$

To prove : $WF_{\mathrm{b}}\Delta \Rightarrow \Delta \vdash_{\mathrm{b}} (\Pi x{:}A.\ B) : s_3 : \boldsymbol{A}_{\mathsf{T}}(s_3)$.
By the IH : $WF_{\mathrm{b}}\Delta, x : A : s_1 \Rightarrow \Delta, x : A : s_1 \vdash_{\mathrm{b}} B : S_2 : s_2'$
$WF_{\mathrm{b}}\Delta \Rightarrow \Delta \vdash_{\mathrm{b}} A : S_1 : s_1'$
Assume $WF_{\mathrm{b}}\Delta$. Then $\Delta \vdash_{\mathrm{b}} A : S_1 : s_1'$, by $TR_{\vdash_{\mathrm{b}}}$(lemma C.4) $\Delta \vdash_{\mathrm{b}} A : s_1 : s_1'$ and so $WF_{\mathrm{b}}\Delta, x : A : s_1$.
Then also by the IH : $\Delta, x : A : s_1 \vdash_{\mathrm{b}} B : S_2 : s_2'$.
By $TR_{\vdash_{\mathrm{b}}}$(lemma C.4) : $\Delta, x : A : s_1 \vdash_{\mathrm{b}} B : s_2 : s_2'$, and finally by the $\vdash_{\mathrm{b}}$-formation rule

$$\frac{\Delta, x : A : s_1 \vdash_{\mathrm{b}} B : s_2 : s_2' \quad \Delta \vdash_{\mathrm{b}} A : s_1 : s_1'}{\Delta \vdash_{\mathrm{b}} (\Pi x{:}A.\ B) : s_3 : \boldsymbol{A}_{\mathsf{T}}(s_3)}$$

Abstraction. The last step in the derivation is

$$\frac{\Delta, x : A : s_1 \vdash_{\mathrm{bsd}} b : B : s_2 \qquad \Delta \vdash_{\mathrm{bsd}} A : S_1 : s_1' \qquad S_1 \twoheadrightarrow_\beta s_1}{\Delta \vdash_{\mathrm{bsd}} (\lambda x{:}A.\ b) : (\Pi x{:}A.B) : s_3} \qquad (s_1, s_2, s_3) \in \boldsymbol{R}$$

To prove : $WF_{\mathrm{b}}\Delta \Rightarrow \Delta \vdash_{\mathrm{b}} (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B) : s_3$.
By the IH : $WF_{\mathrm{b}}\Delta, x : A : s_1 \Rightarrow \Delta, x : A : s_1 \vdash_{\mathrm{b}} b : B : s_2$
$WF_{\mathrm{b}}\Delta \Rightarrow \Delta \vdash_{\mathrm{b}} A : S_1 : s_1'$.
Assume $WF_{\mathrm{b}}\Delta$. Then $\Delta \vdash_{\mathrm{b}} A : S_1 : s_1'$, by $TR_{\vdash_{\mathrm{b}}}$(lemma C.4): $\Delta \vdash_{\mathrm{b}} A : s_1 : s_1'$ and so $WF_{\mathrm{b}}\Delta, x : A : s_1$.
Then also by the IH : $\Delta, x : A : s_1 \vdash_{\mathrm{b}} b : B : s_2$.
By $TR_{\vdash_{\mathrm{b}}}$(lemma C.4): $\Delta \vdash_{\mathrm{b}} A : s_1 : s_1'$, and then by the $\vdash_{\mathrm{b}}$-formation rule

$$\frac{\Delta, x : A : s_1 \vdash_{\mathrm{b}} b : B : s_2 \quad \Delta \vdash_{\mathrm{b}} A : s_1 : s_1'}{\Delta \vdash_{\mathrm{b}} (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B) : s_3}$$

Application. The last step in the derivation is

$$\frac{\Delta \vdash_{\mathrm{bsd}} b : C : s_3 \qquad C \twoheadrightarrow_{\mathrm{wh}} (\Pi x{:}A'.\ B) \qquad \Delta \vdash_{\mathrm{bsd}} a : A : s_1 \qquad A \simeq A'}{\Delta \vdash_{\mathrm{bsd}} ba : B[x := a] : s_2} \qquad (s_1, s_2, s_3) \in \boldsymbol{R}$$

To prove : $WF_{\mathrm{b}}\Delta \Rightarrow \Delta \vdash_{\mathrm{b}} ba : B[x := a] : s_2$.
By the IH : $WF_{\mathrm{b}}\Delta \Rightarrow \Delta \vdash_{\mathrm{b}} b : C : s_3$
$WF_{\mathrm{b}}\Delta \Rightarrow \Delta \vdash_{\mathrm{b}} a : A : s_1$
Assume $WF_{\mathrm{b}}\Delta$. Then by the IH : $\Delta \vdash_{\mathrm{b}} b : C : s_3$ and $\Delta \vdash_{\mathrm{b}} a : A : s_1$.
Let A'' be a common reduct of A and A'.
Then by $TR_{\vdash_{\mathrm{b}}}$: $\Delta \vdash_{\mathrm{b}} b : (\Pi x{:}A''.\ B) : s_3$ and $\Delta \vdash_{\mathrm{b}} a : A'' : s_1$, so by the $\vdash_{\mathrm{b}}$-application rule

$$\frac{\Delta \vdash_{\mathrm{b}} b : (\Pi x{:}A''.\ B) : s_3 \qquad \Delta \vdash_{\mathrm{b}} a : A'' : s_1}{\Delta \vdash_{\mathrm{b}} ba : B[x := a] : s_2}$$

□

Finally, we still have to prove $\Delta \vdash_{\mathrm{b}} b : B : s \Rightarrow WF_{\mathrm{bsd}}\Delta$.

Lemma C.6. *For all bijective PTS :* $\Delta \vdash_{\mathrm{b}} b : B : s \Rightarrow WF_{\mathrm{bsd}}\Delta$.

Proof. By lemma C.2 it suffices to proof $WF_{\mathrm{b}}\Delta \Rightarrow WF_{\mathrm{bsd}}\Delta$. This is proved by induction on the structure of Δ. The case $\Delta \equiv \epsilon$ is trivial. Suppose $\Delta \equiv \Delta', x : A : s$. Then

$WF_{\mathrm{b}}\Delta', x : A : s$
$\Longleftrightarrow$ {definition WF_{b}}
$\exists_{s'}\, WF_{\mathrm{b}}\Delta' \wedge \Delta' \vdash_{\mathrm{b}} A : s : s'$
$\Longrightarrow$ {IH}
$\exists_{s'}\, WF_{\mathrm{bsd}}\Delta' \wedge \Delta' \vdash_{\mathrm{b}} A : s : s'$
$\Longrightarrow$ {lemma C.1 : $\vdash_{\mathrm{b}} \Rightarrow \vdash_{\mathrm{bsd}}$}
$\exists_{S,s'}\, WF_{\mathrm{bsd}}\Delta' \wedge \Delta' \vdash_{\mathrm{bsd}} A : S : s' \wedge S \twoheadrightarrow_\beta s$
$\Longleftrightarrow$ {definition WF_{bsd}}
$WF_{\mathrm{bsd}}\Delta', x : A : s$

□

D Theorem 3.4: $\vdash \Longleftrightarrow \Vdash_f$

We now prove theorem 3.4: for all functional PTS

$$\begin{array}{rcl} \Gamma \vdash b : B & \Rightarrow & (\exists s \in \boldsymbol{S_T}.\ \tilde{\Gamma} \Vdash_f b : B : s) \\ \Delta \Vdash_f b : B : s & \Rightarrow & |\Delta| \vdash b : B \ \wedge\ (|\Delta| \vdash B : s \ \vee\ (B : s) \in \boldsymbol{A_T} \backslash \boldsymbol{A}) \end{array}$$

This is proved in the same way theorem 3.7 was proved in appendix B. Only when the application rule is involved is there any difference. This main difference is in the first half: in the proof of lemma D.7 a substitution lemma for $\Vdash_f$ (lemma D.6) is required to deal with the application rule.

Lemma D.1. *For all functional PTS*

$$\Delta \Vdash_f b : B : s \ \Rightarrow |\Delta| \vdash b : B \wedge (|\Delta| \vdash B : s \ \vee\ (B : s) \in \boldsymbol{A_T} \backslash \boldsymbol{A})$$

Proof. By induction on the derivation of $\Delta \Vdash_f b : B : s$. Only if the last step is the application rule is there any difference with the proof of lemma B.1.
Application. The last step in the derivation is

$$\frac{\begin{array}{l} \Delta \Vdash_f b : (\Pi x{:}A.\ B) : s_3 \\ \Delta \Vdash_f a : A : s_1 \\ \Delta \Vdash_f B[x := a] : s_2 : s_2' \end{array}}{\Delta \Vdash_f ba : B[x := a] : s_2}$$

To prove : (i) $|\Delta| \vdash ba : B[x := a]$
(ii) $|\Delta| \vdash B[x := a] : s_2 \ \vee\ (B[x := a] : s_2) \in \boldsymbol{A_T} \backslash \boldsymbol{A}$
By the IH $|\Delta| \vdash B[x := a] : s_2$, so (ii) holds.
By the IH $|\Delta| \vdash b : (\Pi x{:}A.\ B)$ and $|\Delta| \vdash a : A$, so the $\vdash$ application rule yields (i):

$$\frac{|\Delta| \vdash b : (\Pi x{:}A.\ B) \quad |\Delta| \vdash a : A}{|\Delta| \vdash ba : B[x := a]}$$

□

Lemma D.2. [unicity of types for $\Vdash_f$:$UT_{\Vdash_f}$]
(i) if $\Delta \Vdash_f b : B : s$ *and* $\Delta \Vdash_f b : B' : s'$
if $\Delta \Vdash_f b : B : s$ *and* $\Delta \Vdash_f B : S : s'$ *then* $s \simeq S$.

Proof. Exactly as lemma B.2. □

Lemma D.3. [start lemma for $\Vdash_f$]
If $\Delta \Vdash_f b : B : s$ *and* $(s : s') \in \boldsymbol{A}$, *then* $\Delta \Vdash_f s : s' : \boldsymbol{A_T}(s')$.
If $\Delta, x{:}A{:}s, \Delta' \Vdash_f b : B : s$ *then* $\Delta, x{:}A{:}s, \Delta' \Vdash_f x : A : s$.

Proof. Exactly as lemma B.3. □

Lemma D.4. [thinning lemma for $\Vdash_f$]
Suppose $\Delta' \Vdash_f b : B : s$, $\Delta' \subseteq \Delta$ *and* $\Delta \Vdash_f a : A : s'$.
Then $\Delta \Vdash_f b : B : s$.

Proof. Exactly as lemma B.4. □

Lemma D.5. [Π-generation lemma for $\Vdash_f$]
Let $\Delta \Vdash_f (\Pi x{:}A.\ B) : s_3 : s_3'$. *Then there are* s_1, s_1', s_2 *and* s_2' *such that*
$(s_1, s_2, s_3) \in \boldsymbol{R}$, $\Delta \Vdash_f A : s_1 : s_1'$ *and* $\Delta, x : A : s_1 \Vdash_f B : s_2 : s_2'$.

Proof. Exactly as lemma B.5. □

Lemma D.6. [substitution lemma for $\vdash_f$]
Suppose $\Delta, x : C : s, \Delta' \vdash_f b : B : s'$ *and* $\Delta \vdash_f c : C : s$.
Then $\Delta, \Delta'[x := C] \vdash_f b[x := c] : B[x := c] : s'$.

Proof. By induction on the derivation of $\Delta, x : C : s, \Delta' \vdash_f b : B : s'$. □

Lemma D.7. *For functional PTS*

$$\Gamma \vdash b : B \;\Rightarrow\; (\exists s \in \mathcal{S}_T.\ \overline{\Gamma} \vdash_f b : B : s)$$

where $\overline{\Gamma}$ *is defined by* $\begin{cases} \overline{\epsilon} & = & \epsilon \\ \overline{\Gamma, x : A} & = & \overline{\Gamma}, x : A : s \quad \text{if } \overline{\Gamma} \vdash_f A : s : s' \end{cases}$.

N.B. In lemma B.6 $\overline{\Gamma}$ is defined differently, viz. in terms of $\vdash_b$ instead of $\vdash_f$.

Proof. By induction on the derivation of $\Gamma \vdash b : B$. Only if the last step is the application rule is there any difference with the proof of lemma B.6.
Application. The last step in the derivation is

$$\frac{\Gamma \vdash b : (\Pi x : A.\ B) \quad \Gamma \vdash a : A}{\Gamma \vdash ba : B[x := a]}$$

To prove : $(\exists s \in \mathcal{S}_T.\ \overline{\Gamma} \vdash_f ba : B[x := a] : s)$.
By the IH there are s_3 and s_1 such that

$$\overline{\Gamma} \vdash_f b : (\Pi x : A.\ B) : s_3 \tag{1}$$
$$\overline{\Gamma} \vdash_f a : A : s_1 \tag{2}$$

By the Π-generation lemma for $\vdash_f$ it follows from (1) that there are t_1, t_1' ,s_2 and s_2' with $(t_1, s_2, s_3) \in \mathcal{R}$ such that

$$\overline{\Gamma} \vdash_f A : t_1 : t_1' \tag{3}$$
$$\overline{\Gamma}, x : A : t_1 \vdash_f B : s_2 : s_2' \tag{4}$$

By $UT_{\vdash_f}$ $t_1 \equiv s_1$, so by the substitution lemma for $\vdash_f$ it follows from (2) and (4) that

$$\overline{\Gamma} \vdash_f B[x := a] : s_2 : s_2'$$

Now by the $\vdash_f$ application rule

$$\frac{\begin{array}{l}\overline{\Gamma} \vdash_f b : (\Pi x : A.\ B) : s_3 \\ \overline{\Gamma} \vdash_f a : A : s_1 \\ \overline{\Gamma} \vdash_f B[x := a] : s_2 : s_2'\end{array}}{\overline{\Gamma} \vdash_f ba : B[x := a] : s_2}$$

□

Lemma D.8. *The two ways to extend* $\vdash$*-contexts to* $\vdash_f$*-contexts used in theorem 3.4 and the previous lemma are the same, i.e.* $\overline{\Gamma} \equiv \tilde{\Gamma}$.

Proof. Exactly as lemma B.7. □

References

[Bar92] H.P. Barendregt. Typed lambda calculi. In D. M. Gabbai, S. Abramsky, and T. S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1. Oxford University Press, 1992.

[vBJ92] Bert van Benthem Jutting. untitled. manuscript, 1992.

[vBJMP93] Bert van Benthem Jutting, James McKinna, and Randy Pollack. Checking algorithms for Pure Type Systems. draft.

[CH88] Thierry Coquand and Gérard Huet. The Calculus of Constructions. *Information and Computation*, 76:95–120, 1988.

[dB80] N.G. de Bruijn. A survey of the project AUTOMATH. In J.P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 579–606. Academic Press, 1980.

[Pol92] Randy Pollack. Typechecking in Pure Type Systems. In *Procs. Workshop on Types for Programs and Proofs*, pages 271–288, 1992.

In this series appeared:

91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.
91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.

91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. 25.
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.
91/29	H. Ten Eikelder R. van Geldrop	Correctness of Acceptor Schemes for Regular Languages, p. 31.
91/30	J.C.M. Baeten F.W. Vaandrager	An Algebra for Process Creation, p. 29.
91/31	H. ten Eikelder	Some algorithms to decide the equivalence of recursive types, p. 26.
91/32	P. Struik	Techniques for designing efficient parallel programs, p. 14.
91/33	W. v.d. Aalst	The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.
91/34	J. Coenen	Specifying fault tolerant programs in deontic logic, p. 15.
91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.

92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.
92/17	W.M.P. van der Aalst	Interval timed coloured Petri nets and their analysis, p. 20.
92/18	R.Nederpelt F. Kamareddine	A unified approach to Type Theory through a refined lambda-calculus, p. 30.
92/19	J.C.M.Baeten J.A.Bergstra S.A.Smolka	Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.
92/20	F.Kamareddine	Are Types for Natural Language? P. 32.
92/21	F.Kamareddine	Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.

92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.

92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.
93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.

93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness?, p. 24.