A type inference algorithm for pure type systems

Eindhoven University of Technology

Department of Mathematics and Computing Science

A Type Inference Algorithm for Pure Type Systems

by

Paula Severi

95/05

ISSN 0926-4515

editors: prof.dr. J.C.M. Baeten
prof.dr. M. Rem

Computing Science Report 95/05
Eindhoven, March 1995

A Type Inference Algorithm for Pure Type Systems

Paula Severi

University of Eindhoven - The Netherlands

Abstract

A large class of typed lambda calculi can be described in a uniform way as Pure Type Systems(PTS's). This includes for instance the second-order lambda calculus and the Calculus of Constructions. There are several implementations of PTS's such as COQ, LEGO or CONSTRUCTOR . It is important to know that these implementantions are actually correct.

In this paper we present an efficient algorithm for infering types for singly sorted Pure Type Systems and prove its correctness.

1 Introduction

For the implementations of PTS's it is important to consider the following two questions (see [Bar91]):

1. Given a context Γ and terms b and B, is it true that $\Gamma \vdash b : B$?
2. Given a context Γ and a term b, does it exist B such that $\Gamma \vdash b : B$?

These two problems are called *Type Checking* and *Typability* and are denoted as $\Gamma \vdash b : B?$ and $\Gamma \vdash b :?$ respectively .

Notice that from a solution to the second problem we can easily find a solution to the first one.

It is known that for some PTS's, e.g. $\lambda*$, these problems are undecidable.

In [vBJ93] it is shown that if a PTS is normalizing and has a finite set of sorts then these two problems are decidable. In the algorithm they construct it is necessary to compute the normal form of types and this makes it inefficient.

Most of the attempts to construct algorithms for type checking and type inference (see for example [vBJMP93]) pass through the consideration of typing rules for which the type deduction is determined by the shape of the term b and of the context Γ.

A set of rules for a typing relation $\vdash$ are called *syntax directed* if given a context Γ and a term b there exists B such that there is at most one derivation of $\Gamma \vdash b : B$.

A syntax directed set of rules defines a partial function $\Gamma, b \longmapsto B$. The algorithm to compute the corresponding function $\Gamma, b \longmapsto B$ is called a *type inference* algorithm.

As the conversion and the weakening rules can be used at any point in the derivation, it is clear that the rules for PTS's are not syntax directed.

Unfortunately for the syntax directed system presented in [vBJMP93], even though it is very natural, Completeness has not been proved. The main problem seems to be the impossibility to apply the inductive hypothesis to the type premise in the abstraction rule.

The authors of [vBJMP93] solve the problem presenting other syntax directed systems with a more liberal type premise in the abstraction rule. But in this case the new typing relations do not seem to be natural.

In [Pol93] a type inference algorithm for bijective PTS's is presented. The class of bijective PTS's includes all systems of the λ-cube and is a proper subclass of the class we study here, the class of singly sorted PTS's.

In this paper we will present an efficient type inference algorithm for singly sorted Pure Type Systems. It can briefly be described as follows:

1. Infer the type of the term in a system allowing illegal abstractions, i.e. in a system without the type premise in the abstraction rule.

2. Check – separately – if the abstractions in the term are not illegal.

For the step 1) of this algorithm we will consider Pure Type Systems without the type premise "$\Gamma \vdash (\Pi x{:}A.\ B) : s$" in the abstraction rule (PTS^w's). We study the methatheory of PTS^w's in detail.

First we prove that if a PTS is weakly normalizing then the corresponding PTS^w is weakly normalizing too.

Second we prove that the set of typable terms of a PTS and the corresponding PTS^w are the same iff the specification is a completion of itself. In other words we characterize those specifications for which the type premise in the abstraction rule is redundant.

Also we prove that for certain specifications, if a PTS is strongly normalizing then the corresponding PTS^w is strongly normalizing.

We finish this introduction by mentioning the following results (that will not be proved in this paper). The PTS^w's are closely related to Pure Type Systems with definitions (see [SP94]) and to KPTS's [1]. The following open problems are equivalent for single sorted PTS's:

- For any specification, if a PTS is β-strongly normalizing then the corresponding DPTS extended with definitions is $\beta\delta$-strongly normalizing.
- For any specification, if a PTS is β-strongly normalizing then the corresponding KPTS is βk-strongly normalizing.
- For any specification, if a PTS is β-strongly normalizing then the corresponding PTS^w is β-strongly normalizing.

2 Pure Type Systems

We define the concept of Pure Type System as in [Bar92].

Definition 2.1. The specification of Pure Type System (PTS) is a triple $S = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ such that

- $\boldsymbol{S} \subseteq \mathbf{C}$ is the set of *sorts*.
- $\boldsymbol{A} \subseteq \mathbf{C} \times \boldsymbol{S}$ is the set of *axioms*
- $\boldsymbol{R} \subseteq \boldsymbol{S} \times \boldsymbol{S} \times \boldsymbol{S}$ is the set of *rules*

Definition 2.2. The set $\mathcal{T}$ of *pseudoterms* and the set $\mathcal{C}$ of *contexts* are defined as follows:

$$\begin{array}{lcl} \mathcal{T} & ::= & V \mid \mathbf{C} \mid (\mathcal{T}\ \mathcal{T}) \mid (\lambda V{:}\mathcal{T}.\ \mathcal{T}) \mid (\Pi V{:}\mathcal{T}.\ \mathcal{T}) \\ \mathcal{C} & ::= & \epsilon \mid < \mathcal{C}, V{:}\mathcal{T} > \end{array}$$

where V is the set of variables and $\mathbf{C}$ is the set of constants.

The β-reduction is defined as usual by the rule $(\lambda x{:}A.\ a)b \rightarrow_\beta a[x := b]$. The α-equality is defined as usual and α-equal terms are identified.

[1] A KPTS is a PTS extended with the following typing rule: $\dfrac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash (K\ a\ b) : A}$ and the following reduction rule: $(K\ a\ b) \rightarrow_k a$

Definition 2.3. The PTS determined by the specification $S = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ is denoted as $\lambda S = \lambda(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ and defined by the notion of type derivation $\Gamma \vdash_{\lambda S} b : B$ (or $\Gamma \vdash b : B$) given by the following axioms and rules:

$$
\begin{array}{lcl}
(axiom) & \epsilon \vdash c : s & \text{for } (c, s) \in \boldsymbol{A} \\
(start) & \dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} & \text{where } x \text{ is } \Gamma\text{-fresh} \\
(weakening) & \dfrac{\Gamma \vdash b : B \quad \Gamma \vdash A : s}{\Gamma, x : A \vdash b : B} & \\
(formation) & \dfrac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x{:}A.B) : s_3} & \text{for } (s_1, s_2, s_3) \in \boldsymbol{R} \\
(abstraction) & \dfrac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x{:}A.\ B) : s}{\Gamma \vdash (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B)} & \\
(application) & \dfrac{\Gamma \vdash b : (\Pi x{:}A.\ B) \quad \Gamma \vdash a : A}{\Gamma \vdash (b\ \ a) : B[x := a]} & \\
(conversion) & \dfrac{\Gamma \vdash b : B \quad \Gamma \vdash B' : s \quad B =_\beta B'}{\Gamma \vdash b : B'} &
\end{array}
$$

where s ranges over sorts, i.e. $s \in \boldsymbol{S}$.

The following results are well-known (see for example [Bar92]).

Theorem 2.4. (Church Rosser for β-reduction) Let $\Gamma \in \mathcal{C}$ and $a \in \mathcal{T}$ be such that $a \rightarrow_\beta b$ and $a \rightarrow_\beta c$. Then there exists a term $d \in \mathcal{T}$ such that $b \rightarrow_\beta d$ and $c \rightarrow_\beta d$.

Theorem 2.5. (Correctness of Types) Let $\Gamma \in \mathcal{C}$ and $d, d', D \in \mathcal{T}$ be such that $\Gamma \vdash d : D$. Then $\Gamma \vdash D : s$ or $D \equiv s$.

Theorem 2.6. (Subject Reduction Theorem) Let $\Gamma \in \mathcal{C}$ and $d, d', D \in \mathcal{T}$ be such that $\Gamma \vdash d : D$. If $d \rightarrow_\beta d'$ then $\Gamma \vdash d' : D$

Definition 2.7. The specification $S = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ is called singly sorted if

1. $(c, s_1), (c, s_2) \in \boldsymbol{A}$ implies $s_1 \equiv s_2$
2. $(s_1, s_2, s_3), (s_1, s_2, s_3') \in \boldsymbol{R}$ implies $s_3 \equiv s_3'$

Theorem 2.8. (Uniqueness of Types) Let S be a singly sorted specification, $\Gamma \in \mathcal{C}$ and $a, A, B \in \mathcal{T}$ such that $\Gamma \vdash a : A$ and $\Gamma \vdash a : B$. Then $A =_\beta B$.

Definition 2.9. Let λS be a PTS. A sort s in $\boldsymbol{S}$ is called a *topsort* if there is no $s_0 \in \boldsymbol{S}$ such that $(s, s_0) \in \boldsymbol{A}$.

Definition 2.10.
The specification $S = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ is called *full* if for all $s_1, s_2 \in \boldsymbol{S}$ there exists s_3 such that $(s_1, s_2, s_3) \in \boldsymbol{R}$

Definition 2.11. The λ-*cube* is a cube of eight systems defined by the same set of sorts $\boldsymbol{S} = \{*, \Box\}$ and the same set of axioms $\boldsymbol{A} = \{(*, \Box)\}$. They differ in the set of rules $\boldsymbol{R}$.

System	$\boldsymbol{R}$			
$\lambda_\rightarrow$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
λP	$(*,*)$		$(*,\Box)$	
$\lambda P2$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$
$\lambda P\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
$\lambda P\omega = \lambda C$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

The rule (s_1, s_2) is an abbreviation for (s_1, s_2, s_2).

Note that the system λC is the Calculus of Constructions and this system is full.
All the systems of the λ-cube have only one topsort that is $\Box$.

Definition 2.12. The Calculus of Constructions extended with an infinite type hierarchy can be described by the following PTS:

$$\lambda C_\infty \quad \boxed{\begin{array}{ll} \boldsymbol{S} = & \mathbf{N} \\ \boldsymbol{A} = & \{(n, n+1) | n \in \mathbf{N}\} \\ \boldsymbol{R} = & \{(m, 0, 0) | \ m \in \mathbf{N}\} \cup \{(m, n, r) | \ m, n \in \mathbf{N} \ \& max(m, n) \leq r\} \end{array}}$$

The system λC_∞ extended with strong Σ-types and cumulativity is the system **ECC**(see [Luo89]).
We can see that λC_∞ is an extension of λC writing $*$ instead of 0, $\Box$ instead of 1.
Note that there is no topsort in λC_∞.

Definition 2.13. Let λS be a PTS. Then λS is *β-strongly normalizing* if a and A β-strongly normalize for all $a, A \in \mathcal{T}$ and $\Gamma \in \mathcal{C}$ such that $\Gamma \vdash_{\lambda S} a : A$.

The system λC_∞ and the systems of the λ-cube are β-strongly normalizing. Howewer not all PTS's are β-strongly normalizing as next example shows:

Example 2.14. The PTS $\lambda*$ determined by the specification $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ where $\boldsymbol{S} = \{*\}$, $\boldsymbol{A} = \{(*, *)\}$ and $\boldsymbol{R} = \{(*, *)\}$ is not β-strongly normalizing.

3 Pure Type Systems with Weakened Abstraction Rule

In this section we consider Pure Type Systems without the type premise $\Gamma \vdash (\Pi x{:}A.\ B) : s$ in the abstraction rule (PTS^w's or $(\lambda S)^w$). The abstraction rule for these systems will be as follows:

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

The notion of type derivation in PTS^w will be written as $\Gamma \vdash_w a : A$ or $\Gamma \vdash_{\lambda S^w} a : A$. Note that $\lambda S \subseteq \lambda S^w$, i.e. if $\Gamma \vdash_{\lambda S} a : A$ then $\Gamma \vdash_w a : A$. The following example shows that λS^w has more typable terms than λC, i.e. $\lambda C \subset \lambda C^w$.

Example 3.1. The following term is typable in λC^w:

$$A : * \vdash_{\lambda C^w} \lambda x{:}A.(* \to *) : (A \to \Box)$$

But it is not typable in λC because $A \to \Box$ does not have type.

Properties like Subject Reduction, Substitution Lemma and Strengthening for PTS^w's are easy to prove. Note that the property of Correctness of Types does not hold for PTS^w's.

3.1 Description of Toptypes

In this section we will define the notion of toptype and prove that toptypes have a very special form. This will give us an idea of the form of the 'new' terms that we are adding to λS when we do not consider the type premise of the abstraction rule.

Notation 3.2.
From now on $\Gamma \nvdash_w A : -$ will denote that A is not typable in Γ. i.e. $\nexists\, s[\Gamma \vdash_w A : s]$.

It will be important to consider the terms A such that $\Gamma \vdash_w a : A$ and $\Gamma \nvdash_w A : -$ and hence we define:

Definition 3.3. We say that A is a *toptype* if there exist Γ, a such that $\Gamma \vdash_w a : A$ and $\Gamma \not\vdash_w A : -$.

One could say that toptypes are 'types' which do not have 'type'.

Example 3.4.

a) In (λC^w) we can derive

$$A : * \vdash_{C^w} \lambda x{:}A.(* \to *) : (A \to \Box)$$

The term $(A \to \Box)$ has no type in λC^w and is a toptype.

b) In $(\lambda_{\to})^w$ we can derive

$$\vdash_{(\to)^w} (\lambda\alpha{:}*.\lambda x{:}\alpha.x) : \Pi\alpha{:}*.\alpha \to \alpha$$

The term $\Pi\alpha{:}*.\alpha \to \alpha$ has no type in $(\lambda \to)^w$ and is a toptype.

Notice that the notion of toptype makes sense for both, PTS's and PTS^w's. For PTS's, the only toptypes are the topsorts. For example in λC there is only one toptype that is $\Box$.

Now we study the form of the toptypes. For systems with full specification, a toptype has the form $\Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.s_0$ with s_0 a topsort. For example, any toptype in λC^w has the form: $(\Pi x{:}A_1.\ \Pi x_2{:}A_2 \ldots \Pi x_n{:}A_n.\ \Box)$.

The description of toptypes for any PTS^w is a little more complicated: the toptype is not typable either because the main branch of the product is a topsort or because the rule we need to type the product is not in the set of rules $\boldsymbol{R}$.

Theorem 3.5. (Description Theorem)
Suppose A is a toptype, i.e. $\Gamma \vdash_w a : A$ and $\Gamma \not\vdash_w A : -$. Then the following two properties hold:

a) $A \equiv \Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.B$ where

$$\begin{array}{ll} \text{either} & B \text{ is a topsort } s_0 \\ \text{or} & \exists s \quad \Gamma, x_1{:}A_1 \ldots x_{n-1}{:}A_{n-1}, x_n{:}A_n \vdash_w B : s \end{array}$$

b) $a \to_\beta \lambda x_1{:}A_1 \ldots \lambda x_n{:}A_n.b$ and $\Gamma, x_1{:}A_1 \ldots x_n{:}A_n \vdash_w b : B$.

Proof: By induction on the derivation of $\Gamma \vdash_w a : A$.

Q. E. D.

Corollary 3.6. (Toptypes) Let $\Gamma \vdash_w a : A$.

$$\begin{array}{lll} \Gamma \not\vdash_w A : - & \text{iff} & A \equiv \Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.B \text{ where} \\ & & \text{either } B \text{ is a topsort } s_0 \\ & & \text{or } \forall s_1, s_2 \ \left.\begin{array}{l} \Gamma, x_1{:}A_1 \ldots x_{n-1}{:}A_{n-1} \vdash_w A_n : s_1, \\ \Gamma, x_1{:}A_1 \ldots x_{n-1}{:}A_{n-1}, x_n{:}A_n \vdash_w B : s_2 \end{array}\right\} \Rightarrow \nexists s[(s_1, s_2, s) \in \boldsymbol{R}] \end{array}$$

Proof: Immediate. **Q. E. D.**

Theorem 3.7. (β-Closure of Toptypes) [2] Let S be a full or a singly sorted specification.

$$\text{If } \Gamma \vdash_w a : A,\ \Gamma \not\vdash_w A : - \text{ and } A \to_\beta A' \text{ then } \Gamma \not\vdash_w A' : -$$

Proof: It follows from the description theorem. **Q. E. D.**

Corollary 3.8. Let S be a full or singly sorted specification and $\Gamma \vdash_w b : B$.
If $\exists s \in \boldsymbol{S}[\Gamma \vdash_w B' : s]$ and $B' =_\beta B$ then $\exists s' \in \boldsymbol{S}[\Gamma \vdash_w B : s']$.

Proof: By the theorem of Church-Rosser there exists D_0 a common redex of B and B'. By the subject reduction theorem we have that $\Gamma \vdash_w D_0 : s$. By the β-closure of toptypes we have that $\Gamma \vdash_w B : s'$ for some s'. **Q. E. D.**

[2] We call this theorem β-closure of toptypes even though it is not completly right because we do not prove that A' is inhabited.

3.2 Normalization for β-reduction

In this section we prove for general PTS's that:

$$\text{If} \quad \Gamma \vdash_w a : A : s \quad \text{then} \quad \Gamma' \vdash_{\lambda S} a' : A' \quad \begin{array}{ll} \text{for some} & a', A', \Gamma' \\ \text{such that} & a \to_\beta a',\ A \to_\beta A',\ \Gamma \to_\beta \Gamma'. \end{array}$$

In the case of full or functional specifications, we have more: we define a mapping $\mathcal{C}$ such that if $\Gamma \vdash_w a : A : s$ then $\mathcal{C}(\Gamma) \vdash \mathcal{C}(a) : \mathcal{C}(A)$ and $a \to_\beta \mathcal{C}(a)$, $A \to_\beta \mathcal{C}(A)$ and $\Gamma \to_\beta \mathcal{C}(\Gamma)$.

Then we prove that weak normalization of λS implies weak normalization of λS^w.

Definition 3.9. We say that an abstraction $\lambda x{:}A.b$ is *illegal* w.r.t. the context Γ if for all D such that $\Gamma \vdash_w \lambda x{:}A.b : D$ we have that $\Gamma \not\vdash_w D : -$.

Definition 3.10. A redex $(\lambda x : A.\ b)a$ is called *an illegal redex* w.r.t. the context Γ if its abstraction $(\lambda x{:}A.\ b)$ is illegal.

We define a mapping $\mathcal{C}$ that contracts all the illegal redexes of a term.

Definition 3.11. Given $\Gamma \vdash_w a : A$ we define $\mathcal{C}_\Gamma$ on the typable terms as follows:

$$\begin{array}{rcl} \mathcal{C}_\Gamma(x) & = & x \\ \mathcal{C}_\Gamma(a\ b) & = & \begin{cases} a_0[x := \mathcal{C}_\Gamma(b)] & \text{if } \mathcal{C}_\Gamma(a) = \lambda x{:}A.a_0 \text{ is an illegal abstraction w.r.t. } \Gamma \\ (\mathcal{C}_\Gamma(a)\ \mathcal{C}_\Gamma(b)) & \text{otherwise} \end{cases} \\ \mathcal{C}_\Gamma(\lambda x{:}A.\ a) & = & (\lambda x{:}\mathcal{C}_\Gamma(A).\ \mathcal{C}_{\Gamma,x:A}(a)) \\ \mathcal{C}_\Gamma(\Pi x{:}A.\ B) & = & (\Pi x{:}\mathcal{C}_\Gamma(A).\ \mathcal{C}_{\Gamma,x:A}(B)) \end{array}$$

We write $\mathcal{C}(a)$ instead of $\mathcal{C}_\Gamma(a)$.

Lemma 3.12.

1. Suppose $\Gamma \vdash_w a : A$. Then $a \to_\beta \mathcal{C}(a)$.
2. Suppose that S is singly sorted or full. Then $\mathcal{C}(b)[x := \mathcal{C}(a)] \equiv \mathcal{C}(b[x := a])$.

Proof: They are proved by induction on the structure of the term. **Q. E. D.**

Next example shows that if the specification is not singly sorted $\mathcal{C}(b[x := a])$ may not be syntactically equal to $\mathcal{C}(b)[x := \mathcal{C}(a)]$.

Example 3.13. The following specification is not singly sorted:

$$\lambda S \quad \boxed{\begin{array}{ll} \boldsymbol{S} & 0, 1, 2 \\ \boldsymbol{A} & 0:1, 0:2, 1:2 \\ \boldsymbol{R} & (2, 2) \end{array}}$$

We take $\Gamma \equiv x : 1, z : x$, $b \equiv (\lambda y{:}x.y)z$ and $a \equiv 0$. Note that b contains illegal abstractions but $b[x := 0]$ does not. Hence $\mathcal{C}(b[x := 0]) \not\equiv \mathcal{C}(b)[x := \mathcal{C}(0)]$.

Lemma 3.14.

1. If $\Gamma \vdash_w A : s$ and $\Gamma \vdash a : A$ then $\Gamma \vdash A : s'$ for some sort s'.
2. Suppose S be a singly sorted specification.
   If $\Gamma \vdash_w A : s$ and $\Gamma \vdash a : A$ then $\Gamma \vdash A : s$

Proof:

1. By the lemma of correctness of types, we have that $\Gamma \vdash A : s'$ or $A \equiv s'$.
   If $A \equiv s'$ then $s' : s$ is an axiom.

2. It follows from the previous part.

Q. E. D.

Lemma 3.15.

1. Let S be a singly sorted specification.
   If $\Gamma \vdash_w F : (\Pi x{:}A.\ B)$, $\Gamma \vdash_w B[x := a] : s$ and $\Gamma \vdash_w a : A$ then $\exists s_2\ \ \Gamma, x{:}A \vdash_w B : s_2$.

2. Suppose S is full. If $\Gamma \vdash_w a : A$, $\Gamma \vdash_w F : (\Pi x{:}A.\ B)$ and $\Gamma \vdash_w B[x := a] : s$ then,
   $\Gamma \vdash_w (\Pi x{:}A.\ B) : s'$ for some s'.

3. If $\Gamma \not\vdash_w B[x := a] : -$ then $\Gamma \not\vdash_w (\Pi x{:}A.\ B) : -$.

Proof:

1. It is proved using the description theorem.

2. Suppose towards a contradiction that $\Pi x{:}A.\ B \equiv \Pi x_1{:}A_1.\ \ldots \Pi x_n{:}A_n.s_0$ with s_0 a topsort. Then $B[x := a] \equiv \Pi x_1 : A_1[x := a].\ \ldots \Pi x_n : A_n[x := a].s_0$. It follows from the topsort theorem that $B[x := a]$ is a toptype. This is a contradiction.

3. Suppose $\Gamma \vdash_w (\Pi x{:}A.\ B) : s'$. By the generation lemma we have that $\Gamma, x{:}A \vdash_w B : s$. By the substitution lemma we have that $\Gamma \vdash_w B[x := a] : s$.

Q. E. D.

Theorem 3.16. Let S be a full or singly sorted specification.
If $\Gamma \vdash_w a : A$ then

1. If A is not a toptype then
   $\mathcal{C}(\Gamma) \vdash_{\lambda S} \mathcal{C}(a) : \mathcal{C}(A)$.

2. If A is a toptype then

$$\mathcal{C}(\Gamma, x_1{:}A_1 \ldots x_n{:}A_n) \vdash_{\lambda S} a' : \mathcal{C}(B) \quad \text{where} \quad \begin{array}{l} A \equiv \Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.B \text{ and} \\ \mathcal{C}(a) \equiv \lambda x_1{:}\mathcal{C}(A_1) \ldots \lambda x_n{:}\mathcal{C}(A_n).a' \end{array}$$

Proof: This property is proved by induction on the derivation of $\Gamma \vdash_w a : A$.
Note that if $A \equiv s$ then $\mathcal{C}(\Gamma) \vdash_{\lambda S} \mathcal{C}(a) : s$.
We consider 3 cases:

- (abstraction) $\dfrac{\Gamma, x : A \vdash_w b : B}{\Gamma \vdash_w (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B)}$

  1. Suppose $\Gamma \vdash_w (\Pi x{:}A.\ B) : s$. By the generation lemma we have that:
     $\Gamma, x{:}A \vdash_w B : s_2$, $\Gamma \vdash_w A : s_1$ and $(s_1, s_2, s) \in \boldsymbol{R}$.
     By the IH we have that
     $$\mathcal{C}(\Gamma) \vdash \mathcal{C}(A) : s_1 \text{ and } \mathcal{C}(\Gamma, x : A) \vdash \mathcal{C}(b) : \mathcal{C}(B).$$
     If the specification is full by lemma 3.14 we have that $\mathcal{C}(\Gamma, x : A) \vdash \mathcal{C}(B) : s_2'$ and there exists s_3 such that $(s_1, s_2', s_3) \in \boldsymbol{R}$.
     If the specification is singly sorted by lemma 3.14, we have that $s_2 \equiv s_2'$ and we know that $(s_1, s_2, s) \in \boldsymbol{R}$.
     In any case $(s_1, s_2', s_3) \in \boldsymbol{R}$ for some s_3.
     We obtain the following derivation:

$$\frac{\mathcal{C}(\Gamma, x : A) \vdash \mathcal{C}(b) : \mathcal{C}(B) \quad \dfrac{\mathcal{C}(\Gamma) \vdash \mathcal{C}(A) : s_1 \quad \mathcal{C}(\Gamma, x : A) \vdash \mathcal{C}(B) : s_2'}{\mathcal{C}(\Gamma) \vdash \mathcal{C}(\Pi x{:}A.\ B) : s_3}}{\mathcal{C}(\Gamma) \vdash \mathcal{C}(\lambda x{:}A.\ b) : \mathcal{C}(\Pi x{:}A.\ B)}$$

2. Suppose $\Gamma \not\vdash_w (\Pi x{:}A.\ B) : -$. There are two possibilities:
   (a) $\Gamma, x{:}A \vdash_w B : s$.
   By the IH we have that:
   $$\mathcal{C}(\Gamma, x{:}A) \vdash_w \mathcal{C}(b) : \mathcal{C}(B)$$
   where $\mathcal{C}(\lambda x{:}A.\ b) \equiv (\lambda x{:}\mathcal{C}(A).\ \mathcal{C}(b))$
   (b) $\Gamma, x{:}A \not\vdash_w B : -$.
   It follows from the IH that
   $$\mathcal{C}(\Gamma, x{:}A, x_1{:}A_1 \ldots x_n{:}A_n) \vdash_{\lambda S} b' : \mathcal{C}(B') \quad \text{where} \quad B \equiv \Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.B', \quad \mathcal{C}(b) \equiv \lambda x_1{:}\mathcal{C}(A_1) \ldots \lambda x_n{:}\mathcal{C}(A_n).b'$$

- (application) $\dfrac{\Gamma \vdash_w b : (\Pi x{:}A.\ B) \quad \Gamma \vdash a : A}{\Gamma \vdash_w (b\ \ a) : B[x := a]}$.

1. Suppose $\Gamma \vdash_w B[x := a] : s$.
   (a) If the specification is singly sorted then there are two cases:
      i. Suppose $\Gamma \vdash_w (\Pi x{:}A.\ B) : s'$.
      By the IH we have that
      $$\mathcal{C}(\Gamma) \vdash \mathcal{C}(b) : \mathcal{C}(\Pi x{:}A.\ B)$$
      Note that $\Gamma \vdash_w A : s_1$. By the IH we have that $\mathcal{C}(\Gamma) \vdash \mathcal{C}(a) : \mathcal{C}(A)$.
      Hence we have the following derivation:
      $$\frac{\mathcal{C}(\Gamma) \vdash \mathcal{C}(b) : \mathcal{C}(\Pi x{:}A.\ B) \quad \mathcal{C}(\Gamma) \vdash \mathcal{C}(a) : \mathcal{C}(A)}{\mathcal{C}(\Gamma) \vdash \mathcal{C}(b\ \ a) : \mathcal{C}(B)[x := \mathcal{C}(a)]}$$
      By lemma 3.12 we have that $\mathcal{C}(B)[x := \mathcal{C}(a)] \equiv \mathcal{C}(B[x := a])$.
      ii. Suppose $\Gamma \not\vdash_w (\Pi x{:}A.\ B) : -$.
      Note that $\Gamma \vdash_w A : s_1$. By the IH we have that $\mathcal{C}(\Gamma) \vdash \mathcal{C}(a) : \mathcal{C}(A)$.
      By lemma 3.15 we have that $\Gamma, x : A \vdash_w B : s_2$. By the IH we have that $\mathcal{C}(\Gamma, x{:}A) \vdash b' : \mathcal{C}(B)$ with $\mathcal{C}(b) \equiv \lambda x{:}A'.b'$.
      Due to the β-closure of toptypes we have that $\mathcal{C}(b) \equiv \lambda x : A'.b'$ is an illegal abstraction and then $\mathcal{C}(b\ \ a) \equiv b'[x := \mathcal{C}(a)]$.
      By the substitution lemma,
      $$\mathcal{C}(\Gamma) \vdash b'[x := \mathcal{C}(a)] : \mathcal{C}(B)[x := \mathcal{C}(a)]$$
      By lemma 3.12, we have that $\mathcal{C}(B)[x := \mathcal{C}(a)] \equiv \mathcal{C}(B[x := a])$.
   (b) If the specification is full then by lemma 3.15 we have that $\Gamma \vdash_w (\Pi x{:}A.\ B) : s'$. The proof proceeds as in case 1(a)i.
2. Suppose $\Gamma \not\vdash_w B[x := a] : -$.
   It follows from lemma 3.15 that $\Gamma \not\vdash_w (\Pi x{:}A.\ B) : -$.
   By the description theorem we have that $(\Pi x{:}A.\ B) \equiv \Pi x{:}A.\Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.B_0$.
   By the IH we have that $\mathcal{C}(\Gamma) \vdash \mathcal{C}(a) : \mathcal{C}(A)$ and that
   $$\mathcal{C}(\Gamma, x{:}A, x_1{:}A_1, \ldots, x_n{:}A_n) \vdash_{\lambda S} b' : \mathcal{C}(B_0) \quad \text{where} \quad \mathcal{C}(b) \equiv \lambda x{:}\mathcal{C}(A).\ \lambda x_1{:}\mathcal{C}(A_1) \ldots \lambda x_n{:}\mathcal{C}(A_n).b'$$
   Due to the β-closure of toptypes we have that $\mathcal{C}(b)$ is an illegal abstraction and then the value of $\mathcal{C}(b\ a)$ is computed as follows:

$$\mathcal{C}(b\ \ a) \quad \equiv \quad \lambda x_1{:}\mathcal{C}(A_1)[x := \mathcal{C}(a)] \ldots \lambda x_n{:}\mathcal{C}(A_n)[x := \mathcal{C}(a)].b'[x := \mathcal{C}(a)]$$

By the substitution lemma we have that:

$$\mathcal{C}(\Gamma), x_1{:}\mathcal{C}(A_1)[x := \mathcal{C}(a)], \ldots, x_n{:}\mathcal{C}(A_n)[x := \mathcal{C}(a)]) \vdash_{\lambda S} b'[x := \mathcal{C}(a)] : \mathcal{C}(B_0)[x := \mathcal{C}(a)]$$

By lemma 3.12 we have that: $\mathcal{C}(B_0)[x := \mathcal{C}(a)] = \mathcal{C}(B_0[x := a])$.

- (conversion) $\dfrac{\Gamma \vdash_w b : B \quad \Gamma \vdash_w A : s \quad B =_\beta A}{\Gamma \vdash_w b : A}$

  By corollary 3.6, we have that $\Gamma \vdash_w B : s'$. By the IH we have that $\mathcal{C}(\Gamma) \vdash \mathcal{C}(b) : \mathcal{C}(B)$. By the IH we also have that $\mathcal{C}(\Gamma) \vdash \mathcal{C}(A) : s$. By conversion rule $\mathcal{C}(\Gamma) \vdash \mathcal{C}(b) : \mathcal{C}(A)$.

Q. E. D.

In [vBJ93] the set $\mathcal{T}$ is partitioned into sets $\mathcal{T}_V$ and $\mathcal{T}_S$ such that terms in $\mathcal{T}_V$ have a unique type and terms in $\mathcal{T}_S$ may have more than one type in a PTS. The same property holds for PTS^w's.

Definition 3.17.

$$\begin{array}{lcll} \mathcal{T}_V & ::= & V & \mid (\mathcal{T}_V\ \mathcal{T}_{VS}) \mid (\lambda V{:}\mathcal{T}_{VS}.\ \mathcal{T}_V) \\ \mathcal{T}_S & ::= & \mathbf{C} & \mid (\mathcal{T}_S\ \mathcal{T}_{VS}) \mid (\lambda V{:}\mathcal{T}_{VS}.\ \mathcal{T}_S) \quad \mid (\Pi V{:}\mathcal{T}_{VS}.\ \mathcal{T}_{VS}) \\ \mathcal{T}_{VS} & ::= & \mathcal{T}_V & \mid \mathcal{T}_S \end{array}$$

Theorem 3.18. Let $a \in \mathcal{T}_S$.

$$\begin{array}{ll} \Gamma \vdash_w a : A, & A \rightarrow_\beta \Pi x_1{:}A_1 \ldots x_n{:}A_n.s \\ \Gamma \vdash a : \Pi x_1{:}A_1 \ldots x_n{:}A_n.s' & \end{array}$$

Then

$$\Gamma, x_1{:}A_1 \ldots x_n{:}A_n \vdash b : s \text{ with } a \rightarrow_\beta \lambda x_1 : A_1 \ldots \lambda x_n : A_n.b$$

Proof: By induction on the derivation of $\Gamma \vdash_w a : A$.

Q. E. D.

Corollary 3.19. If $\Gamma \vdash_w A : s$ and $\Gamma \vdash A : s'$ then $\Gamma \vdash A : s$.

Proof:
Suppose $A \in \mathcal{T}_V$. By the uniqueness of types theorem for PTS^w's, we have that $s \equiv s'$.
Suppose $A \in \mathcal{T}_S$. By the previous theorem we have that $\Gamma \vdash A : s$.

Q. E. D.

Corollary 3.20. If $\Gamma \vdash_w A : s$ and $\Gamma \vdash a : A$ then $\Gamma \vdash A : s$.

Proof: It follows from lemma 3.14 that $\Gamma \vdash A : s'$ for some s'. By the previous corollary we have that $\Gamma \vdash A : s$. **Q. E. D.**

For an arbitrary specification we prove a weaker statement than theorem 3.16:

Theorem 3.21. Let $\Gamma \vdash_{\lambda S^w} a : A$.

1. If $\exists s \in \mathcal{S}[\Gamma \vdash_w A : s]$ then

$$\Gamma' \vdash_{\lambda S} a' : A' \text{ for } a \rightarrow_\beta a',\ A \rightarrow_\beta A' \text{ and } \Gamma \rightarrow_\beta \Gamma'$$

2. If $\Gamma \not\vdash_w A : -$ then

$$\Gamma', x_1{:}A'_1 \ldots x_n{:}A'_n \vdash_{\lambda S} a' : B \quad \text{where} \quad \begin{array}{l} A \equiv \Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.B, \\ a \rightarrow_\beta \lambda x_1{:}A'_1 \ldots \lambda x_n{:}A'_n.a' \\ \Gamma \rightarrow_\beta \Gamma',\ B \rightarrow_\beta B',\ A_i \rightarrow_\beta A'_i \text{ for all } i. \end{array}$$

Note that if $A \equiv s$ then $\Gamma' \vdash_{\lambda S} a' : s$ for $a \rightarrow_\beta a'$ and $\Gamma \rightarrow_\beta \Gamma'$.

Proof: This theorem is proved by induction on the derivation of $\Gamma \vdash_{\lambda S^w} a : A$. We consider only 2 cases:

- (abstraction) $\dfrac{\Gamma, x : A \vdash_w b : B}{\Gamma \vdash_w (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B)}$
  Suppose $\Gamma \vdash_w (\Pi x{:}A.\ B) : s$.

  1. By the generation lemma $\Gamma, x{:}A \vdash_w B : s_2$, $\Gamma \vdash_w A : s_1$ and $(s_1, s_2, s) \in \boldsymbol{R}$
     By the IH and corollary 3.20 we have that
     $$\Gamma', x : A' \vdash b' : B' : s_2 \quad \text{for } b \rightarrow_\beta b',\ B \rightarrow_\beta B' \text{ and } \Gamma, x{:}A \rightarrow_\beta \Gamma', x{:}A'.$$
     By the IH we have that $\Gamma' \vdash A' : s_1$. [3]
     Since $(s_1, s_2, s) \in \boldsymbol{R}$ we obtain the following derivation:
     $$\dfrac{\Gamma', x : A' \vdash b' : B' \quad \dfrac{\Gamma' \vdash A' : s_1 \quad \Gamma', x : A' \vdash B' : s_2}{\Gamma' \vdash (\Pi x{:}A'.\ B') : s}}{\Gamma' \vdash (\lambda x{:}A'.\ b') : (\Pi x{:}A'.\ B')}$$
  2. Suppose $\Gamma \not\vdash_w (\Pi x{:}A.\ B) : -$. There are two possibilities:
     (a) $\Gamma, x{:}A \vdash_w B : s$.
     By the IH we have that:
     $$\Gamma', x{:}A' \vdash_w b' : B' : s \quad \text{where} \quad \begin{array}{l} \Gamma, x{:}A \rightarrow_\beta \Gamma', x{:}A', \\ b \rightarrow_\beta b', B \rightarrow_\beta B' \end{array}$$
     (b) $\Gamma, x{:}A \not\vdash_w B : -$
     It follows from the IH that
     $$\Gamma', x{:}A', x_1{:}A'_1 \ldots x_n{:}A'_n \vdash_{\lambda S} b' : B' \quad \text{where} \quad \begin{array}{l} B \equiv \Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.B_0, \\ b \rightarrow_\beta \lambda x_1{:}A'_1 \ldots \lambda x_n{:}A'_n.b' \\ \Gamma \rightarrow_\beta \Gamma' \\ B_0 \rightarrow_\beta B' \\ A_i \rightarrow_\beta A'_i \text{ for all } i. \end{array}$$

- (application) $\dfrac{\Gamma \vdash_w b : (\Pi x{:}A.\ B) \quad \Gamma \vdash a : A}{\Gamma \vdash_w (b\ \ a) : B[x := a]}$.

  1. Suppose $\Gamma \vdash_w B[x := a] : s$.
     There are two cases:
     (a) Suppose $\Gamma \vdash_w (\Pi x{:}A.\ B) : s'$.
     By the IH we have that
     $$\Gamma' \vdash b' : (\Pi x{:}A'.\ B') \text{ for } b \rightarrow_\beta b',\ A \rightarrow_\beta A',\ B \rightarrow_\beta B' \text{ and } \Gamma \rightarrow_\beta \Gamma'$$
     Note that $\Gamma \vdash_w A : s_1$. By the IH we have that
     $$\Gamma' \vdash a' : A' : s'_1 \text{ for } A \rightarrow_\beta A' \text{ and } a \rightarrow_\beta a'.$$
     Hence we have the following derivation:
     $$\dfrac{\Gamma' \vdash b' : (\Pi x{:}A'.\ B') \quad \Gamma' \vdash a' : A'}{\Gamma' \vdash (b'\ \ a') : B'[x := a']}$$

[3] Actually we deduce $\Gamma" \vdash A" : s_1$ and by the Church Rosser theorem we can always find a common reduct of A' and $A"$ and of Γ' and $\Gamma"$. We omit this kind of details.

(b) Suppose $\Gamma \not\vdash_w (\Pi x{:}A.\ B) : -$.
Hence $B \equiv \Pi x_1{:}A_1 \ldots x_n{:}A_n.B_0$. It follows from lemma 3.15 that B_0 cannot be a topsort. Then

$$\begin{array}{ll} \exists n, s_1, s_2 & \Gamma, x{:}A, x_1{:}A_1 \ldots x_{n-1}{:}A_{n-1} \vdash_w A_n : s_1 \\ & \Gamma, x{:}A, x_1{:}A_1 \ldots x_{n-1}{:}A_{n-1}, x_n{:}A_n \vdash_w B_0 : s_2 \end{array}$$

Moreover, we have that:

$b \rightarrow_\beta \lambda x{:}A'.\ \lambda x_1{:}A'_1 \ldots \lambda x_n{:}A'_n.b'$ and $\Gamma', x{:}A', x_1{:}A'_1 \ldots, x_n{:}A'_n \vdash b' : B'$

By the IH we have that $\Gamma' \vdash a' : A'$ for $a \rightarrow_\beta a'$.
By the substitution lemma we have that:

$$\Gamma', x_1{:}A'_1[x := a'] \ldots, x_n{:}A'_n[x := a'] \vdash b'[x := a'] : B'[x := a']$$

Note that $\Gamma \vdash_w B[x := a] : s$ and

$$\begin{array}{rcl} B[x := a] & \equiv & \Pi x_1{:}A_1[x := a] \ldots x_n{:}A_n[x := a].B_0[x := a] \\ & \rightarrow_\beta & \Pi x_1{:}A'_1[x := a'] \ldots x_n{:}A'_n[x := a'].B'[x := a'] \end{array}$$

By the subject reduction theorem we have that:

$$\Gamma' \vdash_w \Pi x_1{:}A'_1[x := a'] \ldots x_n{:}A'_n[x := a'].B'[x := a'] : s$$

We can easily construct a derivation of
$\Gamma' \vdash \Pi x_1{:}A'_1[x := a'] \ldots x_n{:}A'_n[x := a'].B'[x := a'] : s$ in a PTS.
Applying Abstraction Rule n-times we obtain a derivation of:

$$\Gamma' \vdash (\lambda x_1{:}A'_1 \ldots \lambda x_n{:}A'_n.b')[x := a'] : (\Pi x_1{:}A'_1 \ldots x_n{:}A'_n.B')[x := a']$$

$$\begin{array}{lrl} \text{with} \quad (b\ \ a) & \rightarrow_\beta & (\lambda x{:}A'.\ \lambda x_1{:}A'_1 \ldots \lambda x_n{:}A'_n.b')a' \\ & \rightarrow_\beta & \lambda x_1{:}A'_1 \ldots \lambda x_n{:}A'_n.b'[x := a'] \end{array}$$

2. Suppose $\Gamma \not\vdash_w B[x := a] : -$.
It follows from lemma 3.15 that $\Gamma \not\vdash_w (\Pi x{:}A.\ B) : -$.
By description theorem we have that $(\Pi x{:}A.\ B) \equiv \Pi x{:}A.\Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.B_0$.
By the IH we have that

$$\Gamma', x{:}A', x_1{:}A'_1, \ldots, x_n{:}A'_n \vdash_{\lambda S} b' : B'$$

$$\begin{array}{rl} \text{where} & b \rightarrow_\beta \lambda x{:}A.\ x_1{:}A_1 \ldots \lambda x_n{:}A_n.b' \\ & \Gamma, x{:}A, x_1{:}A_1, \ldots, x_n{:}A_n \rightarrow_\beta \Gamma', x{:}A', x_1{:}A'_1, \ldots, x_n{:}A'_n \\ & B_0 \rightarrow_\beta B' \end{array}$$

By the IH we have that

$$\Gamma' \vdash a' : A' \text{ for } a \rightarrow_\beta a',\ A \rightarrow_\beta A',\ \Gamma \rightarrow_\beta \Gamma'$$

Then

$$\begin{array}{lrl} (b\ \ a) & \rightarrow_\beta & (\lambda x{:}A.\ \lambda x_1{:}A_1 \ldots \lambda x_n{:}A_n.b')a' \\ & \rightarrow_\beta & \lambda x_1{:}A_1[x := a'] \ldots \lambda x_n{:}A_n[x := a'].b'[x := a'] \\ & \rightarrow_\beta & \lambda x_1{:}A'_1[x := a'] \ldots \lambda x_n{:}A'_n[x := a'].b'[x := a'] \end{array}$$

By the substitution lemma we have that

$$\Gamma', x_1{:}A'_1[x := a'], \ldots, x_n{:}A'_n[x := a'] \vdash_{\lambda S} b'[x := a'] : B'[x := a']$$

- (conversion) $\dfrac{\Gamma \vdash_w b : B \quad \Gamma \vdash_w A : s \quad B =_\beta A}{\Gamma \vdash_w b : A}$

1. Suppose $\Gamma \vdash_w B : s'$ By the IH we have that $\Gamma' \vdash b' : B'$ for $B \rightarrow_\beta B'$. By the theorem of Church Rosser there exists B" such that $A, B' \rightarrow_\beta B$".
Hence $\Gamma' \vdash b' : B$".

2. Suppose $\Gamma \not\vdash_w B : -$. Hence $B \equiv \Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.B_0$. Note that B_0 cannot be a topsort. Then

$$\begin{array}{ll} \exists n, s_1, s_2 & \Gamma, x_1{:}A_1 \ldots x_{n-1}{:}A_{n-1} \vdash_w A_n : s_1 \\ & \Gamma, x_1{:}A_1 \ldots x_{n-1}{:}A_{n-1}, x_n{:}A_n \vdash_w B_0 : s_2 \end{array}$$

By the IH we have that:

$$\Gamma', x_1{:}A'_1 \ldots, x_n{:}A'_n \vdash b' : B'$$
$$\text{with } B_0 \rightarrow_\beta B' \text{ and } b \rightarrow_\beta \lambda x_1{:}A'_1 \ldots \lambda x_n{:}A'_n.b'$$

By the IH we have that $\Gamma' \vdash A' : s$ with $A \rightarrow_\beta A'$ and $\Gamma \rightarrow_\beta \Gamma'$.
By the theorem of Church-Rosser, A' and $\Pi x_1{:}A'_1 \ldots x_n{:}A'_n.B'$ reduces to

$$\Pi x_1{:}A_1" \ldots x_n{:}A_n".B"$$

By subject reduction theorem we have that:

$$\Gamma' \vdash \Pi x_1{:}A_1" \ldots x_n{:}A_n".B" : s$$

By subject reduction theorem and conversion rule we have that:

$$\Gamma', x_1{:}A_1" \ldots, x_n{:}A_n" \vdash b' : B"$$

Applying Abstraction Rule n-times we obtain a derivation of:

$$\Gamma' \vdash \lambda x_1{:}A_1" \ldots \lambda x_n{:}A_n".b' : \Pi x_1{:}A_1" \ldots x_n{:}A_n".B"$$

Q. E. D.

Corollary 3.22. (Normalization)
If λS is weakly normalizing then λS^w is weakly normalizing too.

Proof: Suppose $\Gamma \vdash_w a : A$. We have two possibilities:

1. Suppose $\Gamma \vdash_w A : s$. By the previous theorem there exists Γ', a' and A' such that $\Gamma' \vdash a' : A'$ and $a \rightarrow_\beta a'$. Since λS is weakly normalizing, we have that $a' \rightarrow_\beta nf(a')$ where $nf(a')$ is the β-normal form of a'. Then a is weakly normalizing.

2. Suppose $\Gamma \not\vdash_w A : -$. By previous theorem we have that

$$\begin{array}{rl} \Gamma', x_1{:}A'_1 \ldots x_n{:}A'_n \vdash_{\lambda S} a' : B & \\ \text{where} & A \equiv \Pi x_1{:}A_1 \ldots \Pi x_n{:}A_n.B, \\ & a \rightarrow_\beta \lambda x_1{:}A'_1 \ldots \lambda x_n{:}A'_n.a' \\ & \Gamma \rightarrow_\beta \Gamma' \\ & B \rightarrow_\beta B' \\ & A_i \rightarrow_\beta A'_i \text{ for all } i. \end{array}$$

Since λS is weakly normalizing in particular a' and A'_i for all i are weakly normalizing. Hence

$$\begin{array}{rcl} a & \rightarrow_\beta & \lambda x_1{:}A'_1 \ldots \lambda x_n{:}A'_n.a' \\ & \rightarrow_\beta & \lambda x_1{:}nf(A'_1) \ldots \lambda x_n{:}nf(A'_n).nf(a') \\ & \equiv & nf(a) \end{array}$$

Then a is weakly normalizing.

Q. E. D.

Corollary 3.23. Let λS be a PTS extending $\lambda 2$.
If there is a proof of $\Pi\alpha : *.\alpha$ in λS^w then there is also a proof of $\Pi\alpha : *.\alpha$ in λS.

Proof: Suppose there exist Γ and p such that $\Gamma \vdash_w p : \Pi\alpha : *.\alpha$. The type $\Pi\alpha : *.\alpha$ is not a topsort in $(\lambda 2)^w$. By previous theorem we have that there exist Γ' and p' such that $\Gamma' \vdash_w p' : \Pi\alpha : *.\alpha$.
Q. E. D.

3.3 Strong Normalization for β-reduction

In this section we define the notion of 'completion' and we prove that $\lambda S = \lambda S^w$ if and only if S is a completion of itself. We also prove that if S' is a completions of S, strong normalization of $\lambda S'$ implies strong normalization of λS^w.

First we define the notion of completion as in [SP94]:

Definition 3.24. Let $S = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ and $S' = (\boldsymbol{S}', \boldsymbol{A}', \boldsymbol{R}')$ be such that

1. $\boldsymbol{S} \subseteq \boldsymbol{S}'$, $\boldsymbol{A} \subseteq \boldsymbol{A}'$, and $\boldsymbol{R} \subseteq \boldsymbol{R}'$
2. $\boldsymbol{S}'$ is full, i.e. for all $s_1, s_2 \in \boldsymbol{S}'$ there exists s_3 such that $(s_1, s_2, s_3) \in \boldsymbol{R}$.
3. for all $s \in \boldsymbol{S}$ there is a sort $s' \in \boldsymbol{S}'$ such that $(s, s') \in \boldsymbol{A}'$ (i.e. the sorts of S are not topsorts in S').

Then the specification S' is called *a completion of* S.

Example 3.25. The system λC_∞ is a completion of λC and of itself.

Lemma 3.26. Let S' be a completion of S.

$$\text{If } \Gamma \vdash_{\lambda S^w} a : A \text{ then } \Gamma \vdash_{\lambda S'^w} A : s \text{ for some sort } s.$$

Theorem 3.27. Let S' be a completion of S.

$$\text{If } \Gamma \vdash_{\lambda S^w} a : A \text{ then } \Gamma \vdash_{\lambda S'} a : A \text{ and } \Gamma \vdash_{\lambda S'} A : s \text{ for some } s.$$

Corollary 3.28. Let S be a completion of itself.

$$\Gamma \vdash_{\lambda S^w} a : A \text{ iff } \Gamma \vdash_{\lambda S} a : A$$

A consequence of this corollary is that $(\lambda C_\infty)^w = \lambda C_\infty$. Hence it is redundant to write the type premise in the abstraction rule for λC_∞.

Next we will prove that the set of legal terms in λS is equal to the set of legal terms in λS^w if and only if S is a completion of itself. We refer to the set of legal terms of λS as $\mathcal{L}(\lambda S)$.

Theorem 3.29. (Redundancy of the type premise)
$\mathcal{L}(\lambda S) = \mathcal{L}(\lambda S^w)$ iff S is a completion of itself.

Proof: Corollary 3.28 is one of the implications of this theorem.

Conversely, we will prove that S is full and has no topsorts.

Suppose there is a topsort s_0. There is at least one axiom $c : s_0$. Hence $\vdash_w \lambda x{:}c.\ c : (c \to s_0)$. Since $\mathcal{L}(\lambda S) = \mathcal{L}(\lambda S^w)$, we have that

$$\Gamma \vdash (c \to s_0) : s \text{ for some } s \text{ and } \Gamma$$

Hence s_0 cannot be a topsort.

Given the sorts s_1, s_2 we prove that there exists a sort s such that $(s_1, s_2, s) \in \boldsymbol{R}$. Since S has no topsorts we have that $s_1 : s_1'$ and $s_2 : s_2'$ for some s_1' and s_2'. Hence we have that:

$$\vdash_w (\lambda\alpha : s_1.\lambda\beta : s_2.\lambda y : \beta.\lambda x{:}\alpha.\ y) : \Pi\alpha : s_1.\Pi\beta : s_2.(\alpha \to \beta)$$

The type of this term is a legal term in λS then:

$$\Gamma \vdash \Pi\alpha : s_1.\Pi\beta : s_2.(\alpha \to \beta) : s$$

Hence there exists s such that $(s_1, s_2, s) \in \boldsymbol{R}$.

Q. E. D.

Theorem 3.30. Strong Normalization
Let S' be a completion of S.
If $\lambda S'$ is β-strongly normalizing then λS^w is β-strongly normalizing too.

3.4 Strong Normalization for the illegal reduction

In this section we will show that the β-reduction restricted to illegal redexes is strongly normalizing.

Definition 3.31. Given Γ such that $\Gamma \vdash_w (\lambda x{:}A.\ b)a : D$.
We define *the illegal reduction* w.r.t. the context Γ as:

$$(\lambda x{:}A.\ b)a \rightarrow_{\beta_I} b[x := a] \text{ if } (\lambda x{:}A.\ b)a \text{ is an illegal redex w.r.t. } \Gamma.$$

Of course we have to add all the compatibility rules to this rule.

Recall that a development is a reduction sequence in which only descendents of redexes that are present in the initial term may be contracted. In a development one is not allowed to contract redexes that are created along the way.

An extension of the notion of development, called *superdevelopments*, was introduced and proved to be finite in [Raa93]. In that paper the notion of development was extended to include reduction sequences in which one can contract not only redexes that descend from the initial term but also some redexes that are created during reduction.

There are three ways of creating new redexes (see [Lev78]):

1. $((\lambda x{:}A.\lambda y{:}B.d)e)f \rightarrow_\beta (\lambda y{:}B.d[x := e])f$
2. $(\lambda x{:}A.x)(\lambda y{:}B.d)e \rightarrow_\beta (\lambda y{:}B.d)e$
3. $(\lambda x : A.C[x\ \ d])(\lambda y : B.e) \rightarrow_\beta C'[(\lambda y : B.e)d']$ where C' and d' are obtained from C and d replacing all free occurrences of x by $(\lambda y{:}B.e)$.

The first two ways of creating redexes are 'innocent' and they may be contracted in a superdevelopment. The result that all superdevelopments are finite shows that infinte β-reduction sequences are due to the presence of the third type of redexes.

New redexes containing illegal abstractions can be created only with case 1). This is because the type of a variable cannot be a toptype. In case 2) and 3) a variable x is substituted by $(\lambda y{:}B.e)$ and this abstraction is not illegal. Hence the illegal abstractions of a term constitute an initial labelling of a superdevelopment. All the redexes with illegal abstractions that are created along this superdevelopment are labelled.

Hence we have the following result:

Theorem 3.32. The reduction $\rightarrow_{\beta_I}$ is strongly normalizing.

Proof: It follows from the finite superdevelopments theorem (see [Raa93]) and the considerations above. **Q. E. D.**

Since all superdevelopments are finite we can get rid of all the illegal abstractions occuring in a term in a finite number of steps. Note that the last term of a complete superdevelopment of illegal abstractions is computed using $\mathcal{C}$.

3.5 A Type Inference Algorithm for PTS^w

Next we define a system that is nearly syntax directed for Pure Type Systems without the type premise in the abstraction rule. It is nearly syntax directed because given b and Γ the term B such that $\Gamma \vdash_{wnsd} b : B$ is not unique.

Notation 3.33. We write $\Gamma \vdash_{wnsd} a :\twoheadrightarrow A$ for $\Gamma \vdash_{wnsd} a : A_0$ and $A_0 \rightarrow_\beta A$.
We write $\Gamma \vdash_{wnsd} A :\twoheadrightarrow_{wh} A$ for $\Gamma \vdash_{wnsd} a : A_0$ and A_0 weak-head reduces to A.

Definition 3.34. The Syntax Directed Pure Type System PTS^w_{nsd} determined by the specification $S = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ is denoted as λS^w_{nsd} and defined by the notion of type derivation $\Gamma \vdash_{wnsd} b : B$ given by the following axioms and rules:

$$(axiom) \quad \epsilon \vdash_{wnsd} c : s \qquad \text{for } (c, s) \in \boldsymbol{A}$$

$$(start) \quad \frac{\Gamma \vdash_{wnsd} A :\twoheadrightarrow s}{\Gamma, x : A \vdash_{wnsd} x : A} \qquad \text{where } x \text{ is } \Gamma\text{-fresh}$$

$$(weakening) \quad \frac{\Gamma \vdash_{wnsd} b : B \quad \Gamma \vdash_{wnsd} A :\twoheadrightarrow s}{\Gamma, x : A \vdash_{wnsd} b : B} \qquad \text{where } x \text{ is } \Gamma\text{-fresh and } b \in \mathbf{C} \cup V$$

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

$$(abstraction) \quad \frac{\Gamma, x : A \vdash_{wnsd} b : B}{\Gamma \vdash_{wnsd} (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B)}$$

$$(application) \quad \frac{\Gamma \vdash_{wnsd} b :\twoheadrightarrow_{wh} (\Pi x{:}A.\ B) \quad \Gamma \vdash_{wnsd} a : A'}{\Gamma \vdash_{wnsd} (b\ \ a) : B[x := a]} \qquad A =_\beta A'$$

where s ranges over sorts, i.e. $s \in \boldsymbol{S}$.

Note that when the specification is singly sorted, this set of rules is syntax directed. The proof of the following theorems will be omitted because they are direct.

Theorem 3.35. **(Soundness)** If $\Gamma \vdash_{wnsd} a : A$ then $\Gamma \vdash_w a : A$.

Theorem 3.36. **(Completness)** If $\Gamma \vdash_w a : A$ then $\Gamma \vdash_{wnsd} a : A'$ for $A =_\beta A'$.

Next we define a type inference algorithm for PTS^w's:

Primitives

$\mathcal{NF} : \mathcal{T} \to \mathcal{T}$ computes the normal form,
$\mathcal{WHNF} : \mathcal{T} \to \mathcal{T}$ computes the weak-head normal form,
$\mathcal{EQ} : \mathcal{T} \times \mathcal{T} \to \mathbf{Bool}$ yields true if two terms are β-equal.

Type Inference Algorithm for arbitrary PTS^w's

We define a mapping $\mathcal{INF}_w : \mathcal{C} \times \mathcal{T} \to \mathcal{P}(\mathcal{T})$ as follows:

$$\begin{aligned}
\mathcal{INF}_w(\epsilon; c) &= \{s \mid (c, s) \in \boldsymbol{A}\} \\
\mathcal{INF}_w(< \Gamma, x : A >; x) &= \{A\} \quad \text{if } s \in \mathcal{NF}(\mathcal{INF}_w(< \Gamma; A)) \\
\mathcal{INF}_w(< \Gamma, x : A >; y) &= \mathcal{INF}_w(\Gamma; y) \quad \text{if } [s \in \mathcal{NF}(\mathcal{INF}_w(\Gamma; A))] \wedge y \neq x \\
\mathcal{INF}_w(< \Gamma, x : A >; c) &= \mathcal{INF}_w(\Gamma; c) \text{ if } (s \in \mathcal{NF}(\mathcal{INF}_w(\Gamma; A)) \ ) \\
\mathcal{INF}_w(\Gamma; (\Pi x{:}A.B)) &= \{s_3 \mid [s_1 \in \mathcal{NF}(\mathcal{INF}_w(\Gamma; A))] \wedge [s_2 \in \mathcal{NF}(\mathcal{INF}_w(\Gamma, x : A; B))] \\
&\qquad \wedge (s_1, s_2, s_3) \in \boldsymbol{R}\} \\
\mathcal{INF}_w(\Gamma; (\lambda x{:}A.\ b)) &= \{(\Pi x{:}A.\ B) \mid B \in \mathcal{INF}_w(\Gamma, x : A; b)\} \\
\mathcal{INF}_w(\Gamma; (b\ \ a)) &= \{B[x := a] \mid [(\Pi x{:}A.\ B) \in \mathcal{WHNF}(\mathcal{INF}_w(\Gamma; b))] \wedge \\
&\qquad [A' \in \mathcal{INF}_w(\Gamma; a)] \wedge \mathcal{EQ}(A; A')\}
\end{aligned}$$

Note that if the term a has no type under the context Γ then $\mathcal{INF}_w(\Gamma, a) = \emptyset$.
In case of singly sorted PTS's we write $\mathcal{INF}_w(\Gamma, a) = A$ instead of $\mathcal{INF}_w(\Gamma, a) = \{A\}$.

Theorem 3.37. Given iven $\Gamma \in \mathcal{C}$ and $a \in \mathcal{T}$, $\mathcal{INF}_w(\Gamma, a) = \{A \mid \Gamma \vdash_w a : A\}/=_\beta$.

Proof: It is proved by induction on (Γ, a).

Q. E. D.

4 Typability for PTS's

We compare several syntax directed systems and we discuss which system would yield 'the best algorithm for type inference' :

- In section 4.1, we present the syntax directed system PTS_{sd} (see [vBJMP93]). All attempts to prove completeness for this algorithm have been unsuccesfull. The main difficulty seems to be the impossibility to apply the inductive hypothesis to the type premise in the abstraction rule.

  We consider variations of this system by changing the type premise of the abstraction rule.

- In section 4.2, we present another syntax directed system PTS_{nf}. The PTS_{nf} and PTS_{sd} differ only in the abstraction rule. The type in the abstraction rule of PTS_{nf} is reduced to the normal form.

  We can prove Soundness and Completeness of this algorithm with respect to PTS's.

- In section 4.3, we define a type inference algorithm for singly sorted PTS's. We consider the syntax directed system PTS^{w}_{sd} for Pure Type Systems without the type premise in the abstraction rule. This system allows "illegal abstractions" and it is not yet a type inference algorithm for PTS's. We will turn the type inference algorithm for Pure Type Systems without the type premise in the abstraction rule into a type inference algorithm for Pure Type Systems checking separately that the term does not contain illegal abstractions.

  Given Γ, a, the type inference algorithm for PTS's can be described as follows:

  1. Find A such that $\Gamma \vdash_w a : A$.
  2. The algorithm yields A if the abstractions in a and in Γ are not illegal.

  This type inference algorithm for singly sorted PTS's is efficient and it is also simple as the one presented in section 4.1.

4.1 The simplest Type Inference Algorithm

Next we define the syntax directed system as in [vBJMP93].

Notation 4.1. We write $\Gamma \vdash_{sd} a :\twoheadrightarrow A$ for $\Gamma \vdash_{sd} a : A_0$ and $A_0 \rightarrow_\beta A$.
We write $\Gamma \vdash_{sd} A :\twoheadrightarrow_{wh} A$ for $\Gamma \vdash_{sd} a : A_0$ and A_0 weak-head reduces to A.

Definition 4.2. The Pure Type System PTS_{sd} determined by the specification $S = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ is denoted as λS_{sd} and defined by the notion of type derivation $\Gamma \vdash_{sd} b : B$ given by the following axioms and rules:

$$(axiom) \quad \epsilon \vdash_{sd} c : s \qquad \text{for } (c, s) \in \boldsymbol{A}$$

$$(start) \quad \frac{\Gamma \vdash_{sd} A :\twoheadrightarrow s}{\Gamma, x : A \vdash_{sd} x : A} \qquad \text{where } x \text{ is } \Gamma\text{-fresh}$$

$$(weakening) \quad \frac{\Gamma \vdash_{sd} b : B \quad \Gamma \vdash_{sd} A :\twoheadrightarrow s}{\Gamma, x : A \vdash_{sd} b : B} \qquad \text{where } x \text{ is } \Gamma\text{-fresh and } b \in \mathbf{C} \cup V$$

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

$$(abstraction) \quad \frac{\begin{array}{c}\Gamma, x : A \vdash_{sd} b : B \\ \Gamma \vdash_{sd} (\Pi x{:}A.B) : s\end{array}}{\Gamma \vdash_{sd} (\lambda x{:}A.\ b) : (\Pi x{:}A.\ B)}$$

$$(application) \quad \frac{\Gamma \vdash_{sd} b :\twoheadrightarrow_{wh} (\Pi x{:}A.\ B) \quad \Gamma \vdash_{sd} a : A'}{\Gamma \vdash_{sd} (b\ \ a) : B[x := a]} \qquad A' =_\beta A$$

where s ranges over sorts, i.e. $s \in \boldsymbol{S}$.

Note that the shape of a term together with the context determines the rule to be applied:

- The wekening rule is now deterministic because it has been restricted to variables and constants.
- The conversion rule has been spread in the rest of the rules adding reduction to some rules.

Note that when the specification is singly sorted, $\mathbf{PTS}_{sd}$ is syntax directed.
It is easy to prove Soundness of these systems with respect to PTS's:

Theorem 4.3. (Soundness)

$$\text{If } \Gamma \vdash_{sd} a : A \text{ then } \Gamma \vdash a : A.$$

Next example shows that completeness is not true for non-singly sorted PTS's.

Example 4.4. The following specification is not singly sorted:

$$\lambda S \begin{array}{|ll|} \boldsymbol{S} & 0,1,2 \\ \boldsymbol{A} & 0:1, 0:2, 1:2 \\ \boldsymbol{R} & (2,2) \end{array}$$

We take $A \equiv (\lambda y{:}1.y)0$. We can derive $x : A \vdash (\lambda z : 0.x) : (0 \to 0)$. Howewer there is no D such that $x : A \vdash_{sd} (\lambda z : 0.x) : D$.

For singly sorted PTS's, Completeness has not been possible to prove:

(Completeness) Suppose S is singly sorted. If $\Gamma \vdash a : A$ then there exists A' such that $A =_\beta A'$ and $\Gamma \vdash_{sd} a : A'$.

4.2 A Simple But Inefficient Type Inference Algorithm

Next we define a syntax directed set of rules for normalizing Pure Type Systems. The type derivation for these systems is denoted as $\vdash_{nf}$.

This system is defined from the previous one changing just the abstraction rule. The type of the abstraction $(\Pi x{:}A.B)$ is reduced to its normal form.

The new abstraction rule is as follows:

$$\frac{\Gamma, x : A \vdash_{nf} b : B \quad \Gamma \vdash_{nf} (\Pi x{:}A_0.B_0) : s}{\Gamma \vdash_{nf} (\lambda x{:}A.\ b) : (\Pi x{:}A_0.\ B_0)}$$

where $(\Pi x{:}A_0.\ B_0)$ is the normal form of $(\Pi x{:}A.\ B)$.

We called these systems λS_{nf} because the normal form is computed in the abstraction rule.
The proof of the following theorem will be omitted because it is direct.

Theorem 4.5. (Soundness) If $\Gamma \vdash_{nf} a : A$ then $\Gamma \vdash_w a : A$.

Theorem 4.6. (Completeness) If $\Gamma \vdash a : A$ then $\Gamma \vdash_{nf} a : A'$ for $A =_\beta A'$.

Note that for these systems we can prove subject reduction and completeness. This is because the type premise remains invariant under reduction.

If the specification is singly sorted then these systems are syntax directed.

Howewer the type inference algorithm associated to these systems is not efficient because it computes the normal form of $(\Pi x{:}A.B)$ in the case of the abstraction rule.

4.3 A More efficient Type Inference Algorithm

Restricting the system PTS^{w}_{nsd}, we obtain a type inference algorithm for singly sorted PTS's.

First notice that the mapping $\mathcal{C}$ defined in 3.11 can be adapted to the system PTS^{w}_{nsd}. We call this mapping $\mathcal{C}'$. This mapping contracts the illegal redexes of a term with respect to λS^{w}_{sd}. The notions of illegal abstraction and illegal redex for λS^{w}_{sd} are defined similarly to definition 3.9 and definition 3.10.

Contraction of illegal redexes

Next we define a mapping that contracts the illegal redexes of a term. Suppose that the specification is singly sorted and that $\Gamma \vdash_w a : A$. We define $\mathcal{C}'_\Gamma$ on the typable terms as follows:

$$
\begin{array}{rcl}
\mathcal{C}'_\Gamma(x) & = & x \\
\mathcal{C}'_\Gamma(a\ b) & = & \begin{cases} a_0[x := \mathcal{C}'_\Gamma(b)] & \text{if } (\mathcal{C}'_\Gamma(a) = \lambda x{:}A.a_0) \wedge (\mathcal{INF}_w(\Gamma; \lambda x{:}A.a_0) = B) \wedge \\ & \qquad (\mathcal{INF}_w(\Gamma, B) = \emptyset) \\ (\mathcal{C}'_\Gamma(a)\ \mathcal{C}'_\Gamma(b)) & \text{otherwise} \end{cases} \\
\mathcal{C}'_\Gamma(\lambda x{:}A.\ a) & = & (\lambda x{:}\mathcal{C}'_\Gamma(A).\ \mathcal{C}'_{\Gamma,x:A}(a)) \\
\mathcal{C}'_\Gamma(\Pi x{:}A.\ B) & = & (\Pi x{:}\mathcal{C}'_\Gamma(A).\ \mathcal{C}'_{\Gamma,x:A}(B))
\end{array}
$$

We write $\mathcal{C}'(a)$ instead of $\mathcal{C}'_\Gamma(a)$.

Type Inference Algorithm

We define a mapping $\mathcal{INF} : \mathcal{C} \times \mathcal{T} \rightarrow \mathcal{T} \cup \perp$ that given $\Gamma \in \mathcal{C}$ and $a \in \mathcal{T}$, $\mathcal{INF}(\Gamma, a)$ yields A if $\Gamma \vdash a : A$ and $\perp$ otherwise. as follows:

$$
\begin{array}{rcllll}
\mathcal{INF}(\Gamma, a) & = & \text{let} & A & = \mathcal{INF}_w(\Gamma; a) \text{ in} \\
 & & & \text{if} & (\ A = s \vee \mathcal{NF}(\mathcal{INF}_w(\Gamma; A)) = s\ ) \wedge \\
 & & & & (\mathcal{C}'(a) = a) \wedge (\mathcal{C}'(\Gamma) = \Gamma) \\
 & & & \text{then} & \mathcal{C}'(A) \\
 & & & \text{else} & \perp
\end{array}
$$

Note that this algorithm is more efficient than the previous one because it does not compute the normal form.

We will prove some lemmas which are necessary to prove Completeness. First we show that $\mathcal{C}$ and $\mathcal{C}'$ are the same function on the typable terms of λS^{w}_{sd}:

Lemma 4.7. Let S be a full or singly sorted specification. If $\Gamma \vdash_{wnsd} a : A$ then $\mathcal{C}_\Gamma(a) = \mathcal{C}'_\Gamma(a)$.

Proof: Due to theorem 3.35, $\Gamma \vdash_w a : A$. Hence $\mathcal{C}_\Gamma$ is defined for a. We prove that $\mathcal{C}_\Gamma(a) = \mathcal{C}'_\Gamma(a)$ by induction on the structure of a.

Suppose $a \equiv (d\ e)$ and $\mathcal{C}'(a) = (\lambda x : A.f)$ is an illegal abstraction of a w.r.t. $(\lambda S)^w$. By theorem 3.36 it is also an illegal abstraction w.r.t. $(\lambda S)^{w}_{sd}$. **Q. E. D.**

Theorem 4.8. (Soundness) Let S be a full or functional specification.
If the following conditions hold:

1. $\Gamma \vdash_{wnsd} a : A$,
2. $\Gamma \vdash_{wnsd} A :\twoheadrightarrow s$ or $A \equiv s$
3. $\mathcal{C}'(\Gamma) = \Gamma$ and $\mathcal{C}'(a) = a$.

then $\Gamma \vdash a : A$.

Proof: By theorem 3.35 we have that $\Gamma \vdash_w a : A$. By theorem 3.16 we have that $\mathcal{C}(\Gamma) \vdash \mathcal{C}(a) : \mathcal{C}(A)$. By previous lemma, $\mathcal{C}'(\Gamma) \vdash \mathcal{C}'(a) : \mathcal{C}'(A)$. Hence $\Gamma \vdash a : \mathcal{C}'(A)$.

Q. E. D.

Theorem 4.9. (Completeness)
If $\Gamma \vdash a : A$ then the following holds:

1. $\Gamma \vdash_{wnsd} a : A$
2. $\Gamma \vdash_{wnsd} A :\twoheadrightarrow s$ or $A \equiv s$
3. $\mathcal{C}'(\Gamma) = \Gamma$, $\mathcal{C}'(a) = a$ and $\mathcal{C}'(A) = A$.

Proof: It is a direct verification. **Q. E. D.**

Acknowledgements. I would like to thank Herman Geuvers for his suggestions and various comments to a preliminary version of this paper specially for a better formulation of the results of section 3.2.

I would also like to thank Erik Poll and Femke van Raamsdonk, Erik for his assistance on Pure Type Systems and their type checking algorithms and Femke for our conversations on superdevelopments.

References

[Bar91] Henk Barendregt. Introduction to generalised type systems. *Journal of Functional Programming*, 1:124–154, 1991.

[Bar92] Henk Barendregt. Lambda calculi with types. In D. M. Gabbai, S. Abramsky, and T. S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1. Oxford University Press, 1992.

[Lev78] J. J. Levy. *Réductions correctes et optimales dans le lambda-calcul.* PhD thesis, Université de Paris VII, 1978.

[Luo89] Z. Luo. ECC, the Extended Calculus of Constructions. In *Logic in Computer Science*, pages 386–395. IEEE, 1989.

[Pol93] Erik Poll. A type checker for bijective pure type systems. Computing Science Note (93/22), Eindhoven University of Technology, 1993.

[Raa93] Femke Van Raamsdonk. Confluence and superdevelopment. In Claude Kirchner, editor, *Proceedings of the RTA*, pages 168–183, 1993.

[SP94] Paula Severi and Erik Poll. Pure type systems with definitions. In A. Nerode and Yu.V. Matiyasevich, editors, *Logical Foundations of Computer Science: Proceedings of the Third International Symposium*, number 813, pages 316–328. LFCS'94, St. Petersburg Russia, Springer-Verlag, Berlin, New York, 1994.

[vBJ93] Bert van Benthem Jutting. Typing in pure type systems. *Information and Computation*, 105:30–41, 1993.

[vBJMP93] Bert van Benthem Jutting, James McKinna, and Randy Pollack. Checking algorithms for pure type systems. *LNCS*, 806, 1993.

Contents

In this series appeared:

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.
93/22	E. Poll	A Typechecker for Bijective Pure Type Systems, p. 28.
93/23	E. de Kogel	Relational Algebra and Equational Proofs, p. 23.
93/24	E. Poll and Paula Severi	Pure Type Systems with Definitions, p. 38.
93/25	H. Schepers and R. Gerth	A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems, p. 31.
93/26	W.M.P. van der Aalst	Multi-dimensional Petri nets, p. 25.
93/27	T. Kloks and D. Kratsch	Finding all minimal separators of a graph, p. 11.
93/28	F. Kamareddine and R. Nederpelt	A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.
93/29	R. Post and P. De Bra	GOLD, a Graph Oriented Language for Databases, p. 42.
93/30	J. Deogun T. Kloks D. Kratsch H. Müller	On Vertex Ranking for Permutation and Other Graphs, p. 11.
93/31	W. Körver	Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.
93/32	H. ten Eikelder and H. van Geldrop	On the Correctness of some Algorithms to generate Finite Automata for Regular Expressions, p. 17.
93/33	L. Loyens and J. Moonen	ILIAS, a sequential language for parallel matrix computations, p. 20.
93/34	J.C.M. Baeten and J.A. Bergstra	Real Time Process Algebra with Infinitesimals, p.39.
93/35	W. Ferrer and P. Severi	Abstract Reduction and Topology, p. 28.
93/36	J.C.M. Baeten and J.A. Bergstra	Non Interleaving Process Algebra, p. 17.
93/37	J. Brunekreef J-P. Katoen R. Koymans S. Mauw	Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.
93/38	C. Verhoef	A general conservative extension theorem in process algebra, p. 17.
93/39	W.P.M. Nuijten E.H.L. Aarts	Job Shop Scheduling by Constraint Satisfaction, p. 22.

	D.A.A. van Erp Taalman Kip K.M. van Hee	
93/40	P.D.V. van der Stok M.M.M.P.J. Claessen D. Alstein	A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.
93/41	A. Bijlsma	Temporal operators viewed as predicate transformers, p. 11.
93/42	P.M.P. Rambags	Automatic Verification of Regular Protocols in P/T Nets, p. 23.
93/43	B.W. Watson	A taxomomy of finite automata construction algorithms, p. 87.
93/44	B.W. Watson	A taxonomy of finite automata minimization algorithms, p. 23.
93/45	E.J. Luit J.M.M. Martin	A precise clock synchronization protocol,p.
93/46	T. Kloks D. Kratsch J. Spinrad	Treewidth and Patwidth of Cocomparability graphs of Bounded Dimension, p. 14.
93/47	W. v.d. Aalst P. De Bra G.J. Houben Y. Kornatzky	Browsing Semantics in the "Tower" Model, p. 19.
93/48	R. Gerth	Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.
94/01	P. America M. van der Kammen R.P. Nederpelt O.S. van Roosmalen H.C.M. de Swart	The object-oriented paradigm, p. 28.
94/02	F. Kamareddine R.P. Nederpelt	Canonical typing and Π-conversion, p. 51.
94/03	L.B. Hartman K.M. van Hee	Application of Marcov Decision Processe to Search Problems, p. 21.
94/04	J.C.M. Baeten J.A. Bergstra	Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.
94/05	P. Zhou J. Hooman	Formal Specification and Compositional Verification of an Atomic Broadcast Protocol, p. 22.
94/06	T. Basten T. Kunz J. Black M. Coffin D. Taylor	Time and the Order of Abstract Events in Distributed Computations, p. 29.
94/07	K.R. Apt R. Bol	Logic Programming and Negation: A Survey, p. 62.

94/08	O.S. van Roosmalen	A Hierarchical Diagrammatic Representation of Class Structure, p. 22.
94/09	J.C.M. Baeten J.A. Bergstra	Process Algebra with Partial Choice, p. 16.
94/10	T. verhoeff	The testing Paradigm Applied to Network Structure. p. 31.
94/11	J. Peleska C. Huizing C. Petersohn	A Comparison of Ward & Mellor's Transformation Schema with State- & Activitycharts, p. 30.
94/12	T. Kloks D. Kratsch H. Müller	Dominoes, p. 14.
94/13	R. Seljée	A New Method for Integrity Constraint checking in Deductive Databases, p. 34.
94/14	W. Peremans	Ups and Downs of Type Theory, p. 9.
94/15	R.J.M. Vaessens E.H.L. Aarts J.K. Lenstra	Job Shop Scheduling by Local Search, p. 21.
94/16	R.C. Backhouse H. Doornbos	Mathematical Induction Made Calculational, p. 36.
94/17	S. Mauw M.A. Reniers	An Algebraic Semantics of Basic Message Sequence Charts, p. 9.
94/18	F. Kamareddine R. Nederpelt	Refining Reduction in the Lambda Calculus, p. 15.
94/19	B.W. Watson	The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.
94/20	R. Bloo F. Kamareddine R. Nederpelt	Beyond β-Reduction in Church's $\lambda\rightarrow$, p. 22.
94/21	B.W. Watson	An introduction to the Fire engine: A C++ toolkit for Finite automata and Regular Expressions.
94/22	B.W. Watson	The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.
94/23	S. Mauw and M.A. Reniers	An algebraic semantics of Message Sequence Charts, p. 43.
94/24	D. Dams O. Grumberg R. Gerth	Abstract Interpretation of Reactive Systems: Abstractions Preserving $\forall$CTL*, $\exists$CTL* and CTL*, p. 28.
94/25	T. Kloks	$K_{1,3}$-free and W_4-free graphs, p. 10.
94/26	R.R. Hoogerwoord	On the foundations of functional programming: a programmer's point of view, p. 54.

94/27	S. Mauw and H. Mulder	Regularity of BPA-Systems is Decidable, p. 14.
94/28	C.W.A.M. van Overveld M. Verhoeven	Stars or Stripes: a comparative study of finite and transfinite techniques for surface modelling, p. 20.
94/29	J. Hooman	Correctness of Real Time Systems by Construction, p. 22.
94/30	J.C.M. Baeten J.A. Bergstra Gh. Ştefanescu	Process Algebra with Feedback, p. 22.
94/31	B.W. Watson R.E. Watson	A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.
94/32	J.J. Vereijken	Fischer's Protocol in Timed Process Algebra, p. 38.
94/33	T. Laan	A formalization of the Ramified Type Theory, p.40.
94/34	R. Bloo F. Kamareddine R. Nederpelt	The Barendregt Cube with Definitions and Generalised Reduction, p. 37.
94/35	J.C.M. Baeten S. Mauw	Delayed choice: an operator for joining Message Sequence Charts, p. 15.
94/36	F. Kamareddine R. Nederpelt	Canonical typing and Π-conversion in the Barendregt Cube, p. 19.
94/37	T. Basten R. Bol M. Voorhoeve	Simulating and Analyzing Railway Interlockings in ExSpect, p. 30.
94/38	A. Bijlsma C.S. Scholten	Point-free substitution, p. 10.
94/39	A. Blokhuis T. Kloks	On the equivalence covering number of splitgraphs, p. 4.
94/40	D. Alstein	Distributed Consensus and Hard Real-Time Systems, p. 34.
94/41	T. Kloks D. Kratsch	Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.
94/42	J. Engelfriet J.J. Vereijken	Concatenation of Graphs, p. 7.
94/43	R.C. Backhouse M. Bijsterveld	Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.
94/44	E. Brinksma, J. Davies R. Gerth, S. Graf W. Janssen, B. Jonsson S. Katz, G. Lowe M. Poel, A. Pnueli C. Rump, J. Zwiers	Verifying Sequentially Consistent Memory, p. 160
94/45	G.J. Houben	Tutorial voor de ExSpect-bibliotheek voor "Administratieve Logistiek", p. 43.

94/46	R. Bloo F. Kamareddine R. Nederpelt	The λ-cube with classes of terms modulo conversion, p. 16.
94/47	R. Bloo F. Kamareddine R. Nederpelt	On Π-conversion in Type Theory, p. 12.
94/48	Mathematics of Program Construction Group	Fixed-Point Calculus, p. 11.
94/49	J.C.M. Baeten J.A. Bergstra	Process Algebra with Propositional Signals, p. 25.
94/50	H. Geuvers	A short and flexible proof of Strong Normalazation for the Calculus of Constructions, p. 27.
94/51	T. Kloks D. Kratsch H. Müller	Listing simplicial vertices and recognizing diamond-free graphs, p. 4.
94/52	W. Penczek R. Kuiper	Traces and Logic, p. 81
94/53	R. Gerth R. Kuiper D. Peled W. Penczek	A Partial Order Approach to Branching Time Logic Model Checking, p. 20.
95/01	J.J. Lukkien	The Construction of a Small Communication Library, p. 16.
95/02	M. Bezem R. Bol J.F. Groote	Formalizing Process Algebraic Verifications in the Calculus of Constructions, p. 49.
95/03	J.C.M. Baeten C. Verhoef	Concrete process algebra, p. 134.