A type inference algorithm for pure type systems

Severi, P.G.

Published: 01/01/1995

Citation for published version (APA):
A Type Inference Algorithm for Pure Type Systems

by

Paula Severi

95/05

ISSN 0926-4515

All rights reserved
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 implementations are actually correct.

In this paper we present an efficient algorithm for inferring 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 \( \Gamma \) and terms \( b \) and \( B \), is it true that \( \Gamma \vdash b : B \)?
2. Given a context \( \Gamma \) 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., \( \lambda \ast \), 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 \( \Gamma \).

A set of rules for a typing relation \( \vdash \) are called syntax directed if given a context \( \Gamma \) 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 \( \Gamma, b \mapsto B \). The algorithm to compute the corresponding function \( \Gamma, b \mapsto B \) 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 [PoI93] a type inference algorithm for bijective PTS's is presented. The class of bijective PTS's includes all systems of the \( \lambda \)-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 "Γ ⊢ (Πx: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. The following open problems are equivalent for single sorted PTS's:

- For any specification, if a PTS is $\beta$-strongly normalizing then the corresponding DPTS extended with definitions is $\beta\delta$-strongly normalizing.
- For any specification, if a PTS is $\beta$-strongly normalizing then the corresponding KPTS is $\beta k$-strongly normalizing.
- For any specification, if a PTS is $\beta$-strongly normalizing then the corresponding PTS'w is $\beta$-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 $S = (S, A, R)$ such that

- $S \subseteq C$ is the set of sorts.
- $A \subseteq C \times S$ is the set of axioms
- $R \subseteq S \times S \times S$ is the set of rules

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

$$
T \quad ::= \quad V \mid C \mid (T \ T) \mid (\lambda V:T. \ T) \mid (\Pi V:T. \ T) \\
C \quad ::= \quad \epsilon \mid <C, V:T>
$$

where $V$ is the set of variables and $C$ is the set of constants.

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

---

1 A KPTS is a PTS extended with the following typing rule: $\Gamma \vdash a : A \quad \Gamma \vdash b : B \quad \Gamma \vdash (K \ a \ b) : A$ and the following reduction rule: $(K \ a \ b) \rightarrow_\alpha a$
**Definition 2.3.** The PTS determined by the specification \( S = (S, A, R) \) is denoted as \( \lambda S = \lambda(S, A, R) \) 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:

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

where \( s \) ranges over sorts, i.e. \( s \in S \).

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

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

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

**Theorem 2.6.** (Subject Reduction Theorem) Let \( \Gamma \in C \) and \( a, A, B, d', D \in T \) be such that \( \Gamma \vdash d : D \). If \( d \rightarrow_\beta d' \) then \( \Gamma \vdash d' : D \).

**Definition 2.7.** The specification \( S = (S, A, R) \) is called singly sorted if

1. \( (c, s_1), (c, s_2) \in A \) implies \( s_1 \equiv s_2 \)

2. \((s_1, s_2, s_3), (s_1, s_2, s_3') \in R \) implies \( s_3 \equiv s_3' \)

**Theorem 2.8.** (Uniqueness of Types) Let \( S \) be a singly sorted specification, \( \Gamma \in C \) and \( a, A, B \in T \) such that \( \Gamma \vdash a : A \) and \( \Gamma \vdash a : B \). Then \( A \equiv_\beta B \).

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

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

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

<table>
<thead>
<tr>
<th>System</th>
<th>( R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda )</td>
<td>( (*, *) )</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>( (*, *) )</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>( (*, *) )</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>( (*, *) )</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>( (*, *) )</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>( (*, *) )</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>( (*, *) )</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>( (*, *) )</td>
</tr>
</tbody>
</table>
The rule $(s_1, s_2)$ is an abbreviation for $(s_1, s_2, s_2)$. 

Note that the system $\lambda C$ is the Calculus of Constructions and this system is full. All the systems of the $\lambda$-cube have only one toposort that is $\Box$.

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

$$\lambda C_\infty$$

where

- $S = N$
- $A = \{(n, n+1)| n \in \mathbb{N}\}$
- $R = \{(m, 0, 0)| m \in \mathbb{N}\} \cup \{(n, r)| m, n \in \mathbb{N} \land \max(m, n) \leq r\}$

The system $\lambda C_\infty$ extended with strong $\Sigma$-types and cumulativity is the system $ECC$ (see [Luo89]). We can see that $\lambda C_\infty$ is an extension of $\lambda C$ writing $\ast$ instead of $0$, $\Box$ instead of 1. Note that there is no toposort in $\lambda C_\infty$.

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

The system $\lambda C_\infty$ and the systems of the $\lambda$-cube are $\beta$-strongly normalizing. However not all PTS's are $\beta$-strongly normalizing as next example shows:

**Example 2.14.** The PTS $\lambda \ast$ determined by the specification $(S, A, R)$ where $S = \{\ast\}$, $A = \{(\ast, \ast)\}$ and $R = \{(\ast, \ast)\}$ is not $\beta$-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:

$$\Gamma, \pi : A \vdash_\pi b : B$$

$$\Gamma \vdash_\pi (\lambda x : A. b) : (\Pi x : A. B)$$

The notion of type derivation in $PTS^w$ will be written as $\Gamma \vdash_\pi a : A$ 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 $\Gamma \vdash_\pi a : A$. The following example shows that $\lambda S^w$ has more typable terms than $\lambda C$, i.e. $\lambda C \subseteq \lambda C^w$.

**Example 3.1.** The following term is typable in $\lambda C^w$:

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

But it is not typable in $\lambda C$ because $A \rightarrow \Box$ 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 $\lambda S$ when we do not consider the type premise of the abstraction rule.

**Notation 3.2.** From now on $\Gamma \not\vdash_\pi A : -$ will denote that $A$ is not typable in $\Gamma$, i.e. $\exists s[\Gamma \vdash_\pi A : s]$.

It will be important to consider the terms $A$ such that $\Gamma \vdash_\pi a : A$ and $\Gamma \not\vdash_\pi A : -$ and hence we define:
Definition 3.3. We say that $A$ is a toptype if there exist $\Gamma, a$ 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 $(\lambda C^w)$ we can derive

$$A : + \vdash_{C^w} \lambda x : A. (\ast \rightarrow \ast) : (A \rightarrow \Box)$$

The term $(A \rightarrow \Box)$ has no type in $\lambda C^w$ and is a toptype.

b) In $(\lambda \rightarrow)^w$ we can derive

$$\Gamma(\rightarrow) (\lambda \alpha : \ast, \lambda z : \alpha \cdot \alpha) : \Pi \alpha : \ast \rightarrow \alpha$$

The term $\Pi \alpha : \ast \rightarrow \alpha$ has no type in $(\lambda \rightarrow)^w$ and is a toptype.

Notice that the notion of toptype makes sense for both, PTS's and $\text{PTSw}^w$'s. For PTS's, the only toptypes are the topsorts. For example in $\lambda C$ 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 $\Pi x_1 : A_1 \ldots \Pi x_n : A_n : B$ with $s_0$ a topsort. For example, any toptype in $\lambda C^w$ has the form: $(\Pi x_1 : A_1, \Pi x_2 : A_2 \ldots \Pi x_n : A_n, \Box)$.

The description of toptypes for any $\text{PTSw}^w$ 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 $R$.

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 either $B$ is a topsort $s_0$

or $\exists s \in \Gamma, x_1 : A_1 \ldots x_n : A_n : B : s$

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

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

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

$$\Gamma \not\vdash_w A : - \quad \text{iff} \quad A \equiv \Pi x_1 : A_1 \ldots \Pi x_n : A_n : B$$

where either $B$ is a topsort $s_0$

or $\forall s_1, s_2 \in \Gamma, x_1 : A_1 \ldots x_n : A_n : B : s$.

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 $\beta$-closure of toptypes we have that $\Gamma \vdash_w B : s'$ for some $s'$.

We call this theorem $\beta$-closure of toptypes even though it is not completely right because we do not prove that $A'$ is inhabited.
3.2 Normalization for \( \beta \)-reduction

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

If \( \Gamma \vdash_w a : A \vdash s \) then \( \Gamma' \vdash_{\lambda S} a' : A' \) for some \( a', A', \Gamma' \)

such that \( a \rightarrow_{\beta} a', A \rightarrow_{\beta} A', \Gamma \rightarrow_{\beta} \Gamma' \).

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

Then we prove that weak normalization of \( \lambda S \) implies weak normalization of \( \lambda S^w \).

**Definition 3.9.** We say that an abstraction \( \lambda x : A. b \) is illegal w.r.t. the context \( \Gamma \) 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 \( \Gamma \) if its abstraction \( \lambda x : A. b \) is illegal.

We define a mapping \( C \) that contracts all the illegal redexes of a term.

**Definition 3.11.** Given \( \Gamma \vdash_w a : A \) we define \( C_\Gamma \) on the typable terms as follows:

\[
\begin{align*}
C_\Gamma(x) & = x \\
C_\Gamma(a \ b) & = \begin{cases} 
  a_0[x := C_\Gamma(b)] & \text{if } C_\Gamma(a) = \lambda x : A. a_0 \text{ is an illegal abstraction w.r.t. } \Gamma \\
  (C_\Gamma(a), C_\Gamma(b)) & \text{otherwise}
\end{cases} \\
C_\Gamma(\lambda x : A. a) & = (\lambda x : C_\Gamma(A). C_\Gamma(\Gamma_\Gamma(x : A. a))) \\
C_\Gamma(\Pi x : A. B) & = (\Pi x : C_\Gamma(A). C_\Gamma(\Gamma_\Gamma(x : A. B)))
\end{align*}
\]

We write \( C(a) \) instead of \( C_\Gamma(a) \).

**Lemma 3.12.**

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

**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 \( C(b[x := a]) \) may not be syntactically equal to \( C(b)[x := C(a)] \).

**Example 3.13.** The following specification is not singly sorted:

\[
\begin{array}{|c|c|}
\hline
S & 0,1,2 \\
A & 0 : 1, 0 : 2, 1 : 2 \\
R & (2,2) \\
\hline
\end{array}
\]

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 \( C(b[x := 0]) \neq C(b)[x := C(0)] \).

**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:**

6
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 \( \exists s_0 \ \Gamma, x : A \vdash_w B : s_2 \).

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

3. If \( \not\vdash_w B[x := a] : \ldots \) then \( \not\vdash_w (\Pi x : A. \ B) : \ldots \).

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 \( \not\vdash_w (\Pi x : A. \ B) : s' \). 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
   \( C(\Gamma) \vdash_{\lambda_\mathcal{S}} C(a) : C(A) \).

2. If \( A \) is a toptype then
   \[ C(\Gamma, x_1 : A_1 \ldots x_n : A_n) \vdash_{\lambda_\mathcal{S}} a' : C(B) \]
   where \( A \equiv \Pi x_1 : A_1 \ldots \Pi x_n : A_n. B \) and
   \( C(a) \equiv \lambda x_1 : C(A_1) \ldots \lambda x_n : C(A_n). a' \).

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

- (abstraction)

\[
\begin{array}{c}
\frac{\Gamma, x : A \vdash_w b : B}{\Gamma \vdash_w (\lambda x : A. \ b) : (\Pi x : A. \ B)}
\end{array}
\]

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 \( (s_1, s_2, s) \in R \).
   By the IH we have that
   \[ C(\Gamma) \vdash C(A) : s_1 \] and \( C(\Gamma, x : A) \vdash C(b) : C(B) \).
   If the specification is full by lemma 3.14 we have that \( C(\Gamma, x : A) \vdash C(B) : s'_2 \) and there exists \( s_3 \) such that \( (s_1, s_2, s_3) \in R \).
   If the specification is singly sorted by lemma 3.14, we have that \( s_2 = s'_2 \) and we know that
   \( (s_1, s_2, s) \in R \).
   In any case \( (s_1, s'_2, s_3) \in R \) for some \( s_3 \).
   We obtain the following derivation:
2. Suppose $\Gamma \vdash^w (\Pi x : A . B) : s$. There are two possibilities:

(a) $\Gamma, x : A \vdash^w B : s$.

By the IH we have that:

\[ C(\Gamma, x : A) \vdash C(b) : C(B) \]

where $C(\lambda x : A . b) \equiv (\lambda x : C(A). C(b))$

(b) $\Gamma, x : A \not\vdash^w B : s$.

It follows from the IH that

\[ C(\Gamma, x : A, x_1 : A_1 \ldots x_n : A_n) \vdash_{SA} b' : C(B') \]

where $B \equiv \Pi x_1 : A_1 \ldots \Pi x_n : A_n . B'$,

$C(b) \equiv \lambda x_1 : C(A_1) \ldots \lambda x_n : C(A_n) . b'$

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

\[ C(\Gamma) \vdash C(b) : C(\Pi x : A . B) \]

Note that $\Gamma \vdash^w A : s_1$. By the IH we have that $C(\Gamma) \vdash C(a) : C(A)$.

Hence we have the following derivation:

\[ C(\Gamma) \vdash C(b) : C(\Pi x : A . B) \quad C(\Gamma) \vdash C(a) : C(A) \]

\[ \frac{C(\Gamma) \vdash C(b) : C(B)[x := a]}{C(\Gamma) \vdash C(b[a]) : C(B)[x := C(a)]} \]

By lemma 3.12 we have that $C(B)[x := C(a)] \equiv C(B[x := a])$.

ii. Suppose $\Gamma \not\vdash^w (\Pi x : A . B) : s$.

Note that $\Gamma \vdash^w A : s_1$. By the IH we have that $C(\Gamma) \vdash C(a) : C(A)$.

By lemma 3.15 we have that $\Gamma, x : A \vdash^w B : s_2$. By the IH we have that

\[ C(\Gamma, x : A) \vdash b' : C(B') \]

with $C(b) \equiv \lambda x : A . b'$.

Due to the $\beta$-closure of toptypes we have that $C(b) \equiv \lambda x : A . b'$ is an illegal abstraction and then $C(b[a]) \equiv b'[x := C(a)]$.

By the substitution lemma,

\[ C(\Gamma) \vdash b'[x := C(a)] : C(B)[x := C(a)] \]

By lemma 3.12, we have that $C(B)[x := C(a)] \equiv C(B[x := a])$.

(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 $\Gamma \not\vdash^w B[x := a] : s$.

It follows from lemma 3.15 that $\Gamma \not\vdash^w (\Pi x : A . B) : s$.

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 $C(\Gamma) \vdash C(a) : C(A)$ and that

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

where $C(b) \equiv \lambda x : C(A) . \lambda x_1 : C(A_1) \ldots \lambda x_n : C(A_n) . b'$

Due to the $\beta$-closure of toptypes we have that $C(b)$ is an illegal abstraction and then the value of $C(b[a])$ is computed as follows:
\[ C(b) \equiv \lambda x_1 : C(A_1)[x := C(a)] \ldots \lambda x_n : C(A_n)[x := C(a)].b'[x := C(a)] \]

By the substitution lemma we have that:
\[ C(\Gamma), x_1 : C(A_1)[x := C(a)], \ldots, x_n : C(A_n)[x := C(a)] \vdash_{\lambda S} b'[x := C(a)] : C(B_0)[x := C(a)] \]

By lemma 3.12 we have that: \[ C(B_0)[x := C(a)] = C(B_0[x := a]) \]

• (conversion) \[ \frac{\Gamma \vdash_w b : B \quad \Gamma \vdash_w A : s \quad B =_{\beta} A}{\Gamma \vdash_w b : A} \]

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

Q.E.D.

In \([vBJ93]\) the set \( T \) is partitioned into sets \( T_V \) and \( T_S \) such that terms in \( T_V \) have a unique type and terms in \( T_S \) may have more than one type in a PTS. The same property holds for PTS\(^w\)'s.

**Definition 3.17.**
\[
\begin{align*}
T_V & := V \mid (T_V \ T_V) \mid (\lambda V. : T_V. \ T_V) \\
T_S & := C \mid (T_S \ T_S) \mid (\lambda V. : T_S. \ T_S) \mid (\Pi V. : T_S. \ T_S)
\end{align*}
\]

**Theorem 3.18.** Let \( a \in T_S \).
\[
\Gamma \vdash_w a : A, \quad A \rightarrow_{\beta} \Pi x_1 : A_1 \ldots x_n : A_n. s \\
\Gamma \vdash a : \Pi x_1 : A_1 \ldots x_n : A_n. s'
\]

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 19.** If \( \Gamma \vdash_w A : s \) and \( \Gamma \vdash A : s' \) then \( \Gamma \vdash A : s \).

**Proof:**
Suppose \( A \in T_V \). By the uniqueness of types theorem for PTS\(^w\)'s, we have that \( s \equiv s' \).
Suppose \( A \in T_S \). By the previous theorem we have that \( \Gamma \vdash A : s \).

Q.E.D.

**Corollary 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 \( \exists s \in S[\Gamma \vdash_w A : s] \) 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

Q.E.D.
\[ \Gamma', x_1: A'_1, \ldots, x_n: A'_n \vdash_{\lambda S} a' : B \]

where \( A \equiv \Pi x_1: A_1 \ldots \Pi x_n: A_n. B \),
\[ a \rightarrow_{\beta} \lambda x_1: A'_1 \ldots \lambda x_n: A'_n. a' \]
\[ \Gamma \rightarrow_{\beta} \Gamma', B \rightarrow_{\beta} B', A_i \rightarrow_{\beta} A'_i \text{ for all } i. \]

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 \( \Gamma \vdash_{\lambda S} a : A \). We consider only 2 cases:

- (abstraction) \[ \frac{\Gamma, x : A \vdash_{w} b : B}{\Gamma \vdash_{w} (\lambda x : A. b) : (\Pi x : A. B)} \]

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 \( (s_1, s_2, s) \in R \)
   By the IH and corollary 3.20 we have that
   \[ \Gamma', x : A' \vdash_{w} b' : B' : s_2 \]
   for \( b \rightarrow_{\beta} b', B \rightarrow_{\beta} B' \) and \( \Gamma, x : A \vdash_{\beta} \Gamma', x : A' \).

   By the IH we have that \( \Gamma' \vdash A' : s_1 \).

   Since \( (s_1, s_2, s) \in R \) we obtain the following derivation:

   \[ \frac{\Gamma', x : A' \vdash_{w} b' : B' \quad \Gamma' \vdash A' : s_1 \quad \Gamma', x : A' \vdash_{w} b' : B'}{\Gamma' \vdash (\Pi x : A'. B') : s} \]

2. Suppose \( \Gamma \vdash_{w} (\Pi x : A. B) : s \). There are two possibilities:

   (a) \( \Gamma, x : A \vdash_{w} B : s \).
      By the IH we have that:
      \[ \Gamma', x : A' \vdash_{w} b' : B' : s \]
      where \( \Gamma, x : A \vdash_{\beta} \Gamma', x : A' \),
      \[ b \rightarrow_{\beta} b', B \rightarrow_{\beta} B' \]

   (b) \( \Gamma, x : A \vdash_{w} B : s \).
      It follows from the IH that
      \[ \Gamma', x : A', x_1 : A'_1, \ldots, x_n : A'_n \vdash_{\lambda S} b' : B' \]
      where \( B \equiv \Pi x_1 : A_1 \ldots \Pi x_n : A_n. B_0 \),
      \[ b \rightarrow_{\beta} \lambda x_1 : A'_1 \ldots \lambda x_n : A'_n. b' \]
      \[ \Gamma \rightarrow_{\beta} \Gamma' \]
      \[ B_0 \rightarrow_{\beta} B' \]
      \[ A_i \rightarrow_{\beta} A'_i \text{ for all } i. \]

- (application) \[ \frac{\Gamma \vdash_{w} b : (\Pi x : A. B) \quad \Gamma \vdash a : A}{\Gamma \vdash_{w} (b \ a) : B[x := a]} \]

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
      \[ \Gamma' \vdash (\Pi x : A'. B') \text{ for } b \rightarrow_{\beta} b', A \rightarrow_{\beta} A', B \rightarrow_{\beta} B' \text{ and } \Gamma \rightarrow_{\beta} \Gamma' \]
      Note that \( \Gamma \vdash_{w} A : s_1 \).
      By the IH we have that
      \[ \Gamma' \vdash a' : s'_1 \text{ for } A \rightarrow_{\beta} A' \text{ and } a \rightarrow_{\beta} a'. \]
      Hence we have the following derivation:

      \[ \frac{\Gamma' \vdash (b' \ a') : B'[x := a']}{\Gamma' \vdash_{w} (b' \ a') : B'[x := a']} \]

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 \( \Gamma'' \) and \( \Gamma'' \). We omit this kind of details.
(b) Suppose \( \Gamma \vdash (\Pi x : A. B) : s \). 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

\[
\exists n, s_1, s_2 \quad \Gamma, x : A, x_1 : A_1 \ldots x_{n-1} : A_{n-1} \vdash_w A_n : s_1 \\
\Gamma, x : A, x_1 : A_1 \ldots x_{n-1} : A_{n-1}, x_n : A_n \vdash_w B_0 : s_2
\]

Moreover, we have that:

\[
b \beta \lambda x : A'. \lambda x_1 : A_1' \ldots \lambda x_n : A_n'. b' \text{ and } \Gamma', x : A', x_1 : A_1', \ldots, x_n : A_n' \vdash b' : B'
\]

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

\[
B[x := a] \equiv \Pi x_1 : A_1[x := a] \ldots x_n : A_n[x := a]. B_0[x := a]
\]

\[
\rightarrow \quad \Pi x_1 : A_1'[x := a'] \ldots x_n : A_n'[x := a']. B'[x := a']
\]

By the subject reduction theorem we have that:

\[
\Gamma' \vdash \Pi x_1 : A_1'[x := a'] \ldots x_n : A_n'[x := a']. B'[x := a'] : s
\]

We can easily construct a derivation of

\[
\Gamma' \vdash \Pi x_1 : A_1'[x := a'] \ldots x_n : A_n'[x := a']. B'[x := a'] : s \text{ in a PTS.}
\]

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

\[
\Gamma' \vdash (\lambda x_1 : A_1'[x := a'] \ldots \lambda x_n : A_n'[x := a']). (\Pi x_1 : A_1' \ldots x_n : A_n'. B')[x := a']
\]

with \( (b \ a) \rightarrow_\beta (\lambda x : A'. \lambda x_1 : A_1' \ldots \lambda x_n : A_n'. b') a' \)

\[
\rightarrow_\beta \lambda x_1 : A_1' \ldots \lambda x_n : A_n'. b'[x := a']
\]

2. Suppose \( \Gamma \vdash_w B[x := a] : s \).

It follows from lemma 3.15 that \( \Gamma \vdash_w (\Pi x : A. B) : s \).

By description theorem we have that \( (\Pi x : A. B) \equiv \Pi x_1 : A_1 \ldots \Pi x_n : A_n. B_0 \).

By the IH we have that

\[
\Gamma', x : A', x_1 : A_1', \ldots, x_n : A_n' \vdash_{\lambda s} b' : B'
\]

where \( b \beta \lambda x_1 : A_1 \ldots \lambda x_n : A_n. y \)

\[
\Gamma, x : A, x_1 : A_1, \ldots, x_n : A_n \rightarrow_\beta \Gamma', x_1 : A_1', \ldots, x_n : A_n', B_0 \rightarrow_\beta B'
\]

By the IH we have that

\[
\Gamma' \vdash b' : A'
\]

Then

\[
(b \ a) \rightarrow_\beta (\lambda x_1 : A_1 \ldots \lambda x_n : A_n. b')a' \\
\rightarrow_\beta \lambda x_1 : A_1[x := a'] \ldots \lambda x_n : A_n[x := a']. b'[x := a'] \\
\rightarrow_\beta \lambda x_1 : A_1'[x := a'] \ldots \lambda x_n : A_n'[x := a']. b'[x := a']
\]

By the substitution lemma we have that:

\[
\Gamma', x_1 : A_1'[x := a'], \ldots, x_n : A_n'[x := a'] \vdash_{\lambda s} b'[x := a'] : B'[x := a']
\]

\[
\begin{array}{c}
\text{(conversion)} \\
\hline
\Gamma \vdash_w b : B \\
\Gamma \vdash_w A : s \\
\hline
\Gamma \vdash_w b : A
\end{array}
\]

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 $\Gamma \vdash_w B : \alpha$. 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

$$\exists n, s_1, s_2 \quad \Gamma, x_1 : A_1 \ldots x_{n-1} : A_{n-1} . L_w A_n : s_1$$
$$\Gamma, x_1 : A_1 \ldots x_{n-1} : A_{n-1} , x_n : A_n . L_w B_0 : s_2$$

By the IH we have that:

$$\Gamma', x_1 : A'_1 \ldots x_n : A'_n . B'$$

with $B_0 \rightarrow^\beta B'$ 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:

$$\Gamma', x_1 : A_1'' \ldots x_n : A_n'' . B''$$

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 $\lambda S$ is weakly normalizing then $\lambda S^w$ 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 $\Gamma', a'$ and $A'$ such that $\Gamma' \vdash a' : A'$ and $a \rightarrow^\beta a'$. Since $\lambda S$ is weakly normalizing, we have that $a' \rightarrow^\beta n f(a')$ where $n f(a')$ is the $\beta$-normal form of $a'$. Then $a$ is weakly normalizing.

2. Suppose $\Gamma \vdash_w A : s$. By previous theorem we have that

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

where

$$A \equiv \Pi x_1 : A_1 \ldots \Pi x_n : A_n . B,$$

$$a \rightarrow^\beta \lambda x_1 : A'_1 \ldots \lambda x_n : A_n . a'$$

$$\Gamma \rightarrow^\beta \Gamma'$$

$$B \rightarrow^\beta B'$$

$$A_i \rightarrow^\beta A'_i \text{ for all } i.$$  

Since $\lambda S$ is weakly normalizing in particular $a'$ and $A'_i$ for all $i$ are weakly normalizing. Hence

$$a \rightarrow^\beta \lambda x_1 : A'_1 \ldots \lambda x_n : A_n . a'$$

$$\rightarrow^\beta \lambda x_1 : n f(A'_1) \ldots \lambda x_n : n f(A'_n) . n f(a')$$

$$\equiv n f(a)$$

Then $a$ is weakly normalizing.

Q. E. D.

Corollary 3.23. Let $\lambda S$ be a PTS extending $\lambda 2$.

If there is a proof of $\Pi \alpha : * . \alpha$ in $\lambda S^w$ then there is also a proof of $\Pi \alpha : * . \alpha$ in $\lambda S$.

Proof: Suppose there exist $\Gamma$ 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 $\Gamma'$ and $p'$ such that $\Gamma' \vdash_w p' : \Pi \alpha : * . \alpha$.

Q. E. D.
### 3.3 Strong Normalization for $\beta$-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 completion of $S$, strong normalization of $\lambda S'$ implies strong normalization of $\lambda S^w$.

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

**Definition 3.24.** Let $S = (S, A, R)$ and $S' = (S', A', R')$ be such that

1. $S \subseteq S'$, $A \subseteq A'$, and $R \subseteq R'$
2. $S'$ is full, i.e. for all $s_1, s_2 \in S'$ there exists $s_3$ such that $(s_1, s_2, s_3) \in R$.
3. for all $s \in S$ there is a sort $s' \in S'$ such that $(s, s') \in A'$ (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 $\lambda C_{\infty}$ is a completion of $\lambda C$ and of itself.

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

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

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

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

**Corollary 3.28.** Let $S$ be a completion of itself.

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

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 $\lambda C_{\infty}$.

Next we will prove that the set of legal terms in $\lambda S$ is equal to the set of legal terms in $\lambda S^w$ if and only if $S$ is a completion of itself. We refer to the set of legal terms of $\lambda S$ 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 $\vdash_{w} \lambda x : c \cdot (c \rightarrow s_0)$. Since $\mathcal{L}(\lambda S) = \mathcal{L}(\lambda S^w)$, we have that

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

Hence $s_0$ cannot be a topsort.

Given the sorts $s_1, s_2$ we prove that there exists a sort $s$ such that $(s_1, s_2, s) \in R$. 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:

$$\vdash_{w} (\lambda \alpha : s_1. \lambda \beta : s_2. \lambda y : \beta. \lambda x : \alpha. y) : \Pi \alpha : s_1. \Pi \beta : s_2. (\alpha \rightarrow \beta)$$

The type of this term is a legal term in $\lambda S$ then:

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

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

**Q. E. D.**

**Theorem 3.30.** Strong Normalization

Let $S'$ be a completion of $S$.

If $\lambda S'$ is $\beta$-strongly normalizing then $\lambda S^w$ is $\beta$-strongly normalizing too.
### 3.4 Strong Normalization for the illegal reduction

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

**Definition 3.31.** Given $\Gamma$ such that $\Gamma \vdash_w (\lambda x:A. b) : D$.

We define the illegal reduction w.r.t. the context $\Gamma$ as:

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

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. $(\lambda x:A.C[z d]) (\lambda y:B.e) \rightarrow_{{\beta}} C'[\lambda y:B.e]d'$ 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 infinite $\beta$-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 $z$ 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 $\rightarrow_{{\beta}}$ 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 $C$.

### 3.5 A Type Inference Algorithm for $\text{PTS}^w$

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 $\Gamma$ the term $B$ such that $\Gamma \vdash_w b : B$ is not unique.

**Notation 3.33.** We write $\Gamma \vdash_w a : \rightarrow A$ for $\Gamma \vdash_w a : A_0$ and $A_0 \rightarrow A$. We write $\Gamma \vdash_w A : \rightarrow_{{wh}} A$ for $\Gamma \vdash_w a : A_0$ and $A_0$ weak-head reduces to $A$.

**Definition 3.34.** The Syntax Directed Pure Type System $\text{PTS}_{\text{nd}}^w$ determined by the specification $S = (S, A, R)$ is denoted as $\lambda S_{\text{nd}}^w$ and defined by the notion of type derivation $\Gamma \vdash_w b : B$ given by the following axioms and rules:

---

14
\[\begin{align*}
\text{(axiom)} & & c \vdash_{\text{unif}} c : s & \text{for } (c, s) \in A \\
\text{(start)} & & \Gamma \vdash_{\text{unif}} A : \rightsquigarrow s \\
& & \Gamma, x : A \vdash_{\text{unif}} z : A & \text{where } z \text{ is } \Gamma\text{-fresh} \\
\text{(weakening)} & & \Gamma \vdash_{\text{unif}} b : B, \Gamma \vdash_{\text{unif}} A : \rightsquigarrow s \\
& & \Gamma, z : A \vdash_{\text{unif}} \overline{z} : A & \text{where } z \text{ is } \Gamma\text{-fresh and } b \in C \cup V \\
\text{(formation)} & & \Gamma, z : A \vdash_{\text{unif}} B : \rightsquigarrow s_1 \\
& & \Gamma, z : A \vdash_{\text{unif}} B : s_2 & \text{for } (s_1, s_2, s_3) \in R \\
\text{(abstraction)} & & \Gamma, z : A \vdash_{\text{unif}} b : B \\
& & \Gamma \vdash_{\text{unif}} (\lambda z : A. b) : (\Pi z : A. B) & \text{where } s \text{ ranges over sorts, i.e. } s \in S. \\
\text{(application)} & & \Gamma, z : A \vdash_{\text{unif}} b : B \\
& & \Gamma \vdash_{\text{unif}} (\Pi z : A. B) : s_3 \\
& & \Gamma \vdash_{\text{unif}} (\lambda z : A. b) : (\Pi z : A. B) \\
& & \Gamma \vdash_{\text{unif}} (\lambda z : A. b) : (\Pi z : A. B) \\
& & \Gamma \vdash_{\text{unif}} (\Pi z : A. B) : s_3 & \text{for } (s_1, s_2, s_3) \in R \\
& & \Gamma \vdash_{\text{unif}} (\lambda z : A. b) : (\Pi z : A. B) \\
& & \Gamma \vdash_{\text{unif}} (\Pi z : A. B) : s_3 & \text{for } (s_1, s_2, s_3) \in R \\
& & \Gamma \vdash_{\text{unif}} (\lambda z : A. b) : (\Pi z : A. B) \\
& & \Gamma \vdash_{\text{unif}} (\Pi z : A. B) : s_3 & \text{for } (s_1, s_2, s_3) \in R. \\
\end{align*}\]

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 \( \Gamma \vdash_{\text{unif}} a : A \) then \( \Gamma \vdash_w a : A \).

**Theorem 3.36. (Completeness)** If \( \Gamma \vdash_w a : A \) then \( \Gamma \vdash_{\text{unif}} a : A' \) for \( A \equiv \beta A' \).

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

**Primitives**

\[
\begin{align*}
N F : T \rightarrow T & \text{ computes the normal form,} \\
WHNF : T \rightarrow T & \text{ computes the weak-head normal form,} \\
EQ : T \times T \rightarrow \text{Bool} & \text{ yields true if two terms are } \beta\text{-equal.}
\end{align*}
\]

**Type Inference Algorithm for arbitrary PTS\(^w\)'s**

We define a mapping \( \text{INF}_w : C \times T \rightarrow P(T) \) as follows:

\[
\begin{align*}
\text{INF}_w(\varepsilon; c) & = \{ s \mid (c, s) \in A \} \\
\text{INF}_w(< \Gamma, x : A >; x) & = \{ A \} \text{ if } s \in NF(\text{INF}_w(< \Gamma; A >)) \\
\text{INF}_w(< \Gamma, x : A >; y) & = \text{INF}_w(\Gamma; y) \text{ if } [s \in NF(\text{INF}_w(\Gamma; A))] \land y \neq x \\
\text{INF}_w(< \Gamma, x : A >; c) & = \text{INF}_w(\Gamma; c) \text{ if } (s \in NF(\text{INF}_w(\Gamma; A))) \\
\text{INF}_w(\Gamma; (\Pi x : A. B)) & = \{ s_3 \mid [s_1 \in NF(\text{INF}_w(\Gamma; A))] \land [s_2 \in NF(\text{INF}_w(\Gamma, x : A; B))] \\
& \land (s_1, s_2, s_3) \in R \} \\
\text{INF}_w(\Gamma; (\lambda x : A. b)) & = \{ (\Pi x : A. B) \mid B \in INF_w(\Gamma, x : A; b) \} \\
\text{INF}_w(\Gamma; (b \ a)) & = \{ B[x := a] \mid (\Pi x : A. B) \in WHNF(\text{INF}_w(\Gamma; b)) \land \\
& [A' \in INF_w(\Gamma; a)] \land EQ(A; A') \}
\end{align*}
\]

Note that if the term \( a \) has no type under the context \( \Gamma \) then \( \text{INF}_w(\Gamma, a) = \emptyset \).

In case of singly sorted PTS's we write \( \text{INF}_w(\Gamma, a) = A \) instead of \( \text{INF}_w(\Gamma, a) = \{ A \} \).

**Theorem 3.37.** Given \( \Gamma \in C \) and \( a \in T \), \( \text{INF}_w(\Gamma, a) = \{ A \mid \Gamma \vdash_w a : A \} / \equiv_\beta \).

**Proof:** It is proved by induction on \( (\Gamma, a) \).
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 unsuccessful. 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_{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 \( \Gamma, a \), 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 \( \Gamma \) 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 \( \Gamma \vdash_{sd} a : A \) for \( \Gamma \vdash_{sd} a : A_{0} \) and \( A_{0} \rightarrow_{\rho} A \).
We write \( \Gamma \vdash_{sd} A : \Rightarrow_{w} A \) 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 \( S = (S, A, R) \) is denoted as \( \lambda S_{sd} \) and defined by the notion of type derivation \( \Gamma \vdash_{sd} b : B \) given by the following axioms and rules:

\[
\begin{align*}
(\text{axiom}) & \quad \varepsilon \vdash_{sd} c : s \quad \text{for } (c, s) \in A \\
(\text{start}) & \quad \Gamma, x : A \vdash_{sd} x : A \\
(\text{weakening}) & \quad \Gamma \vdash_{sd} b : B, \Gamma \vdash_{sd} A : \Rightarrow_{s} \quad \Gamma, x : A \vdash_{sd} b : B \quad \text{where } x \text{ is } \Gamma\text{-fresh} \\
(\text{formation}) & \quad \Gamma \vdash_{sd} A : \Rightarrow_{s_{1}}, \Gamma, x : A \vdash_{sd} B : \Rightarrow_{s_{2}} \quad \Gamma \vdash_{sd} (\Pi x : A. B) : s_{3} \\
(\text{abstraction}) & \quad \Gamma \vdash_{sd} (\Pi x : A. B) : s \\
(\text{application}) & \quad \Gamma \vdash_{sd} b : \Rightarrow_{w} (\Pi x : A. B), \Gamma \vdash_{sd} a : A' \quad A' =_{\rho} A \\
\end{align*}
\]
where $s$ ranges over sorts, i.e. $s \in S$.

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

- The weakening 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, $\text{PTS}_{sd}$ is syntax directed.

It is easy to prove Soundness of these systems with respect to $\text{PTS}$'s:

**Theorem 4.3. (Soundness)**

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

Next example shows that completeness is not true for non-singly sorted $\text{PTS}$'s.

**Example 4.4.** The following specification is not singly sorted:

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

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

For singly sorted $\text{PTS}$'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 $A \equiv_{\beta} A'$ 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 $\vdash_{n_f}$.

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:

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

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

We called these systems $\lambda S_{n_f}$ 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 $\Gamma \vdash_{n_f} a : A$ then $\Gamma \vdash a : A$.

**Theorem 4.6. (Completeness)** If $\Gamma \vdash a : A$ then $\Gamma \vdash_{n_f} a : A'$ for $A \equiv_{\beta} A'$.

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.

However 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 $\text{PTS}_{\text{ned}}$, we obtain a type inference algorithm for singly sorted $\text{PTS}$'s.

First notice that the mapping $C$ defined in 3.11 can be adapted to the system $\text{PTS}_{\text{ned}}$. We call this mapping $C'$. This mapping contracts the illegal redexes of a term with respect to $\lambda S_{\text{ned}}^w$. The notions of illegal abstraction and illegal redex for $\lambda S_{\text{ned}}^w$ 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 $C'_{\Gamma}$ on the typable terms as follows:

$$C'_{\Gamma}(x) = x$$

$$C'_{\Gamma}(a b) = \begin{cases} 
\alpha_0[x := C'_{\Gamma}(b)] & \text{if } (C'_{\Gamma}(a) = \lambda x : A . a_0) \land (\lambda x : A . a_0) = B) \land (\lambda x : A . a_0) = \emptyset) \\
(C'_{\Gamma}(a) C'_{\Gamma}(b)) & \text{otherwise}
\end{cases}$$

$$C'_{\Gamma}(\lambda x : A . a) = (\lambda x : C'_{\Gamma}(a)) \land (\lambda x : C'_{\Gamma}(a))$$

$$C'_{\Gamma}(\Pi x : A . B) = (\Pi x : C'_{\Gamma}(a)) \land (\Pi x : C'_{\Gamma}(a))$$

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

Type Inference Algorithm

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

$$\text{INF}(\Gamma, a) = \begin{cases} 
\text{let } A = \text{INF}(\Gamma, a) \text{ in } \\
\text{if } (A = s \lor \text{NF}(\text{INF}(\Gamma, a)) = s) \land (C'(a) = a) \land (C'(\Gamma) = \Gamma) \\
\text{then } C'(A) \\
\text{else } \bot
\end{cases}$$

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 $C$ and $C'$ are the same function on the typable terms of $\lambda S_{\text{ned}}^w$:

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

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

Suppose $a \equiv (d e)$ and $C'(a) = (\lambda x : A . f)$ 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. $(\lambda S)^w_{\text{ned}}$.

**Theorem 4.8.** (Soundness) Let $S$ be a full or functional specification.

If the following conditions hold:

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

then $\Gamma \vdash a : A$. 

18
Proof: By theorem 3.35 we have that $\Gamma \vdash \omega a : A$. By theorem 3.16 we have that $C(\Gamma) \vdash C(a) : C(A)$. By previous lemma, $C'(\Gamma) \vdash C'(a) : C'(A)$. Hence $\Gamma \vdash a : C'(A)$.

Q. E. D.

Theorem 4.9. (Completeness)

If $\Gamma \vdash a : A$ then the following holds:

1. $\Gamma \vdash \text{wnd} a : A$
2. $\Gamma \vdash \text{wnd} A : \rightarrow s$ or $A \equiv s$
3. $C'(\Gamma) = \Gamma$, $C'(a) = a$ and $C'(A) = A$.

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


## Contents

1 Introduction .................................. 1

2 Pure Type Systems ............................. 2

3 Pure Type Systems with Weakened Abstraction Rule .. 4
   3.1 Description of Toptypes .................. 4
   3.2 Normalization for $\beta$-reduction ...... 6
   3.3 Strong Normalization for $\beta$-reduction .. 13
   3.4 Strong Normalization for the illegal reduction . 14
   3.5 A Type Inference Algorithm for PTS\^w ..... 14

4 Typability for PTS's ............................... 16
   4.1 The simplest Type Inference Algorithm .... 16
   4.2 A Simple But Inefficient Type Inference Algorithm .. 17
   4.3 A More efficient Type Inference Algorithm . 18
Computing Science Reports

In this series appeared:

93/01 R. van Geldrop
93/02 T. Verhoeff
93/03 T. Verhoeff
93/04 E.H.L. Aarts
  J.H.M. Korst
  PJ. Zwietering
93/05 J.C.M. Baeten
  C. Verhoef
93/06 J.P. Veltkamp
93/07 P.D. Moerland
93/08 J. Verhoosel
93/09 K.M. van Hee
93/10 K.M. van Hee
93/11 K.M. van Hee
93/12 K.M. van Hee
93/13 K.M. van Hee
93/14 J.C.M. Baeten
  J.A. Bergstra
93/15 J.C.M. Baeten
  J.A. Bergstra
  R.N. Bol
93/16 H. Schepers
  J. Hooman
93/17 D. Alstein
  P. van der Stok
93/18 C. Verhoef
93/19 G-J. Houben
93/20 F.S. de Boer

Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
A continuous version of the Prisoner's Dilemma, p. 17
Quicksort for linked lists, p. 8.
Deterministic and randomized local search, p. 78.
A congruence theorem for structured operational semantics with predicates, p. 18.
On the unavoidability of metastable behaviour, p. 29
Exercises in Multiprogramming, p. 97
A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
The Design of an Online Help Facility for ExSpect, p. 21.
<table>
<thead>
<tr>
<th>Code</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>93/22</td>
<td>E. Poll</td>
<td>A Typechecker for Bijective Pure Type Systems, p. 28.</td>
</tr>
<tr>
<td>93/23</td>
<td>E. de Kogel</td>
<td>Relational Algebra and Equational Proofs, p. 23.</td>
</tr>
<tr>
<td>93/24</td>
<td>E. Poll, Paula Severi</td>
<td>Pure Type Systems with Definitions, p. 38.</td>
</tr>
<tr>
<td>93/26</td>
<td>W.M.P. van der Aalst</td>
<td>Multi-dimensional Petri nets, p. 25.</td>
</tr>
<tr>
<td>93/27</td>
<td>T. Kloks, D. Kratsch</td>
<td>Finding all minimal separators of a graph, p. 11.</td>
</tr>
<tr>
<td>93/28</td>
<td>F. Kamareddine, R. Nederpelt</td>
<td>A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.</td>
</tr>
<tr>
<td>93/29</td>
<td>R. Post, P. De Bra</td>
<td>GOLD, a Graph Oriented Language for Databases, p. 42.</td>
</tr>
<tr>
<td>93/30</td>
<td>J. Deogun, T. Kloks, D. Kratsch, H. Müller</td>
<td>On Vertex Ranking for Permutation and Other Graphs, p. 11.</td>
</tr>
<tr>
<td>93/31</td>
<td>W. Körver</td>
<td>Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.</td>
</tr>
<tr>
<td>93/33</td>
<td>L. Loyens, J. Moonen</td>
<td>ILIAS, a sequential language for parallel matrix computations, p. 20.</td>
</tr>
<tr>
<td>93/34</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Real Time Process Algebra with Infinitesimals, p. 39.</td>
</tr>
<tr>
<td>93/36</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Non Interleaving Process Algebra, p. 17.</td>
</tr>
<tr>
<td>93/38</td>
<td>C. Verhoeft</td>
<td>A general conservative extension theorem in process algebra, p. 17.</td>
</tr>
<tr>
<td>93/39</td>
<td>W.P.M. Nuijten, E.H.L. Aarts</td>
<td>Job Shop Scheduling by Constraint Satisfaction, p. 22.</td>
</tr>
<tr>
<td>93/40</td>
<td>P.D.V. van der Stok</td>
<td>A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.</td>
</tr>
<tr>
<td>93/41</td>
<td>A. Bijlsma</td>
<td>Temporal operators viewed as predicate transformers, p. 11.</td>
</tr>
<tr>
<td>93/42</td>
<td>P.M.P. Rambags</td>
<td>Automatic Verification of Regular Protocols in P/T Nets, p. 23.</td>
</tr>
<tr>
<td>93/43</td>
<td>B.W. Watson</td>
<td>A taxonomy of finite automata construction algorithms, p. 87.</td>
</tr>
<tr>
<td>93/44</td>
<td>B.W. Watson</td>
<td>A taxonomy of finite automata minimization algorithms, p. 23.</td>
</tr>
<tr>
<td>93/48</td>
<td>R. Gerth</td>
<td>Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.</td>
</tr>
<tr>
<td>94/01</td>
<td>P. America M. van der Kammen R.P. Nederpelt O.S. van Roosmalen H.C.M. de Swart</td>
<td>The object-oriented paradigm, p. 28.</td>
</tr>
<tr>
<td>94/02</td>
<td>F. Kamareddine R.P. Nederpelt</td>
<td>Canonical typing and Π-conversion, p. 51.</td>
</tr>
<tr>
<td>94/04</td>
<td>J.C.M. Baeten J.A. Bergstra</td>
<td>Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.</td>
</tr>
<tr>
<td>94/06</td>
<td>T. Basten T. Kunz J. Black M. Coffin D. Taylor</td>
<td>Time and the Order of Abstract Events in Distributed Computations, p. 29.</td>
</tr>
<tr>
<td>ID</td>
<td>Author(s)</td>
<td>Title</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>94/08</td>
<td>O.S. van Roosmalen</td>
<td>A Hierarchical Diagrammatic Representation of Class Structure, p. 22.</td>
</tr>
<tr>
<td>94/09</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Process Algebra with Partial Choice, p. 16.</td>
</tr>
<tr>
<td>94/10</td>
<td>T. verhoeff</td>
<td>The testing Paradigm Applied to Network Structure, p. 31.</td>
</tr>
<tr>
<td>94/13</td>
<td>R. Seljée</td>
<td>A New Method for Integrity Constraint checking in Deductive Databases, p. 34.</td>
</tr>
<tr>
<td>94/14</td>
<td>W. Peremans</td>
<td>Ups and Downs of Type Theory, p. 9.</td>
</tr>
<tr>
<td>94/16</td>
<td>R.C. Backhouse, H. Doombos</td>
<td>Mathematical Induction Made Calculational, p. 36.</td>
</tr>
<tr>
<td>94/18</td>
<td>F. Kamareddine, R. Nederpelt</td>
<td>Refining Reduction in the Lambda Calculus, p. 15.</td>
</tr>
<tr>
<td>94/19</td>
<td>B.W. Watson</td>
<td>The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.</td>
</tr>
<tr>
<td>94/20</td>
<td>R. Bloo, F. Kamareddine, R. Nederpelt</td>
<td>Beyond β-Reduction in Church’s λ→, p. 22.</td>
</tr>
<tr>
<td>94/22</td>
<td>B.W. Watson</td>
<td>The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.</td>
</tr>
<tr>
<td>94/23</td>
<td>S. Mauw and M.A. Reniers</td>
<td>An algebraic semantics of Message Sequence Charts, p. 43.</td>
</tr>
<tr>
<td>94/25</td>
<td>T. Kloks</td>
<td>K₁,₃-free and W₄-free graphs, p. 10.</td>
</tr>
<tr>
<td>94/26</td>
<td>R.R. Hoogerwoord</td>
<td>On the foundations of functional programming: a programmer’s point of view, p. 54.</td>
</tr>
</tbody>
</table>
Correctness of Real Time Systems by Construction, p. 22.
Process Algebra with Feedback, p. 22.
A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.
A formalization of the Ramified Type Theory, p. 40.
The Barendregt Cube with Definitions and Generalised Reduction, p. 37.
Delayed choice: an operator for joining Message Sequence Charts, p. 15.
Canonical typing and Π-conversion in the Barendregt Cube, p. 19.
Simulating and Analyzing Railway Interlockings in ExSpect, p. 30.
Point-free substitution, p. 10.
On the equivalence covering number of splitgraphs, p. 4.
Distributed Consensus and Hard Real-Time Systems, p. 34.
Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.
Concatenation of Graphs, p. 7.
Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.
Verifying Sequentially Consistent Memory, p. 160
Tutorial voor de ExSpect-bibliotheek voor "Administratieve Logistiek", p. 43.
94/46 R. Bloo
F. Kamareddine
R. Nederpelt

The \(\lambda\)-cube with classes of terms modulo conversion,

p. 16.

94/47 R. Bloo
F. Kamareddine
R. Nederpelt

On \(\Pi\)-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 Normalization 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.