Parameters in Pure Type Systems

Roel Bloo[⋆], Fairouz Kamareddine[⋆⋆], Twan Laan[⋆⋆⋆], and Rob Nederpelt[†]

Abstract. In this paper we study the addition of parameters to typed λ-calculus with definitions. We show that the resulting systems have nice properties and illustrate that parameters allow for a better fine-tuning of the strength of type systems as well as staying closer to type systems used in practice in theorem provers and programming languages.

1 What are parameters?

Parameters occur when functions are only allowed to occur when provided with arguments. As we will show below, both in mathematics and in programming languages the use of parameters is abundant and closely connected to the use of constants and definitions. If we want to be able to use type systems in accordance with practice and yet described in a precise manner, we therefore need parameters, constants, and definitions in type theory as well.

Parameters, constants and definitions in theorem proving It is interesting to note that the first tool for mechanical representation and verification of mathematical proofs, AUTOMATH, already has a combined constant, definition and parameter mechanism and was developed from the viewpoint of mathematicians (see [7]). The representation of a mathematical text in AUTOMATH consists of a finite list of *lines* where every line has the format: $x_1 : A_1, \ldots, x_n : A_n \vdash g(x_1, \ldots, x_n) = t : T$. Here g is a new name; a constant if t is the acronym 'primitive notion', or an abbreviation (definition) for the expression t of type T, and $x_1, \ldots, x_n$ are the parameters of g, with respective types $A_1, \ldots, A_n$. Use of the definition g in the rest of the list of lines is only allowed when g is supplied with a list of arguments $t_1, \ldots, t_n$ (with types conforming to $A_1, \ldots, A_n$, see again [7]) to be substituted for the parameters of g.

We see that parameters and definitions are a very substantial part of AUTOMATH since each line introduces a new constant or definition which is *inherently parameterized* by the variables occurring in the context needed for it. Actual development of ordinary mathematical theory in the AUTOMATH system by e.g. van Benthem Jutting (cf. [3]) revealed that this combined definition and parameter mechanism is vital for keeping proofs manageable and sufficiently readable for humans.

[⋆] Eindhoven University of Technology, Computing science, PO-Box 513, 5600 MB Eindhoven, The Netherlands, email: c.j.bloo@tue.nl

[⋆⋆] Heriot-Watt University, dept. of Computing and Electrical Eng., Edinburgh, Scotland, email: fairouz@cee.hw.ac.uk

[⋆⋆⋆] email: twan.laan@wxs.nl

[†] Same address as Bloo, email: r.p.nederpelt@tue.nl

Similar but more recent experience with the Coq proof system [10] suggests the same necessity of parameterized definitions, and indeed, state of the art theorem provers have parameter mechanisms as well.

There is another advantage to the use of parameters. Allowing only parameters of a certain type and not the corresponding abstractions may yield a weaker type system. This can have advantages such as a first-order system instead of a higher-order one, or a simpler typecheck algorithm as has been observed for the type system $\lambda P-$ in [14].

Parameters, constants and definitions in programming languages Most non-assembly level programming languages have parameterized definitions as part of the syntax. Consider the Pascal definition of a function double:

```
function  double(z : Integer) : Integer;
begin
   double := z + z
end;
```

The argument $(\texttt{z} : \texttt{Integer})$ is a parameter in our sense: the function `double` can only be used when given an argument, ergo `double` is a non-abstracted function. In ordinary λ-calculus this function `double` can only be represented by the λ-abstraction $(\lambda x : \texttt{Int}.(x + x))$. This representation is unfaithful since this way `double` is a term on its own, of 'higher-order character' (it can be used without a parameter).

For an example of the use of constants, we consider the programming language ML, which has the basic types `int` and `list`. However, `list` is not just an ordinary constant. It can only be used when given an argument (which is written prefix in ML) as in `int list`. We see that `list` is in fact a parameterized constant.

1.1 Extending pure type systems with parameters, constants and definitions

There are many other examples of the frequent use of parameters in mathematics and computer science, occurring in combination with both definitions and constants. The general framework used to describe type systems, Pure Type Systems (PTSs, [2]), does not possess constants[1] or parameters nor does it have syntax for definitions. Therefore we set out to extend pure type systems with parameters, constants and definitions in order to better be able to describe type systems used in practice. This work is based on the parameterized type systems of Laan in [12], although there are several subtle differences[2] in the precise definition of the system. We first discuss work that has already been done in this direction.

Two approaches are known for extending type theory with definitions. The first, by Severi and Poll, extends the syntax of λ-terms to include definitions [17]. The second only extends PTSs with global definitions (i.e., definitions in the context of derivations), and treats local definitions as ordinary β-redexes [5].

[1] The role of unparameterized constants is usually imitated by variables, by agreeing not to make any abstraction over such variables. This is done in ordinary PTSs for the sorts $*$ and $\Box$. Another extension of PTSs in which constants play an essential role are Modal PTSs, cf. [6].

[2] Definition 1 is slightly different, the rules (C^p-weak) and (D^p-weak) are now correct and we have a different treatment of topsorts in the rules for definitions.

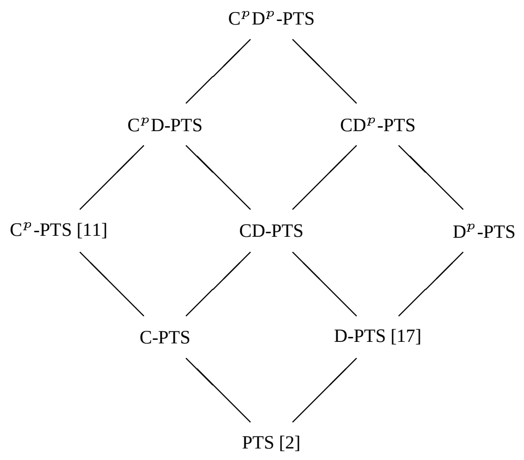

Fig. 1. The hierarchy of parameters, constants and definitions

Both these approaches fail to model the definition of the `double` function from section 1 above, since they don't have parameterized definitions. The best they can do is adding $\texttt{double} = (\lambda x{:}\texttt{Int}.x + x) : \mathit{\Pi} x{:}\texttt{Int}.\texttt{Int}$ to the context, which clearly isn't in the spirit of Pascal where `double` on its own is not a valid expression.

We shall call the extension of PTSs with unparameterized definitions D-PTSs. In this paper, we go one step further and introduce D^p-PTSs, *pure type systems extended with parameterized definitions*, so that we can imitate `double` by adding $\texttt{double}(z : \texttt{Int}) = z + z : \texttt{Int}$ to the context. This will *not* allow the use of `double` unless it is provided an argument for its parameter z. This is an extension of the work of [17] on unparameterized definitions.

Orthogonally, one can extend PTSs with parameterized constants as has been studied in [11]. We shall call these systems C^p-PTSs. Similar to the extension of PTSs with unparameterized definitions, one might consider PTSs extended with unparameterized constants only (C-PTSs). Although C-PTSs are not very interesting on their own, we include them here for symmetry reasons.

Combining the various extensions, we obtain a hierarchy that can be depicted as in Figure 1.

In this paper we study the top system in Figure 1, that is, PTSs extended with parameterized constants as well as parameterized definitions.

Similar to the restrictions on the formation of abstractions in ordinary PTSs, it is natural to put restrictions on the formation of parameters as well. Although an unrestricted use of parameters may seem elegant from a theoretical point of view, this is not custom in programming languages. For instance, in many Pascal versions, parametric *terms* can only have parameters at *term level*. Therefore, in the C^pD^p-PTSs we study in this paper, in a parameterized term $t(p_1, \ldots, p_n)$ we might want to restrict its formation according to the type of $t(p_1,\ldots,p_n)$ as well as according to the types of the parameters $p_1,\ldots,p_n$.

This paper is organized as follows:

In Section 2, we define C^{p}D^p-PTSs, PTSs extended with parametric constants and definitions. This section includes an extension of the δ-reduction of [17] to parametric definitions.

In Section 3 we extend the theory in [17] to C^{p}D^p-PTSs. We show that our extended δ-reduction (needed for unfolding definitions) and $\beta\delta$-reductions are also confluent, and that the extended δ-reduction (under reasonable conditions) is strongly normalizing. Then we show some elementary properties like the Generation Lemma, and the Subject Reduction property for $\beta\delta$-reduction. Finally we prove that $\beta\delta$-reduction in a C^{p}D^p-PTS is strongly normalizing if a slightly stronger PTS is β-strongly normalizing.

Section 4 is devoted to comparing C^{p}D^p-PTSs to ordinary PTSs. We show that for a large class of C^{p}D^p-PTSs there is a natural projection into an ordinary, but stronger, PTS.

We conclude in Section 5.

2 Extending PTSs with parametric constants and definitions

In this section, we extend Pure Type Systems (PTSs) (cf. [2]) with parameterized constants and parameterized definitions.

Pure Type Systems (PTSs) were introduced by Berardi [4] and Terlouw [18] as a general framework in which many current type systems can be described. Though PTSs were not introduced before 1988, many rules in PTSs are highly influenced by rules of known type systems like Church's Simple Theory of Types [8] and AUTOMATH (see 5.5.4. of [9]). The description of our extension of PTSs with parametric constants and definitions is based on the description of PTSs in [2].

Definition 1 Let $\mathcal{V}$, $\mathcal{C}$ and $\boldsymbol{S}$ be disjoint sets of respectively variables, constants and sorts.[3] The set $\mathcal{T}_P$ of *parametric terms* is defined together with the set $\mathcal{L}_V$ of *lists of typed variables* and the set $\mathcal{L}_T$ of *lists of terms* by:

$$\begin{aligned}\mathcal{T}_P &::= \mathcal{V} \mid \boldsymbol{S} \mid \mathcal{C}(\mathcal{L}_T) \mid (\mathcal{T}_P\mathcal{T}_P) \mid (\lambda\mathcal{V}{:}\mathcal{T}_P.\mathcal{T}_P) \\ &\quad\mid (\Pi\mathcal{V}{:}\mathcal{T}_P.\mathcal{T}_P) \mid (\mathcal{C}(\mathcal{L}_V){=}\mathcal{T}_P{:}\mathcal{T}_P \;\texttt{in}\; \mathcal{T}_P);\\ \mathcal{L}_V &::= \varnothing \mid \langle\mathcal{L}_V,\mathcal{V}{:}\mathcal{T}_P\rangle; \qquad \mathcal{L}_T ::= \varnothing \mid \langle\mathcal{L}_T,\mathcal{T}_P\rangle.\end{aligned}$$

Instead of $\langle\cdots\langle\langle\varnothing,x_1{:}A_1\rangle,x_2{:}A_2\rangle\cdots x_n{:}A_n\rangle$, we write $\langle x_1{:}A_1,\ldots,x_n{:}A_n\rangle$ or $x_1{:}A_1,\ldots,x_n{:}A_n$. A similar convention is adopted for lists of terms. In a parametric term of the form $c(b_1,\ldots,b_n)$, the subterms $b_1,\ldots,b_n$ are called the *parameters* of the term.

Terms of the form $\mathcal{C}(\mathcal{L}_V){=}\mathcal{T}_P{:}\mathcal{T}_P$ in $\mathcal{T}_P$ represent parametric local definitions. An example of such a term is $\texttt{double}(\texttt{x}{:}\texttt{Int}){=}(\texttt{x}{+}\texttt{x}){:}\texttt{Int}$ in A which indicates that a subterm of A of the form $\texttt{double}(P)$ is to be interpreted as $P+P$, and has type $\texttt{Int}$. The definition is local, that is: the scope of the definition is the term A. Local definitions contrast with global definitions which are given in a context Γ, and refer to any term that is considered within Γ (see Definition 5). The definition system in AUTOMATH is similar to the system of global definitions in this paper. However, there are no local definitions in AUTOMATH.

[3] Note that, in contrast to PTSs, we require the set of sorts to be disjoint from the set of (parametric) constants.

Definition 2 Let $\vec{x}{:}\vec{A}$ denote the list $x_1{:}A_1, \ldots, x_n{:}A_n$. $FV(A)$, the set of *free variables* of a parametric term A is defined as usual with the extra cases for constants and definitions:
$FV(c(a_1, \ldots, a_n)) = \bigcup_{i=1}^{n} FV(a_i)$; and
$FV(c(\vec{x}{:}\vec{A}){=}A{:}B \texttt{ in } C) =$
$\quad\bigcup_{i=1}^{n}(FV(A_i) \setminus \{x_1, \ldots, x_{i-1}\}) \cup \big((FV(A) \cup FV(B)) \setminus \{x_1, \ldots, x_n\}\big) \cup FV(C)$.
We similarly define $Cons(A)$, the set of *constants and global definitions* of A as follows:
$Cons(s) = Cons(x) = \emptyset$;
$Cons(c(a_1, \ldots, a_n)) = \{c\} \cup \bigcup_{i=1}^{n} Cons(a_i)$;
$Cons(AB) = Cons(\lambda x{:}A.B) = Cons(\Pi x{:}A.B) = Cons(A) \cup Cons(B)$;
$Cons(c(\vec{x}{:}\vec{A}){=}A{:}B \texttt{ in } C) =$
$\quad\bigcup_{i=1}^{n} Cons(A_i) \cup Cons(A) \cup Cons(B) \cup (Cons(C) \setminus \{c\})$.

$FV(A) \cup Cons(A)$ forms the *domain* $Dom(A)$ of A.

We omit parentheses in parametric terms when possible. As usual in PTSs (cf. [2]), we do not distinguish terms that are equal up to renaming of bound variables. Moreover, we assume the Barendregt variable convention so that names of bound variables and constants will always be chosen such that they differ from the free ones in a term.

Definition 3 We extend the usual definition of substitution of a term a for a variable x in a term b, $b[x{:=}a]$, to parametric terms, assuming that x is not a bound variable of either b or a:
$\quad c(b_1, \ldots, b_n)[x{:=}a] \equiv c(b_1[x{:=}a], \ldots, b_n[x{:=}a])$;
$\quad (c(\vec{x}{:}\vec{A}) = A{:}B \texttt{ in } C)[x{:=}a] \equiv$
$\qquad c(x_1{:}A_1[x{:=}a], \ldots, x_n{:}A_n[x{:=}a]) = A[x{:=}a]{:}B[x{:=}a] \texttt{ in } C[x{:=}a]$.

Definition 4 The set $\mathcal{C}_P$ of *contexts*, which we denote by $\Gamma, \Gamma', \ldots$, is given by:
$\quad \mathcal{C}_P ::= \varnothing \mid \langle \mathcal{C}_P, \mathcal{V}{:}\mathcal{T}_P \rangle \mid \langle \mathcal{C}_P, \mathcal{C}(\mathcal{L}_V){=}\mathcal{T}_P{:}\mathcal{T}_P \rangle \mid \langle \mathcal{C}_P, \mathcal{C}(\mathcal{L}_V){:}\mathcal{T}_P \rangle$.

Notice that $\mathcal{L}_V \subseteq \mathcal{C}_P$: all lists of variable declarations are contexts as well.

Definition 5 Let Γ be a context. *Declarations* are elements of Γ as follows:

- $x{:}A$ is a *variable declaration* with *subject* x and *type* A;
- $c(x_1{:}B_1, \ldots, x_n{:}B_n){:}A$ is a *constant declaration* with *subject* c (also called *primitive constant*), *parameters* $x_1, \ldots, x_n$ and *type* A;
- $c(x_1{:}B_1, \ldots, x_n{:}B_n){=}a{:}A$ is a *global definition (declaration)* with *subject* c (also called *globally defined constant*), *parameters* $x_1, \ldots, x_n$, *definiens* a and *type* A.

Notation In the rest of this paper, Δ denotes a context $x_1{:}B_1, \ldots, x_n{:}B_n$ consisting of variable declarations only. Such a context is typically used as a list of parameters in a definition $c(\Delta){=}a{:}A$. We write $\Delta_i \equiv x_1{:}B_1, \ldots, x_{i-1}{:}B_{i-1}$ for $i \leq n$. We extend the definition of substitution to contexts in the usual way.

Definition 6 For a context Γ we define $FV(\Gamma)$ to be the set of subjects of variable declarations in Γ and $Cons(\Gamma)$ the set of subjects of constant declarations and global definitions in Γ. The *domain* of Γ, $Dom(\Gamma)$, is defined as $FV(\Gamma) \cup Cons(\Gamma)$.

In ordinary PTSs we have that, for a legal term A in a legal context Γ, $FV(A) \subseteq FV(\Gamma)$. In our C^{p}D^p-PTSs we will have: $FV(A) \subseteq FV(\Gamma)$ and $Cons(A) \subseteq Cons(\Gamma)$.

A natural condition on contexts is that all variables, primitive constants and defined constants are declared only once. Furthermore it is also natural to require that variables and constants are declared *before* they are being used. For this we introduce the notion of *sound* context:

Definition 7 $\Gamma \in \mathcal{C}_P$ is *sound* if variables, primitive constants and defined constants are declared only once and if $\Gamma \equiv \Gamma_1, c(\Delta){=}a{:}A, \Gamma_2$ then $Dom(a) \cup Dom(A) \subseteq Dom(\Gamma_1) \cup Dom(\Delta)$ and for $i = 1, \ldots, n$: $Dom(B_i) \subseteq Dom(\Gamma_1, \Delta_i)$.

The contexts occurring in the type systems proposed in this paper are all sound (see Lemma 15). This fact will be useful when proving properties of these systems.

We now start a more detailed description of the top system in Figure 1, the system with both parameterized defined constants and parameterized primitive constants. We define two reduction relations, namely the δ- and β-reduction. β-reduction is defined as usual, and we use $\to_\beta$, $\twoheadrightarrow_\beta$, $\to_\beta^+$, and $=_\beta$ as usual. As far as global definitions are concerned, δ-reduction is comparable to δ-reduction in AUTOMATH. This is reflected in rule $(\delta 1)$ in Figure 2 and Definition 8 below. But now, a δ-reduction step can also unfold *local* definitions. Therefore, two new reduction steps are introduced. Rule $(\delta 2)$ removes the declaration of a local definition if there is no position within its scope where it can be unfolded ('removal of void local definitions'). Rule $(\delta 3)$ shows how one can treat a local definition as a global definition, and thus how the problem of unfolding local definitions can be reduced to unfolding global definitions ('localization of global definitions'). Remember that $\Delta \equiv x_1{:}B_1, \ldots, x_n{:}B_n$.

Definition 8 (δ-reduction) δ-reduction is defined as the smallest relation $\to_\delta$ on $\mathcal{C}_P \times \mathcal{T}_P \times \mathcal{T}_P$ closed under the rules $(\delta 1)$, $(\delta 2)$, $(\delta 3)$ and the compatibility rules of Figure 2.

$\Gamma \vdash \cdot =_\delta \cdot$ denotes the reflexive, symmetric and transitive closure of $\Gamma \vdash \cdot \to_\delta \cdot$. When Γ is the empty context, we write $a \to_\delta a'$ instead of $\Gamma \vdash a \to_\delta a'$.

Furthermore δ-reduction between contexts is the smallest relation $\to_\delta$ on $\mathcal{C}_P \times \mathcal{C}_P$ closed under the rules in Figure 3.

Before describing the typing rules for C^{p}D^p-PTSs, we introduce the concepts of specification (taken from [2]) and parametric specification.

Definition 9 (Specification) A *specification* is a triple $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$, such that $\boldsymbol{S} \subseteq \mathcal{C}$, $\boldsymbol{A} \subseteq \boldsymbol{S} \times \boldsymbol{S}$ and $\boldsymbol{R} \subseteq \boldsymbol{S} \times \boldsymbol{S} \times \boldsymbol{S}$. The specification is called *singly sorted* if $\boldsymbol{A}$ is a (partial) function $\boldsymbol{S} \to \boldsymbol{S}$, and $\boldsymbol{R}$ is a (partial) function $\boldsymbol{S} \times \boldsymbol{S} \to \boldsymbol{S}$. $\boldsymbol{S}$ is called the set of *sorts*, $\boldsymbol{A}$ is the set of *axioms*, and $\boldsymbol{R}$ is the set of (Π-formation) *rules* of the specification.

A *parametric specification* is a quadruple $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P})$ such that $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ is a specification, and the set of *parametric rules* $\boldsymbol{P} \subseteq \boldsymbol{S} \times \boldsymbol{S}$. The parametric specification is called *singly sorted* if the specification $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ is singly sorted.

We first give the typing rules for ordinary terms of PTSs. These can also be found in [2].

Definition 10 (Typing rules for ordinary terms) Let $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ be a specification. The Pure Type System $\lambda\mathcal{S}$ describes in which ways judgements $\Gamma \vdash_{\mathcal{S}} A : B$ (or

$$(\delta 1):\ \ \Gamma_1, c(\Delta){=}a{:}A, \Gamma_2 \vdash c(b_1,\ldots,b_n) \to_\delta a[x_i{:=}b_i]_{i=1}^n$$

$$(\delta 2):\ \frac{c \notin \mathit{Cons}(b)}{\Gamma \vdash c(\Delta){=}a{:}A \ \mathtt{in}\ b \to_\delta b} \qquad (\delta 3):\ \frac{\Gamma, c(\Delta){=}a{:}A \vdash b \to_\delta b'}{\Gamma \vdash c(\Delta){=}a{:}A\ \mathtt{in}\ b \to_\delta c(\Delta){=}a{:}A\ \mathtt{in}\ b'}$$

$$\frac{\Gamma, \Delta \vdash a \to_\delta a'}{\Gamma \vdash c(\Delta){=}a{:}A\ \mathtt{in}\ b \to_\delta c(\Delta){=}a'{:}A\ \mathtt{in}\ b} \qquad \frac{\Gamma, \Delta \vdash A \to_\delta A'}{\Gamma \vdash c(\Delta){=}a{:}A\ \mathtt{in}\ b \to_\delta c(\Delta){=}a{:}A'\ \mathtt{in}\ b}$$

$$\frac{\Gamma, \Delta_i \vdash B_i \to_\delta B_i'}{\Gamma \vdash c(\Delta){=}a{:}A\ \mathtt{in}\ b \to_\delta c(x_1{:}B_1,\ldots,x_i{:}B_i',\ldots,x_n{:}B_n){=}a{:}A\ \mathtt{in}\ b}$$

$$\frac{\Gamma \vdash a \to_\delta a'}{\Gamma \vdash ab \to_\delta a'b} \qquad \frac{\Gamma \vdash b \to_\delta b'}{\Gamma \vdash ab \to_\delta ab'}$$

$$\frac{\Gamma, x{:}A \vdash a \to_\delta a'}{\Gamma \vdash \lambda x{:}A.a \to_\delta \lambda x{:}A.a'} \qquad \frac{\Gamma \vdash A \to_\delta A'}{\Gamma \vdash \lambda x{:}A.a \to_\delta \lambda x{:}A'.a}$$

$$\frac{\Gamma, x{:}A \vdash a \to_\delta a'}{\Gamma \vdash \Pi x{:}A.a \to_\delta \Pi x{:}A.a'} \qquad \frac{\Gamma \vdash A \to_\delta A'}{\Gamma \vdash \Pi x{:}A.a \to_\delta \Pi x{:}A'.a}$$

$$\frac{\Gamma \vdash a_j \to_\delta a_j'}{\Gamma \vdash c(a_1,\ldots,a_n) \to_\delta c(a_1,\ldots,a_j',\ldots,a_n)}$$

Fig. 2. Reduction rules and compatibility rules for $\to_\delta$

$$\frac{\Gamma_1 \vdash A \to_\delta A'}{\Gamma_1, x{:}A, \Gamma_2 \to_\delta \Gamma_1, x{:}A', \Gamma_2} \qquad \frac{\Gamma_1, \Delta \to_\delta \Gamma_1, \Delta'}{\Gamma_1, c(\Delta){:}A, \Gamma_2 \to_\delta \Gamma_1, c(\Delta'){:}A, \Gamma_2}$$

$$\frac{\Gamma_1, \Delta \vdash A \to_\delta A'}{\Gamma_1, c(\Delta){:}A, \Gamma_2 \to_\delta \Gamma_1, c(\Delta){:}A', \Gamma_2} \qquad \frac{\Gamma_1, \Delta \vdash a \to_\delta a'}{\Gamma_1, c(\Delta){=}a{:}A, \Gamma_2 \to_\delta \Gamma_1, c(\Delta){=}a'{:}A, \Gamma_2}$$

$$\frac{\Gamma_1, \Delta \to_\delta \Gamma_1, \Delta'}{\Gamma_1, c(\Delta){=}a{:}A, \Gamma_2 \to_\delta \Gamma_1, c(\Delta'){=}a{:}A, \Gamma_2} \qquad \frac{\Gamma_1, \Delta \vdash A \to_\delta A'}{\Gamma_1, c(\Delta){=}a{:}A, \Gamma_2 \to_\delta \Gamma_1, c(\Delta){=}a{:}A', \Gamma_2}$$

Fig. 3. Rules for δ-reduction on contexts

$\Gamma \vdash A : B$, if it is clear which $\mathcal{S}$ is used) can be derived. $\Gamma \vdash A : B$ states that A has type B in context Γ. $\lambda\mathcal{S}$ consists of the derivation rules given in Figure 4.

A term a is *legal* (with respect to a certain type system) if there are Γ, b such that either $\Gamma \vdash a : b$ or $\Gamma \vdash b : a$ is derivable (in that type system). Similarly, a context Γ is *legal* if there are a, b such that $\Gamma \vdash a : b$. A sort $s \in \boldsymbol{S}$ is called *topsort* if there is no context Γ and $s' \in \boldsymbol{S}$ such that $\Gamma \vdash s : s'$.

An important class of examples of PTSs is formed by the eight PTSs of the so-called Barendregt Cube. The Barendregt Cube is a three-dimensional presentation of eight well-known PTSs. All systems have sorts $\boldsymbol{S} = \{*, \Box\}$, and axioms $\boldsymbol{A} = \{(*, \Box)\}$. Moreover, all the systems have rule $(*, *, *)$. System $\lambda{\to}$ has no extra rules, but the other seven systems all have one or more of the rules $(*, \Box, \Box)$, $(\Box, *, *)$ and $(\Box, \Box, \Box)$.

(axiom)	$\langle\rangle \vdash s_1 : s_2$	$(s_1, s_2) \in \boldsymbol{A}$
(start)	$\dfrac{\mathit{\Gamma} \vdash A : s}{\mathit{\Gamma}, x{:}A \vdash x : A}$	$x \notin \mathit{Dom}(\mathit{\Gamma})$
(weak)	$\dfrac{\mathit{\Gamma} \vdash A : B \qquad \mathit{\Gamma} \vdash C : s}{\mathit{\Gamma}, x{:}C \vdash A : B}$	$x \notin \mathit{Dom}(\mathit{\Gamma})$
($\mathit{\Pi}$)	$\dfrac{\mathit{\Gamma} \vdash A : s_1 \qquad \mathit{\Gamma}, x{:}A \vdash B : s_2}{\mathit{\Gamma} \vdash (\mathit{\Pi} x{:}A.B) : s_3}$	$(s_1, s_2, s_3) \in \boldsymbol{R}$
(λ)	$\dfrac{\mathit{\Gamma}, x{:}A \vdash b : B \qquad \mathit{\Gamma} \vdash (\mathit{\Pi} x{:}A.B) : s}{\mathit{\Gamma} \vdash (\lambda x{:}A.b) : (\mathit{\Pi} x{:}A.B)}$	
(appl)	$\dfrac{\mathit{\Gamma} \vdash F : (\mathit{\Pi} x{:}A.B) \qquad \mathit{\Gamma} \vdash a : A}{\mathit{\Gamma} \vdash Fa : B[x{:=}a]}$	
(conv)	$\dfrac{\mathit{\Gamma} \vdash A : B \qquad \mathit{\Gamma} \vdash B' : s \qquad B =_\beta B'}{\mathit{\Gamma} \vdash A : B'}$	

Fig. 4. Typing rules for PTSs

(C^p-weak)	$\dfrac{\mathit{\Gamma} \vdash^{C^p} b{:}B \quad \mathit{\Gamma}, \mathit{\Delta} \vdash^{C^p} A{:}s \quad \mathit{\Gamma}, \mathit{\Delta}_i \vdash^{C^p} B_i{:}s_i \quad (s_i, s) \in \boldsymbol{P} \quad (i = 1, \ldots, n)}{\mathit{\Gamma}, c(\mathit{\Delta}) : A \vdash^{C^p} b : B}$
(C^p-app)	$\dfrac{\begin{array}{ll} \mathit{\Gamma}_1, c(\mathit{\Delta}){:}A, \mathit{\Gamma}_2 \vdash^{C^p} b_i{:}B_i[x_j{:=}b_j]_{j=1}^{i-1} & (i = 1, \ldots, n) \\ \mathit{\Gamma}_1, c(\mathit{\Delta}){:}A, \mathit{\Gamma}_2 \vdash^{C^p} A : s & (\text{if } n = 0) \end{array}}{\mathit{\Gamma}_1, c(\mathit{\Delta}){:}A, \mathit{\Gamma}_2 \vdash^{C^p} c(b_1, \ldots, b_n) : A[x_j{:=}b_j]_{j=1}^{n}}$

Fig. 5. Typing rules for parametric constants

Now we add rules for typing parametric constants. In these rules we use the set of parametric rules $\boldsymbol{P}$ to govern which types parameters can have.

Definition 11 (Typing rules for parametric constants) Let $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P})$ be a parametric specification. The *typing relation* $\vdash^{C^p}$ is the smallest relation on $\mathcal{C}_P \times \mathcal{T}_P \times \mathcal{T}_P$ closed under the rules in Definition 10 and the rules (C^p-weak) and (C^p-app) in Figure 5. Recall that $\mathit{\Delta} \equiv x_1{:}B_1, \ldots, x_n{:}B_n$ and $s \in \boldsymbol{S}$. The parameterized constant c in the C^p-weakening rule is, due to the Barendregt variable convention, assumed to be $\mathit{\Gamma}$-fresh, that is, $c \notin \mathit{Cons}(\mathit{\Gamma})$.

At first sight one might miss a C^p-introduction rule. Such a rule, however, is not necessary, as c (on its own) is not a term. A parameterized constant c can only be (part of) a term in the form $c(b_1, \ldots, b_n)$, and such terms can be typed by the C^p-application rule. The extra condition $\mathit{\Gamma}_1, c(\mathit{\Delta}){:}A, \mathit{\Gamma}_2 \vdash^{C^p} A : s$ in the C^p-application rule for $n = 0$ is necessary to prevent an empty list of premises. Such an empty list of premises would make it possible to have almost arbitrary contexts in the conclusion. The extra condition is only needed to assure that the context in the conclusion is a legal context.

(D^p-weak) $$\frac{\Gamma \vdash^{D^p} b{:}B \quad \Gamma, \Delta \vdash^{D^p} a{:}A{:}s \quad \Gamma, \Delta_i \vdash^{D^p} B_i{:}s_i \quad (s_i, s) \in \boldsymbol{P} \quad (i{=}1, \dots, n)}{\Gamma, c(\Delta){=}a{:}A \vdash^{D^p} b : B}$$

(D^p-app) $$\frac{\begin{array}{ll}\Gamma_1, c(\Delta){=}a{:}A, \Gamma_2 \vdash^{D^p} b_i : B_i[x_j{:=}b_j]_{j=1}^{i-1} & (i = 1, \dots, n) \\ \Gamma_1, c(\Delta){=}a{:}A, \Gamma_2 \vdash^{D^p} a : A & (\text{if } n = 0)\end{array}}{\Gamma_1, c(\Delta){=}a{:}A, \Gamma_2 \vdash^{D^p} c(b_1, \dots, b_n) : A[x_j{:=}b_j]_{j=1}^{n}}$$

(D^p-form) $$\frac{\Gamma, c(\Delta){=}a{:}A \vdash^{D^p} B : s}{\Gamma \vdash^{D^p} c(\Delta){=}a{:}A \texttt{ in } B : s}$$

(D^p-intro) $$\frac{\Gamma, c(\Delta){=}a{:}A \vdash^{D^p} b : B \qquad \Gamma \vdash^{D^p} c(\Delta){=}a{:}A \texttt{ in } B : s}{\Gamma \vdash^{D^p} c(\Delta){=}a{:}A \texttt{ in } b : c(\Delta){=}a{:}A \texttt{ in } B}$$

(D^p-conv) $$\frac{\Gamma \vdash^{D^p} b : B \qquad \Gamma \vdash^{D^p} B' : s \qquad \Gamma \vdash B =_\delta B'}{\Gamma \vdash^{D^p} b : B'}$$

Fig. 6. Typing rules for parametric definitions

Note that in the (C^p-weak) rule it is not necessary that all the s_i are equal: in one application of rule (C^p-weak) it is possible to rely on more than one element of $\boldsymbol{P}$.

Remark 12 If we have a parametric constant *plus*$(x{:}\texttt{Int}, y{:}\texttt{Int}){:}\texttt{Int}$ in the context, then it is tempting to think of *plus* as a parametric function. Note however that in PTS-terms it is not a function anymore since the only way to obtain a legal term with it is in its parameterized form *plus*(x, y) which has type $\texttt{Int}$; *plus*$(x{:}\texttt{Int}, y{:}\texttt{Int})$ itself is not a legal term. In order to talk about properties of *plus* 'as a function' we are forced to consider $\lambda x{:}\texttt{Int}.\lambda y{:}\texttt{Int}.$*plus*$(x, y)$.

Adapting the rules from Definition 11 and the rules for definitions of [17] results in rules for *parametric definitions* (by $\Gamma \vdash a : A : s$ we mean $\Gamma \vdash a : A$ and $\Gamma \vdash A : s$):

Definition 13 (Typing rules for parametric definitions) The *typing relation* $\vdash^{D^p}$ is the smallest relation on $\mathcal{C}_P \times \mathcal{T}_P \times \mathcal{T}_P$ closed under the rules in Definition 10 and those in Figure 6, where $s \in \boldsymbol{S}$, and the parameterized definition c that is introduced in the D^p-weakening rule is assumed to be Γ-fresh. Again it is not necessary that all the s_i in the **(D^p-weak)** rule are equal.

Definition 14 (C^{p}D^p-PTSs) Let $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P})$ be a parametric specification. Then $\vdash^{C^pD^p}$ is the smallest relation on $\mathcal{C}_P \times \mathcal{T}_P \times \mathcal{T}_P$ that is closed under the rules of Definitions 10, 11 and 13. The C^{p}D^p-PTS $\lambda^{C^pD^p}\mathcal{S}$ is the system with typing relation $\vdash^{C^pD^p}$.

All contexts occurring in C^{p}D^p-PTSs are sound (see Definition 7). As C^{p}D^p-PTSs are clearly extensions of PTSs, C^p-PTSs and D^p-PTSs, then all contexts occurring in PTSs, C^p-PTSs and D^p-PTSs are sound.

Lemma 15 *Assume* $\Gamma \vdash^{C^pD^p} b : B$. *Then*
1. Dom(b), *Dom*$(B) \subseteq$ *Dom*(Γ); *2.* Γ *is sound.*

PROOF: We prove 1 and 2 simultaneously by induction on the derivation of $\Gamma \vdash^{C^p D^p} b : B$. ⊠

In the specific case of the Barendregt Cube, the combination of $\boldsymbol{R}$ and $\boldsymbol{P}$ leads to a refinement of the Cube, thus making it possible to classify more type systems within one and the same framework. This is studied in detail for PTSs extended with parameterized constants (without definitions) in [11]; it is shown that the type systems of AUTOMATH, LF and ML can be described more naturally and accurately than in ordinary PTSs.

Remark 16 Let $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ be a specification, and observe the parametric specification $\mathcal{S}' = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \varnothing)$. The fact that the set of parametric rules is empty does not exclude the existence of definitions: it is still possible to apply the rules (D^p-weak) and (D^p-app) for $n = 0$. In that case, we obtain only definitions without parameters, and the rules of the parametric system reduce to the rules of a D-PTS with specification $\mathcal{S}$ as introduced by [17].[4] There is however one case of the rules in [17] that a $\mathrm{C}^p\mathrm{D}^p$-PTS cannot simulate, since in rule (D^p-weak) we require $\Gamma, \Delta \vdash^{D^p} A : s$ which [17] does not. Therefore, the system of [17] can abbreviate inhabitants of topsorts which is impossible in the $\mathrm{C}^p\mathrm{D}^p$-PTSs of Definition 14. We feel that abbreviating inhabitants of topsorts is not very useful. It is however routine to check that adding an extra rule

$$(\mathrm{D}^p\text{-weak-top}) \quad \frac{\Gamma \vdash^{D^p} b : B \quad \Gamma, \Delta \vdash^{D^p} a : A}{\Gamma, c() = a : A \vdash^{D^p} b : B}$$

does not change the theory of $\mathrm{C}^p\mathrm{D}^p$-PTSs developed in sections 3 and 4, and yields exactly the same power for unparameterized definitions as the D-PTSs of [17].

3 Meta-Properties

We first list the properties of terms which are not dependent of their being legal. However, we often demand that the free variables and constants of a term are contained in the domain of a sound context.

First we show that δ-reduction is invariant under enlarging of the context ($\rightarrow_\delta$-weakening), then we establish a relation between substitution and $\rightarrow_{\beta\delta}$ (substitutivity), then we can establish confluence for $\rightarrow_{\beta\delta}$ and termination of $\rightarrow_\delta$. Most proofs are similar to those of [19, 17]; more details can be found in the technical report [13].

Lemma 17 ($\rightarrow_\delta$-weakening and Substitutivity)

1. *If $\langle \Gamma_1, \Gamma_2, \Gamma_3 \rangle \in \mathcal{C}_P$ is such that $\Gamma_1, \Gamma_3 \vdash b \rightarrow_\delta b'$, then $\Gamma_1, \Gamma_2, \Gamma_3 \vdash b \rightarrow_\delta b'$;*
2. *If $a \rightarrow_\beta a'$ then $a[x{:=}b] \rightarrow_\beta a'[x{:=}b]$;*
3. *If $\Gamma \vdash a \rightarrow_\delta a'$ then $\Gamma[x{:=}b] \vdash a[x{:=}b] \rightarrow_\delta a'[x{:=}b]$;*

[4] The parametric system with specification $\mathcal{S}'$ has a C^p-weakening rule while the system of [17] does not. But the C^p-weakening rule can only be used for $n = 0$, and in that case C^p-weakening can be imitated by the normal weakening rule of PTSs: a parametric constant with zero parameters is in fact a parameter-free constant, and for such a constant one can use a variable as well.

4. *If $\Gamma \vdash b \rightarrow_{\beta\delta} b'$ then $\Gamma \vdash a[x{:=}b] \twoheadrightarrow_{\beta\delta} a[x{:=}b']$.*

Theorem 18 (Confluence for $\rightarrow_\beta$, $\rightarrow_\delta$ and Strong Normalization for $\rightarrow_\delta$)

1. *$\rightarrow_\beta$ is confluent.*
2. *$\rightarrow_{\beta\delta}$ is confluent when taking contexts into account: if Γ is sound, $\Gamma \vdash a \twoheadrightarrow_{\beta\delta} b_1$ and $\Gamma \vdash a \twoheadrightarrow_{\beta\delta} b_2$ then there exists a term d such that $\Gamma \vdash b_1 \twoheadrightarrow_{\beta\delta} d$ and $\Gamma \vdash b_2 \twoheadrightarrow_{\beta\delta} d$.*
3. *$\rightarrow_\delta$, when restricted to sound contexts Γ and terms a with Dom$(a) \subseteq$ Dom(Γ), is strongly normalizing, i.e. there are no infinite δ-reduction paths.*

Without the restriction to sound contexts Γ and terms a with *Dom*$(a) \subseteq$ *Dom*(Γ), we do not even have weak normalization: consider $\Gamma \equiv \langle \mathtt{c}()=\mathtt{d}():\mathtt{A}, \mathtt{d}()=\mathtt{c}():\mathtt{A}\rangle$. The term $\mathtt{c}()$ does not have a Γ-normal form.

3.1 Properties of Legal terms

The properties in this section are proved for all *legal* terms, i.e. for terms a for which there are A, Γ such that $\Gamma \vdash^{C^pD^p} a : A$ or $\Gamma \vdash^{C^pD^p} A : a$. The main property we prove is that strong normalization of a PTS is preserved by certain extensions.

Many of the standard properties of PTSs in [2] hold for C^{p}D^p-PTSs as well. In the same way as in [2], we can prove the following theorem:

Theorem 19 *Let $\mathcal{S}$ be a parametric specification. The type system $\lambda^{C^pD^p}\mathcal{S}$ has the following properties:*[5]

1) Substitution Lemma; 2) Correctness of Types; 3) Subject Reduction (for $\rightarrow_{\beta\delta}$).
4) If $\mathcal{S}$ is singly sorted then $\lambda^{C^pD^p}\mathcal{S}$ has Uniqueness of Types.

The Generation Lemma is extended with two extra cases:

Lemma 20 (Generation Lemma, extension)

1. *If $\Gamma \vdash^{C^pD^p} c(b_1,\ldots,b_n) : D$ then there exist sort s, $\Delta \equiv x_1{:}B_1,\ldots,x_n{:}B_n$ and term A such that $\Gamma \vdash D =_{\beta\delta} A[x_i{:=}b_i]_{i=1}^n$, and $\Gamma \vdash^{C^pD^p} b_i : B_i[x_j{:=}b_j]_{j=1}^{i-1}$. Besides we have one of these two possibilities: (a) $\Gamma = \langle \Gamma_1, c(\Delta){:}A, \Gamma_2\rangle$ and $\Gamma_1, \Delta \vdash^{C^pD^p} A : s$; or (b) $\Gamma = \langle \Gamma_1, c(\Delta){=}a{:}A, \Gamma_2\rangle$ and $\Gamma_1, \Delta \vdash^{C^pD^p} a : A : s$ for some sort s;*
2. *If $\Gamma \vdash^{C^pD^p} c(\Delta){=}a{:}A\ \mathtt{in}\ b : D$ then either we have (a) or (b) below:*
   (a) $\Gamma, c(\Delta){=}a{:}A \vdash^{C^pD^p} b : B$, $\Gamma \vdash^{C^pD^p} (c(\Delta){=}a{:}A\ \mathtt{in}\ B) : s$ and $\Gamma \vdash D =_{\beta\delta} c(\Delta){=}a{:}A\ \mathtt{in}\ B$;
   (b) $\Gamma, c(\Delta){=}a{:}A \vdash^{C^pD^p} b : s$ and $\Gamma \vdash D =_{\beta\delta} s$.

Also Correctness of Contexts has some extra cases compared to usual PTSs. Recall that Γ is *legal* if there are b, B such that $\Gamma \vdash^{C^pD^p} b : B$.

[5] Substitution Lemma: if $\Gamma, x : A, \Delta \vdash b : B$ and $\Gamma \vdash a : A$ then also $\Gamma, \Delta[x := a] \vdash b[x := a] : B[x := a]$. Correctness of Types: if $\Gamma \vdash a : A$ then $\Gamma \vdash A : s$ for some sort s or A is a topsort. Subject Reduction: if $\Gamma \vdash a : A$ and $a \rightarrow_{\beta\delta} a'$ then also $\Gamma \vdash a' : A$. Uniqueness of Types: if $\Gamma \vdash a : A$ and $\Gamma \vdash a : B$ then $\Gamma \vdash A =_{\beta\delta} B$.

Lemma 21 (Correctness of Contexts)

1. *If $\Gamma, x{:}A, \Gamma'$ is legal then there exists a sort s such that $\Gamma \vdash^{C^pD^p} A : s$;*
2. *If $\Gamma, c(\Delta){:}A, \Gamma'$ is legal then $\Gamma, \Delta \vdash^{C^pD^p} A : s$;*
3. *If $\Gamma, c(\Delta){=}a{:}A, \Gamma'$ is legal then $\Gamma, \Delta \vdash^{C^pD^p} a : A : s$.*

Now we prove that $\lambda^{C^pD^p}\mathcal{S}$ is $\beta\delta$-strongly normalizing if a slightly larger PTS $\lambda\mathcal{S}'$ is β-strongly normalizing. We follow the idea of [17] for PTSs extended with only definitions; the extension is tedious but fairly routine.

For legal terms $a \in \mathcal{T}_P$ in a context Γ, we define a lambda term $\|a\|_\Gamma$ without definitions and without parameters. If a is typable in a C^{p}D^p-PTS $\lambda^{C^pD^p}\mathcal{S}$, then $\|a\|_\Gamma$ will be typable in a PTS $\lambda\mathcal{S}'$, where $\mathcal{S}'$ is a so-called *completion* (see Definition 24) of the specification $\mathcal{S}$. Moreover, we take care that if $a \to_\beta a'$, then $\|a\|_\Gamma \to_\beta^+ \|a'\|_\Gamma$ (that is: $\|a\|_\Gamma \twoheadrightarrow_\beta \|a'\|_\Gamma$ and $\|a\|_\Gamma \not\equiv \|a'\|_\Gamma$). Together with strong normalization of δ-reduction (Theorem 18.3), this guarantees that $\lambda^{C^pD^p}\mathcal{S}$ is $\beta\delta$-strongly normalizing whenever $\lambda\mathcal{S}'$ is β-strongly normalizing.

Definition 22 For $a \in \mathcal{T}_P$ and $\Gamma \in \mathcal{C}_P$ we define $\|a\|_\Gamma$ as in [17], but with the extra cases:

$$\|c(b_1,\ldots,b_n)\|_\Gamma \equiv \begin{cases} \|\lambda_{i=1}^n x_i{:}B_i.a\|_{\Gamma_1} \|b_1\|_\Gamma \cdots \|b_n\|_\Gamma \text{ if } \Gamma{=}\langle\Gamma_1, c(\Delta){=}a{:}A, \Gamma_2\rangle; \\ c\,\|b_1\|_\Gamma \cdots \|b_n\|_\Gamma \ \text{ otherwise;} \end{cases}$$

$$\|c(\Delta){=}a{:}A \ \mathtt{in}\ b\|_\Gamma \equiv \left(\lambda c{:}(\|\textstyle\prod_{i=1}^n x_i{:}B_i.A\|_\Gamma).\,\|b\|_{\Gamma,c(\Delta)=a:A}\right) \|\lambda_{i=1}^n x_i{:}B_i.a\|_\Gamma\,.$$

Note: constants in $\mathcal{T}_P$ are translated to similarly named variables in λ-calculus without definitions and parameters.

We now show that $\|\_\|_\_$ translates a δ-reduction into zero or more β-reductions, and that it translates a β-reduction into one or more β-reductions.

Lemma 23 *Let Γ be sound, and assume Dom$(a) \subseteq$ Dom(Γ). The following holds: If $\Gamma \vdash a \to_\delta b$ then $\|a\|_\Gamma \twoheadrightarrow_\beta \|b\|_\Gamma$. Similarly: If $a \to_\beta b$ then $\|a\|_\Gamma \to_\beta^+ \|b\|_\Gamma$.*

Definition 24 The specification $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$ is called *quasi full* if for all $s_1, s_2 \in \boldsymbol{S}$ there exists $s_3 \in \boldsymbol{S}$ such that $(s_1, s_2, s_3) \in \boldsymbol{R}$.
A specification $\mathcal{S}' = (\boldsymbol{S}', \boldsymbol{A}', \boldsymbol{R}')$ is a *completion* of a parametric specification $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P})$ if $\boldsymbol{S} \subseteq \boldsymbol{S}'$, $\boldsymbol{A} \subseteq \boldsymbol{A}'$, and $\boldsymbol{R} \subseteq \boldsymbol{R}'$, $\boldsymbol{S}'$ is quasi full, and $\forall s \in \boldsymbol{S} \exists s' \in \boldsymbol{S}'[(s, s') \in \boldsymbol{A}']$.[6]

Theorem 25 *Let $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P})$ and $\mathcal{S}' = (\boldsymbol{S}', \boldsymbol{A}', \boldsymbol{R}')$ be such that $\mathcal{S}'$ is a completion of $\mathcal{S}$. If $\Gamma \vdash_{\mathcal{S}}^{C^pD^p} a : A$ then $\|\Gamma\| \vdash_{\mathcal{S}'} \|a\|_\Gamma : \|A\|_\Gamma$.*

Using Theorem 18.3, Lemma 23 and Theorem 25, we can now prove our normalization result for C^{p}D^p-PTSs.

Theorem 26 *Let $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P})$ and $\mathcal{S}' = (\boldsymbol{S}', \boldsymbol{A}', \boldsymbol{R}')$ be such that $\mathcal{S}'$ is a completion of $\mathcal{S}$. If the PTS $\lambda\mathcal{S}'$ is β-strongly normalizing, then the C^{p}D^p-PTS $\lambda^{C^pD^p}\mathcal{S}$ is $\beta\delta$-strongly normalizing.*

[6] Note that there are no requirements on $\boldsymbol{P}$ in the definition of completion.

Since ECC of [15] is β-strongly normalizing and is a completion of all systems of the extended λ-cube, Theorem 26 guarantees that all extended systems of the λ-cube are $\beta\delta$-strongly normalizing. Note that λC itself is not a completion since it has a topsort $\Box$.

4 Comparison of C^{p}D^p-PTSs with D-PTSs

In this section we show that the parameter mechanism in C^{p}D^p-PTSs can be seen as a system for abstraction and application that is weaker than the λ-calculus mechanism. We will make this precise by proving (in Theorem 31) that a C^{p}D^p-PTS with parametric specification $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \varnothing)$ is as powerful as any C^{p}D^p-PTS with parametric specification $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P})$ for which $(s_1, s_2) \in \boldsymbol{P}$ implies $(s_1, s_2, s_2) \in \boldsymbol{R}$. We call such a C^{p}D^p-PTS *parametrically conservative*; each C^{p}D^p-PTS with $\boldsymbol{P} \subseteq \boldsymbol{S} \times \boldsymbol{S}$ can be extended to a parametrically conservative one by taking its *parametric closure*.

Definition 27 Let $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P})$ be a parametric specification. $\mathcal{S}$ is *parametrically conservative* if for all $s_1, s_2 \in \boldsymbol{S}$, $(s_1, s_2) \in \boldsymbol{P}$ implies $(s_1, s_2, s_2) \in \boldsymbol{R}$.

$\mathrm{CL}(\mathcal{S})$, the *parametric closure* of $\mathcal{S}$, is defined as $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}', \boldsymbol{P})$, where $\boldsymbol{R}' = \boldsymbol{R} \cup \{(s_1, s_2, s_2) \mid s_1, s_2 \in \boldsymbol{S} \text{ and } (s_1, s_2) \in \boldsymbol{P}\}$.

Lemma 28 *Let $\mathcal{S}$ be a parametric specification. The following holds:*
1. $\mathrm{CL}(\mathcal{S})$ is parametrically conservative; and 2. $\mathrm{CL}(\mathrm{CL}(\mathcal{S})) = \mathrm{CL}(\mathcal{S})$.

Let $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P})$ be a parametric specification. If $\mathcal{S}$ is parametrically conservative, then each parametric rule (s_1, s_2) of $\mathcal{S}$ has a corresponding Π-formation rule (s_1, s_2, s_2). We show that this Π-formation rule can indeed take over the role of the parametric rule (s_1, s_2). This means that $\mathcal{S}$ has the same 'power' (see Theorem 31) as $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \varnothing)$. This even means that $\mathcal{S}$ has the same power as the D-PTS with specification $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R})$. In order to compare $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P})$ with $\mathcal{S}' = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \varnothing)$, we need to remove the parameters from the syntax of $\lambda^{C^pD^p}\mathcal{S}$. This can be obtained as follows:

- The parametric application in a term $c(b_1, \ldots, b_n)$ is replaced by a function application $cb_1 \cdots b_n$;
- A local parametric definition is translated by a parameter-free local definition, and the parameters are replaced by λ-abstractions;
- A global parametric definition is translated by a parameter-free global definition, and the parameters are replaced by λ-abstractions.

This leads to the following definitions (which can be extended to contexts in the obvious way):

Definition 29 We define the parameter-free translation $\{t\}$ of a term $t \in \mathcal{T}_P$ inductively as follows:
$\{a\} \equiv a$ if $a \equiv x$ or $a \equiv s$; $\quad \{c(b_1, \ldots, b_n)\} \equiv c\,\{b_1\} \cdots \{b_n\}$; $\quad \{ab\} \equiv \{a\}\,\{b\}$;
$\{\mathcal{O}x{:}A.B\} \equiv \mathcal{O}x{:}\{A\}\,.\,\{B\}$ if $\mathcal{O}$ is λ or Π;
$\{c(\Delta){=}a{:}A \texttt{ in } b\} \equiv c{=}\{\lambda\,\Delta.a\}:\{\prod \Delta.A\} \texttt{ in } \{b\}$.

The mapping $\{\_\}$ maintains β-reduction. A δ-reduction is translated into a δ-reduction followed by zero or more β-reductions. These β-reductions take over the n substitutions that are needed in a δ-reduction $c(b_1, \ldots, b_n) \rightarrow_\delta a[x_i{:=}b_i]_{i=1}^n$. Note that the restriction on the formation of parameters induced by the parametric rules $\boldsymbol{P}$ prevents the creation of illegal abstractions $\Pi\Delta.A$ with A a topsort.

Lemma 30 *1. For $a, b \in \mathcal{T}_P$: $\{a[x{:=}b]\} \equiv \{a\}\,[x{:=}\,\{b\}]$;*
2. If $a \rightarrow_\beta a'$ then $\{a\} \rightarrow_\beta^+ \{a'\}$;
3. If $\Gamma \vdash a \rightarrow_\delta a'$ then there is a'' such that $\{\Gamma\} \vdash \{a\} \rightarrow_\delta^+ a'' \twoheadrightarrow_\beta \{a'\}$;
4. If $\Gamma \vdash a \twoheadrightarrow_{\beta\delta} a'$ then $\{\Gamma\} \vdash \{a\} \twoheadrightarrow_{\beta\delta} \{a'\}$.

Now we show that the parameter-free translation $\{\_\}$ embeds the C^{p}D^p-PTS with parametric specification $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P})$ in the C^{p}D^p-PTS with parametric specification $\mathcal{S}' = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \varnothing)$, provided that $\mathcal{S}$ is parametrically conservative.

Thus we can conclude that the (restrictive) use of parameters does not yield a stronger type system than using abstraction, application and unparameterized constants and definitions only.

Theorem 31 *Let $\mathcal{S} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P})$ be a parametric specification. Assume that $\mathcal{S}$ is parametrically conservative. Let $\mathcal{S}' = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{R}, \varnothing)$. Then $\Gamma \vdash_{\mathcal{S}}^{C^pD^p} a : A$ implies $\{\Gamma\} \vdash_{\mathcal{S}'}^{CD} \{a\} : \{A\}$.*

PROOF: Induction on the derivation of $\Gamma \vdash_{\mathcal{S}}^{C^pD^p} a : A$. ⊠

Since unparameterized constants can be mimicked by dedicated variables, Theorem 31 can be paraphrased as "if $\mathcal{S}$ and $\mathcal{S}'$ are parametric specifications such that $\mathcal{S}'$ is $\mathrm{CL}(\mathcal{S})$ with $\boldsymbol{P}$ replaced by $\varnothing$, then type derivations in $\lambda^{C^pD^p}\mathcal{S}$ can be mapped to type derivations in the D-PTS $\lambda\mathcal{S}'$".

Now [17] shows that type derivations in a D-PTS $\lambda\mathcal{S}'$ can be mapped to type derivations in the PTS which is a completion of the D-PTS. We conclude that parametrically conservative C^{p}D^p-PTSs are conservative extensions of PTSs.

Note however that, in order to mimic type derivations in $\lambda^{C^pD^p}\mathcal{S}$ in a PTS, we need to strengthen the system twice: first by considering the parametric closure and then by considering a completion of the D-PTS.

5 Conclusion

In recent literature, extensions of Pure Type Systems with *unparameterized* definitions and with parameterized constants have been proposed. However, parameterized constants and definitions are required for PTSs in order to be suitable for studying semantics and implementations of theorem provers and programming languages and for reasoning about mathematics, since these all depend heavily on the use of parameterized constants as well as parameterized definitions.

In this paper we studied an extension of PTSs with both parameterized constants and parameterized definitions. Extending the existing theory we showed that our extension has all the desired properties and yields well-behaved type systems.

References

1. S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors. *Handbook of Logic in Computer Science, Volume 2: Background: Computational Structures*. Oxford University Press, 1992.
2. H.P. Barendregt. Lambda calculi with types. In [1], pages 117–309. Oxford University Press, 1992.
3. L.S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the Automath system*. PhD thesis, Eindhoven University of Technology, 1977. Published as Mathematical Centre Tracts nr. 83 (Amsterdam, Mathematisch Centrum, 1979).
4. S. Berardi. Towards a mathematical analysis of the Coquand-Huet calculus of constructions and the other systems in Barendregt's cube. Technical report, Dept. of Computer Science, Carnegie-Mellon University and Dipartimento Matematica, Universita di Torino, 1988.
5. R. Bloo, F. Kamareddine, and R. Nederpelt. The Barendregt Cube with Definitions and Generalised Reduction. *Information and Computation*, 126(2):123–143, 1996.
6. V.A.J. Borghuis. Modal Pure Type Systems. *Journal of Logic, Language, and Information*, 7:265–296, 1998.
7. N.G. de Bruijn. Reflections on Automath. Eindhoven University of Technology, 1990. Also in [16], pages 201–228.
8. A. Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.
9. D.T. van Daalen. A description of Automath and some aspects of its language theory. In P. Braffort, editor, *Proceedings of the Symposium APLASM*, volume I, pages 48–77, 1973. Also in [16], pages 101–126.
10. J.H. Geuvers, F. Wiedijk, J. Zwanenburg, R. Pollack, and H. Barendregt. Personal communication on the "Fundamental Theorem of Algebra" project. FTA web page available at `http : //www.cs.kun.nl/gi/projects/fta/index.html`.
11. F. Kamareddine, L. Laan, and R.P. Nederpelt. Refining the Barendregt cube using parameters. *Fifth International Symposium on Functional and Logic Programming, FLOPS 2001*, Lecture Notes in Computer Science:375–389, 2001.
12. T. Laan. *The Evolution of Type Theory in Logic and Mathematics*. PhD thesis, Eindhoven University of Technology, 1997.
13. T. Laan, R. Bloo, F. Kamareddine, and R. Nederpelt. Parameters in pure type systems. Technical Report 00-18, TUE Computing Science Reports, Eindhoven University of Technology, 2000. Available from `http : //www.win.tue.nl/~bloo/parameter − report.ps.gz`.
14. Twan Laan and Michael Franssen. Parameters for first order logic. *Logic and Computation*, 2001.
15. Z. Luo. *An Extended Calculus of Constructions*. PhD thesis, 1990.
16. R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors. *Selected Papers on Automath*. Studies in Logic and the Foundations of Mathematics **133**. North-Holland, Amsterdam, 1994.
17. P. Severi and E. Poll. Pure type systems with definitions. In A. Nerode and Yu.V. Matiyasevich, editors, *Proceedings of LFCS'94 (LNCS* **813***)*, pages 316–328, New York, 1994. LFCS'94, St. Petersburg, Russia, Springer Verlag.
18. J. Terlouw. Een nadere bewijstheoretische analyse van GSTT's. Technical report, Department of Computer Science, University of Nijmegen, 1989.
19. R. de Vrijer. A direct proof of the finite developments theorem. *The Journal of Symbolic Logic*, 50(2):339–343, 1985.