JFP **13** (2): 317–338, March 2003. © 2003 Cambridge University Press
DOI: 10.1017/S0956796802004525 Printed in the United Kingdom

PAL$^+$: *a lambda-free logical framework*

ZHAOHUI LUO
Department of Computer Science, University of Durham,
South Road, Durham DH1 3LE, UK
URL: http://www.dur.ac.uk/zhaohui.luo/

Abstract

A lambda-free logical framework takes parameterisation and definitions as the basic notions to provide schematic mechanisms for specification of type theories and their use in practice. The framework presented here, PAL$^+$, is a logical framework for specification and implementation of type theories, such as Martin-Löf's type theory or UTT. As in Martin-Löf's logical framework (Nordström *et al.*, 1990), computational rules can be introduced and are used to give meanings to the declared constants. However, PAL$^+$ only allows one to talk about the concepts that are intuitively in the object type theories: types and their objects, and families of types and families of objects of types. In particular, in PAL$^+$, one cannot directly represent families of families of entities, which could be done in other logical frameworks by means of lambda abstraction. PAL$^+$ is in the spirit of de Bruijn's PAL for Automath (de Bruijn, 1980). Compared with PAL, PAL$^+$ allows one to represent parametric concepts such as families of types and families of non-parametric objects, which can be used by themselves as totalities as well as when they are fully instantiated. Such parametric objects are represented by local definitions (let-expressions). We claim that PAL$^+$ is a correct meta-language for specifying type theories (e.g., dependent type theories), as it has the advantage of exactly capturing the intuitive concepts in object type theories, and that its implementation reflects the actual use of type theories in practice. We shall study the meta-theory of PAL$^+$ by developing its typed operational semantics and showing that it has nice meta-theoretic properties.

1 Motivations and introduction

A lambda-free logical framework takes parameterisation and definitions as the basic notions to provide schematic mechanisms for specification of type theories and their use in practice. The reasons to consider lambda-free logical frameworks include:

- Parametric constants and definitions and the associated operations of instantiation (substitution or cut) are more basic and arguably simpler notions and mechanisms than that of lambda-abstraction as found in other logical frameworks such as Martin-Löf's logical framework (Nordström *et al.*, 1990). A parametrically defined entity represents a family of entities, rather than a functional operation.
- The user of a proof system based on a lambda-free framework does not have to understand the meta-level lambda-abstraction that can be used to represent concepts such as families of families of entities, which do not exist in object type theories. Rather, one only has to grasp concepts of the object type theory and the definitional mechanism.

- The introduction of a lambda-free logical framework makes clear that a logical framework is a *meta-language* that provides the schematic mechanisms for specifying type theories and the definitional mechanism for pragmatic use. It is worth remarking that such mechanisms are necessary for any framework to be used in practice, with or without λ-abstraction. When an object type theory has types of functions, say Π-types, there is usually a confusion between the object-level functions and the meta-level functional operations, if the latter exist in the meta-framework. For example, in systems like ALF (Magnusson & Nordström, 1994), one tends to use the meta-level functional operations as functional programs while ignoring the object level functions.

The logical framework presented here, PAL$^+$, is such a framework in the spirit of de Bruijn's PAL for Automath (de Bruijn, 1980).

PAL$^+$ is a logical framework for specification and implementation of type theories such as Martin-Löf's type theory (Nordström *et al.*, 1990) and UTT (Luo, 1994). As in Martin-Löf's logical framework (Nordström *et al.*, 1990), computational rules can be introduced and are used to give meanings to the declared constants. However, PAL$^+$ only allows one to talk about the concepts that are intuitively in the object type theories: types and their objects, and families of types and families of objects of types. In particular, in PAL$^+$, one cannot directly represent families of families of entities, which could be done in other logical frameworks by means of lambda abstraction. Compared with PAL, PAL$^+$ allows one to represent parametric concepts such as families of types and families of non-parametric objects, which can be used by themselves (as totalities) as well as when they are fully instantiated. An implementation of a proof development system based on PAL$^+$ can truly reflect the intended use of a type theory. After a type theory is specified (and implemented), the user is concerned only with the object type theory and uses the definitional mechanism for abbreviation.

Parametric objects are represented as let-expressions. One of the distinctive features of PAL$^+$, compared with other logical frameworks, is that it takes definitions rather than lambda abstractions as basic. Let-expressions do not only represent local definitions, but parametric objects. The meta-theory for PAL$^+$, therefore, is new and the first for such a calculus as far as we know. We develop typed operational semantics (Goguen, 1994) for PAL$^+$ and show that PAL$^+$ has nice meta-theoretic properties.

The following section gives a formal presentation of PAL$^+$: its parameterisation mechanism and definitional mechanism, together with some informal explanations. We also explain how parametric abstractions, which represent families of types or objects, can be represented as let-expressions. Section 3 shows how PAL$^+$ can be used in specification of type theories. In sections 4 and 5, we consider the meta-theoretic properties of the logical framework PAL$^+$ by developing its typed operational semantics and proving its computational properties such as Church–Rosser, Subject Reduction and Strong Normalisation. In the Conclusion, we briefly discuss some issues about implementation of PAL$^+$ and possible further extensions.

2 PAL$^+$

In PAL$^+$, we have objects and kinds. Kinds include non-parametric kinds and parametric kinds. Non-parametric kinds are of the form **Type** or $El(A)$. Parametric kinds of the form $(\Delta)T$, where Δ is a non-empty context of the form $x_1{:}K_1, ..., x_n{:}K_n$ and T is a non-parametric kind. Non-parametric objects, objects of non-parametric kinds, represent types and objects of types; parametric objects, objects of parametric kinds, represent families of types and families of non-parametric objects. Let-expressions, representing local definitions, can be used to form kinds and objects. In particular, parametric objects are represented by let-expressions.

In the following, we give a formal description of PAL$^+$, with some informal explanations.

2.1 Terms

Terms are either object expressions or kind expressions. For presentational purposes and meta-theoretic reasons, we introduce terms associated with arities, which are natural numbers. The arity of an object expression indicates the number of arguments it should take when forming a term of instantiation. The arity of a kind expression indicates the number of arguments its objects should take. We write $Var(i)$ and $DV(i)$ for the sets of ordinary variables and definitional variables with arity i, respectively. We assume that $Var(i)$ and $DV(j)$ be all disjoint. Furthermore,

$$Var = \bigcup_{i\in\omega} Var(i) \quad \textit{and} \quad DV = \bigcup_{i\in\omega} DV(i).$$

Definition 2.1 (*expressions, contexts, pure contexts, and terms*)
The following are defined simultaneously by structural induction:

1. The set of object expressions with arity $i \in \omega$, $Obj(i)$, is defined as follows:
   - $Var(i) \subseteq Obj(i)$.
   - $DV(i) \subseteq Obj(i)$.
   - $f[k_1, ..., k_n] \in Obj(0)$ if $f \in Obj(n)$ and k_i are object expressions.
   - **let** $v[\Delta] = t{:}T$ **in** $k \in Obj(i)$ if $k \in Obj(i)$, $t \in Obj(0)$, $T \in Kind(0)$, Δ is a pure context of length n, and $v \in DV(n)$.
2. The set of kind expressions with arity $i \in \omega$, $Kind(i)$, is defined as follows:
   - **Type** $\in Kind(0)$.
   - $El(A) \in Kind(0)$ if $A \in Obj(0)$.
   - $(\Delta)T \in Kind(i)$ if Δ is a pure context of length i and $T \in Kind(0)$.
   - **let** $v[\Delta] = t{:}T$ **in** $K \in Kind(i)$ if $K \in Kind(i)$, $t \in Obj(0)$, $T \in Kind(0)$, Δ is a pure context of length n, and $v \in DV(n)$.
3. A pure context (Δ) is a sequence of entries of the form $x{:}K$ such that for some i, $x \in Var(i)$ and $K \in Kind(i)$. A context (Γ) is a sequence of entries of the form $x{:}K$ (as above) or $v[\Delta] = t{:}T$, where $t \in Obj(0)$, $T \in Kind(0)$, Δ is a pure context of length n, and $v \in DV(n)$. The length of a context, $l(\Gamma)$, is the number of its entries.

The set of terms with arity i, $Term(i)$, is defined as $Obj(i) \cup Kind(i)$. We shall write $Arity(M)$ for the arity of term M.

Remark By definition, every term has a unique arity. Substitutions preserve the arity of a term. More precisely, for terms M and k and variable z, if $Arity(k) = Arity(z)$, then $[k/z]M$ is a term and $Arity([k/z]M) = Arity(M)$.

Notation The following notations are used:

- When Δ is the empty context ($\langle\rangle$, which does not have any entry), we write $v = t{:}T$ for $v[\Delta] = t{:}T$ in a context and **let** $v = t{:}T$ **in** M for the let-expression **let** $v[\Delta] = t{:}T$ **in** M. By our convention, $f[]$ is taken as the same as f, and $()T$ (or $(\langle\rangle)T$) is taken as the same as T.
- A pure context Δ of the form $x_1{:}K_1, ..., x_n{:}K_n$ is often abbreviated as $\bar{x}{:}\bar{K}$ and we also use Δ_i to stand for $x_1{:}K_1, ..., x_i{:}K_i$. We shall sometimes write $\Delta_{\bar{x}}$ for Δ to indicate that $\Delta \equiv \bar{x}{:}\bar{K}$ for some $\bar{K}$.
- We use $Var(M)$ and $DV(M)$, defined inductively on the term structure, to denote the sets of free ordinary variables and free definitional variables in term M, respectively. We use $FV(M)$, defined to be $Var(M) \cup DV(M)$, to denote the set of free variables in term M. These extend to contexts as well.

We identify terms which are α-convertible and use $\equiv$ for syntactic equality. In particular, in **let** $v[\Delta_{\bar{x}}] = t{:}T$ **in** M, v is bound in M and $\bar{x}$ are bound in t and T. Therefore, terms with changes of such bound variables are identified.

2.2 *Judgement forms*

The judgement forms are, where Γ is a context, K and K' are kind expressions, and k and k' are object expressions:

- Γ **valid** – Γ is a valid context.
- $\Gamma \vdash K$ **kind** – K is a kind in Γ.
- $\Gamma \vdash k : K$ – k is an object of kind K in Γ.
- $\Gamma \vdash K = K'$ – K and K' are equal kinds.
- $\Gamma \vdash k = k' : K$ – k and k' are equal objects of kind K.

Notation

- For $\Delta \equiv x_1{:}K_1, ..., x_n{:}K_n$ and $\Delta' \equiv x_1{:}K_1', ..., x_n{:}K_n'$, we write $\Gamma \vdash \Delta = \Delta'$ for the sequence of judgements $\Gamma, \Delta_{i-1} \vdash K_i = K_i'$ $(i = 1, ..., n)$.
- We shall write $\Gamma \vdash \bar{k} : \bar{K}$ for the sequence of judgements $\Gamma \vdash k_1 : K_1$, $\Gamma \vdash k_2 : [k_1/x_1]K_2$, ..., $\Gamma \vdash k_n : [k_{n-1}/x_{n-1}]...[k_1/x_1]K_n$ and similarly the notation $\Gamma \vdash \bar{k} = \bar{k}' : \bar{K}$ stands for the sequence of judgements $\Gamma \vdash k_1 = k_1' : K_1$, $\Gamma \vdash k_2 = k_2' : [k_1/x_1]K_2$, ..., $\Gamma \vdash k_n = k_n' : [k_{n-1}/x_{n-1}]...[k_1/x_1]K_n$.
- The simultaneous substitution $[\bar{k}/\bar{x}]M$ stands for $[k_n/x_n]...[k_1/x_1]M$; note that when this notation is used, we can always assume that $\bar{x} \notin FV(k_i)$ and so the order of substitutions does not matter.

Contexts and assumptions

$$\frac{}{\langle\rangle\ \textbf{valid}} \qquad \frac{\Gamma \vdash K\ \textbf{kind} \quad x \notin FV(\Gamma)\ \textit{and}\ x \in Var(Arity(K))}{\Gamma, x{:}K\ \textbf{valid}} \qquad \frac{\Gamma, x{:}K, \Gamma'\ \textbf{valid}}{\Gamma, x{:}K, \Gamma' \vdash x : K}$$

General equality rules

$$\frac{\Gamma \vdash K\ \textbf{kind}}{\Gamma \vdash K = K} \qquad \frac{\Gamma \vdash K = K'}{\Gamma \vdash K' = K} \qquad \frac{\Gamma \vdash K = K' \quad \Gamma \vdash K' = K''}{\Gamma \vdash K = K''}$$

$$\frac{\Gamma \vdash k : K}{\Gamma \vdash k = k : K} \qquad \frac{\Gamma \vdash k = k' : K}{\Gamma \vdash k' = k : K} \qquad \frac{\Gamma \vdash k = k' : K \quad \Gamma \vdash k' = k'' : K}{\Gamma \vdash k = k'' : K}$$

Equality typing rules

$$\frac{\Gamma \vdash k : K \quad \Gamma \vdash K = K'}{\Gamma \vdash k : K'} \qquad \frac{\Gamma \vdash k = k' : K \quad \Gamma \vdash K = K'}{\Gamma \vdash k = k' : K'}$$

The kind Type

$$\frac{\Gamma\ \textbf{valid}}{\Gamma \vdash \textbf{Type kind}} \qquad \frac{\Gamma \vdash A : \textbf{Type}}{\Gamma \vdash El(A)\ \textbf{kind}} \qquad \frac{\Gamma \vdash A = B : \textbf{Type}}{\Gamma \vdash El(A) = El(B)}$$

Fig. 1. The basic rules of PAL$^+$.

Substitution rules

$$\frac{\Gamma, x{:}K, \Gamma'\ \textbf{valid} \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma'\ \textbf{valid}}$$

$$\frac{\Gamma, x{:}K, \Gamma' \vdash K'\ \textbf{kind} \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K'\ \textbf{kind}} \qquad \frac{\Gamma, x{:}K, \Gamma' \vdash K'\ \textbf{kind} \quad \Gamma \vdash k = k' : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K' = [k'/x]K'}$$

$$\frac{\Gamma, x{:}K, \Gamma' \vdash k' : K' \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]k' : [k/x]K'} \qquad \frac{\Gamma, x{:}K, \Gamma' \vdash k' : K' \quad \Gamma \vdash k_1 = k_2 : K}{\Gamma, [k_1/x]\Gamma' \vdash [k_1/x]k' = [k_2/x]k' : [k_1/x]K'}$$

$$\frac{\Gamma, x{:}K, \Gamma' \vdash K' = K'' \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]K' = [k/x]K''} \qquad \frac{\Gamma, x{:}K, \Gamma' \vdash k' = k'' : K' \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]k' = [k/x]k'' : [k/x]K'}$$

Fig. 2. The substitution rules of PAL$^+$.

2.3 Basic rules

The basic rules are given in figures 1 and 2, the latter of which contains the substitution rules. Formally, these rules are the general inference rules of the logical framework LF (see figure 9.1 in Chapter 9 of Luo, 1994).[1]

The kind **Type** represents the conceptual universe of types to be introduced, and for each type A of kind **Type**, the kind $El(A)$ is the kind of objects of type A.

[1] LF is a typed version of Martin-Löf's logical framework (Nordström *et al.*, 1990) in that the form of abstraction $[x{:}K]k$ has type label K, rather than just $[x]k$. We should point out that LF is different from the Edinburgh Logical Framework (ELF) (Harper *et al.*, 1987). Though quite similar formally, the intended ways of use are very different.

Formation rules for parametric kinds

$$\frac{\Gamma, \Delta \vdash T\ \mathbf{kind} \quad (\Delta)T\ \textit{is a term}}{\Gamma \vdash (\Delta)T\ \mathbf{kind}}$$

$$\frac{\Gamma \vdash \Delta = \Delta' \quad \Gamma, \Delta \vdash T = T' \quad (\Delta)T\ \textit{and}\ (\Delta')T'\ \textit{are terms}}{\Gamma \vdash (\Delta)T = (\Delta')T'}$$

Instantiation rules for parametric objects

$$\frac{\Gamma \vdash f : (\bar{x}{:}\bar{K})T \quad \Gamma \vdash \bar{k} : \bar{K}}{\Gamma \vdash f[\bar{k}] : [\bar{k}/\bar{x}]T} \qquad \frac{\Gamma \vdash f = f' : (\bar{x}{:}\bar{K})T \quad \Gamma \vdash \bar{k} = \bar{k}' : \bar{K}}{\Gamma \vdash f[\bar{k}] = f'[\bar{k}'] : [\bar{k}/\bar{x}]T}$$

Fig. 3. Rules for parametric kinds in PAL$^+$.

2.4 Parametric kinds and instantiations

Parametric objects represent families of types or families of non-parametric objects. They can either be used as a totality or when they are fully instantiated.

The rules for parametric kinds of the form $(\Delta)T$ are given in figure 3. Note that a parametric entity of a parametric kind cannot be used by partial instantiation. Only when given appropriate indexing objects $\bar{k}$, can a parametric object f, i.e. an object of a parametric kind, be instantiated into $f[\bar{k}]$.

Besides variables of a parametric kind, parametric objects also include parametric constants (introduced when specifying an object type theory) and some let-expressions (see below).

2.5 Definitions in PAL$^+$

We introduce in PAL$^+$ both global definitions of the form $v[\Delta] = t{:}T$ as entries in contexts and local definitions or let-expressions of the form **let** $v[\Delta] = t$ **in** M, where variables $\bar{x}$ in $\Delta_{\bar{x}}$ are bound in t and T and the definitional variable v is bound in M.

2.5.1 Global definitions

Global definitions can be introduced into contexts and used by means of the rules in figure 4. We also have substitution rules in figure 5, where in the last rule, J is of the form K **kind**, $k : K$, $K = K'$, or $k = k' : K$.

Remark Several remarks are in order:

- Note that, in the introduction rule for global definitions, we require T to be a kind of arity 0, i.e. it is equal to either **Type** or $El(A)$. Hence the body of a global definition must be a type or an object of a type. Also, when Δ is empty, the rules specialise into those for non-parametric kinds.
- The definiendum of a global definition can either be used when it is fully applied, or as a totality.

Introduction rule for global definitions

$$\frac{\Gamma, \Delta \vdash t : T \quad v \notin FV(\Gamma) \quad \Gamma, v[\Delta] = t{:}T \textit{ is a context}}{\Gamma, v[\Delta] = t{:}T \ \textbf{valid}}$$

Typing and equality rules for global definitions

$$\frac{\Gamma, v[\Delta] = t{:}T, \Gamma' \ \textbf{valid}}{\Gamma, v[\Delta] = t{:}T, \Gamma' \vdash v : (\Delta)T} \quad \frac{\Gamma, v[\Delta] = t{:}T, \Gamma' \ \textbf{valid}}{\Gamma, v[\Delta] = t{:}T, \Gamma' \vdash v = \textbf{let}\ v[\Delta] = t{:}T \ \textbf{in}\ v : (\Delta)T}$$

Fig. 4. Rules for global definitions.

Substitution rules for global definitions

$$\frac{\Gamma, v[\Delta] = t{:}T, \Gamma' \ \textbf{valid}}{\Gamma, [\textbf{let}\ v[\Delta] = t{:}T \ \textbf{in}\ v/v]\Gamma' \ \textbf{valid}}$$

$$\frac{\Gamma, v[\Delta] = t{:}T, \Gamma' \vdash J}{\Gamma, [\textbf{let}\ v[\Delta] = t{:}T \ \textbf{in}\ v/v]\Gamma' \vdash [\textbf{let}\ v[\Delta] = t{:}T \ \textbf{in}\ v/v]J}$$

Fig. 5. Substitution rules for global definitions.

- The "meaning" of a globally defined entity v is given directly by means of let-expressions of the form **let** $v[\Delta] = t{:}T$ **in** v.

2.5.2 Local definitions

Local definitions, or let-expressions, are introduced by the let-introduction rules in figure 6. They abide by the congruence rules in figure 6 and the equality rules in figure 7.

2.6 Parametric abstraction

The let-expressions in PAL$^+$ play a role of "parametric abstraction" as well as local definitions. In particular, when Δ is not empty, the term **let** $v[\Delta] = t{:}T$ **in** v can be viewed as a form of abstraction – parametric abstraction. We may introduce a new notation: a parametric abstraction is of the form $[\Delta]t$ and represents either a family of types or non-parametric objects, indexed by (sequences of) objects of Δ. The variables in Δ are bound variables.

One may introduce parametric abstractions independently by adding the following rules (this was the case in Luo (2000)):

$$\frac{\Gamma, \Delta \vdash t : T \quad T \in Kind(0)}{\Gamma \vdash [\Delta]t : (\Delta)T} \quad \frac{\Gamma \vdash \Delta = \Delta' \quad \Gamma, \Delta \vdash t = t' : T \quad T \in Kind(0)}{\Gamma \vdash [\Delta]t = [\Delta']t' : (\Delta)T}$$

$$\frac{\Gamma, \bar{x}{:}\bar{K} \vdash t : T \quad \Gamma \vdash \bar{k} : \bar{K} \quad T \in Kind(0)}{\Gamma \vdash ([\bar{x}{:}\bar{K}]t)[\bar{k}] = [\bar{k}/\bar{x}]t : [\bar{k}/\bar{x}]T} \quad \frac{\Gamma \vdash f : (\bar{x}{:}\bar{K})T \quad \bar{x} \notin FV(f)}{\Gamma \vdash [\bar{x}{:}\bar{K}]f[\bar{x}] = f : (\bar{x}{:}\bar{K})T}$$

let-introduction rules

$$\frac{\Gamma, v[\Delta] = t{:}T \vdash K\ \mathbf{kind}}{\Gamma \vdash \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ K\ \mathbf{kind}} \qquad \frac{\Gamma, v[\Delta] = t{:}T \vdash k : K}{\Gamma \vdash \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ k : \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ K}$$

Congruence rules for let-expressions

$$\frac{\Gamma \vdash \Delta = \Delta' \quad \Gamma, \Delta \vdash T = T' \quad \Gamma, \Delta \vdash t = t' : T \quad \Gamma, v[\Delta] = t{:}T \vdash K = K'}{\Gamma \vdash (\mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ K) = (\mathbf{let}\ v[\Delta'] = t'{:}T'\ \mathbf{in}\ K')}$$

$$\frac{\Gamma \vdash \Delta = \Delta' \quad \Gamma, \Delta \vdash T = T' \quad \Gamma, \Delta \vdash t = t' : T \quad \Gamma, v[\Delta] = t{:}T \vdash k = k' : K}{\Gamma \vdash (\mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ k) = (\mathbf{let}\ v[\Delta'] = t'{:}T'\ \mathbf{in}\ k') : \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ K}$$

Fig. 6. Introduction and congruence rules for let-expressions.

Equality rules for let-expressions

(let_β)
$$\frac{\Gamma, v[\bar{x}{:}\bar{K}] = t{:}T\ \mathbf{valid} \quad \Gamma \vdash \bar{k} : \bar{K}}{\Gamma \vdash (\mathbf{let}\ v[\bar{x}{:}\bar{K}] = t{:}T\ \mathbf{in}\ v)[\bar{k}] = [\bar{k}/\bar{x}]t : [\bar{k}/\bar{x}]T}$$

(let_η)
$$\frac{\Gamma, v[\Delta_{\bar{x}}] = g[\bar{x}]{:}T \vdash k : K \quad \Gamma \vdash g : (\Delta)T}{\Gamma \vdash \mathbf{let}\ v[\Delta] = g[\bar{x}]{:}T\ \mathbf{in}\ k = [g/v]k : [g/v]K}$$

(let_η^K)
$$\frac{\Gamma, v[\Delta_{\bar{x}}] = g[\bar{x}]{:}T \vdash K\ \mathbf{kind} \quad \Gamma \vdash g : (\Delta)T}{\Gamma \vdash \mathbf{let}\ v[\Delta] = g[\bar{x}]{:}T\ \mathbf{in}\ K = [g/v]K}$$

(let_d)
$$\frac{\Gamma, v[\Delta] = t{:}T \vdash k : K}{\Gamma \vdash \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ k = [\mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ v/v]k : \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ K}$$

(let_d^K)
$$\frac{\Gamma, v[\Delta] = t{:}T \vdash K\ \mathbf{kind}}{\Gamma \vdash \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ K = [\mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ v/v]K}$$

Fig. 7. Equality rules for let-expressions.

However, parametric abstractions are a special form of let-expressions. The following definitional rule defines parametric abstraction in terms of let-expressions in PAL$^+$:

$$\frac{\Gamma, v[\Delta] = t{:}T\ \mathbf{valid}}{\Gamma \vdash [\Delta]t = \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ v : (\Delta)T}$$

Remark One might want to take parametric abstraction and its application as basic and define let-expressions by means of parametric abstraction. There is some technical difficulty in doing so (note that let-expressions have kind information T which is not present in parametric abstraction). However, more importantly, we remark that let-expressions are more general than parametric abstractions as they can be used for all expressions including parametric kinds. Furthermore, local definitions are useful in any proof development or programming environment. It is therefore natural to take let-expressions as basic while taking parametric abstraction as a defined notion.

2.7 Simple properties and distribution laws for let-expressions

The rules of PAL$^+$, as given in figure 1 to figure 7, respect the definition of syntactic notions of term, context, etc. in Definition 2.1.

Lemma 2.2
The following properties hold:

- If Γ **valid**, then Γ is a context.
- If $\Gamma \vdash K$ **kind**, then Γ is a context and K is a kind expression.
- If $\Gamma \vdash K = K'$, then Γ is a context and K and K' are kind expressions of the same arity.
- If $\Gamma \vdash k : K$, then Γ is a context, k is an object expression, K is a kind expression, and k and K have the same arity.
- If $\Gamma \vdash k = k' : K$, then Γ is a context, k and k' are object expressions, K is a kind expression, and k, k', and K have the same arity.

Let-expressions satisfy a number of distribution laws, which say that local definitions can be distributed for all structured expressions. For instance, when $v \notin FV(M)$, **let** $v[\Delta] = t{:}T$ **in** M is computationally equal to M. (See Luo (2000) for details, where distribution rules are taken to replace the equality rules in figure 7.) An example of such distribution rules is:

$$\frac{\Gamma, v[\Delta] = t{:}T \vdash (\Delta')T' \ \mathbf{kind}}{\Gamma \vdash (\mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ (\Delta')T') = (\mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ \Delta')\mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ T'}$$

In PAL$^+$ as presented in this paper, the distribution rules are all admissible.

3 Specification of type theories in PAL$^+$

As in Martin-Löf’s type theory (Nordström *et al.*, 1990), we specify type theories in the logical framework PAL$^+$. One of the key observations is that we can specify type theories with the simpler logical framework without arbitrary lambda-abstraction. For example, all of the types in UTT (Luo, 1994) can be specified, including the impredicative universe of logical propositions, the inductive types and inductive families covered by the inductive schemata, and predicative universes. Similarly, Martin-Löf’s type theory can be specified in PAL$^+$ as well.

In general, a specification of a type theory in PAL$^+$ will consist of a collection of declarations of new constants (either non-parametric or parametric) and a collection of associated computational equality rules. Like other parametric objects, a parametric constant cannot be used by partial instantiation. Such declarations of constants and equalities amount to extensions of (an existing type theory specified in) PAL$^+$ by new rules. One should, of course, make sure that the new rules lead to a type theory that has good properties. For example, inductive types with strictly positive constructors can be specified. We do not consider such issues here.

Given a kind K, if one introduces a constant f by declaring

$$f : K,$$

it has the effect of introducing the following rule:

$$\frac{\Gamma\ \textbf{valid}}{\Gamma \vdash f : K}$$

Computational equality can only be introduced between two objects of a type, or between two types.[2] If one introduces a computation rule by asserting

$$t = t' : T \quad where\ k_i : K_i\ (i = 1, ..., n),$$

it extends the type theory with the following rule:

$$\frac{\Gamma \vdash k_i : K_i\ (i = 1, ..., n) \quad \Gamma \vdash T\ \textbf{kind} \quad T \in Kind(0)}{\Gamma \vdash t = t' : T}$$

In the following, we give several examples of introducing type constructors and their associated operators as constants. We omit El to write A for $El(A)$ in the examples.

Example 3.1
The type of natural numbers can be introduced as follows:

$$\begin{array}{lcl} N & : & \textbf{Type} \\ 0 & : & N \\ s & : & (x{:}N)\ N \\ R & : & (C{:}(x{:}N)\textbf{Type},\ c{:}C[0],\ f{:}(x{:}N, y{:}C[x])C[s[x]],\ z{:}N)\ C[z] \end{array}$$

The corresponding computation rules are:

$$\begin{array}{rcl} R[C, c, f, 0] & = & c\ :\ C[0] \\ R[C, c, f, s[x]] & = & f[x, R[C, c, f, x]]\ :\ C[s[x]] \end{array}$$

where $C{:}(x{:}N)\textbf{Type}$, $c{:}C[0]$, $f{:}(x{:}N, y{:}C[x])C[s[x]]$, and $x{:}N$. (We omit such where-clauses in the following examples.)

Example 3.2
The Π-types, $\Pi[A, B]$ for a type A and a family of types B, can be introduced as follows:

$$\begin{array}{lcl} \Pi & : & (A{:}\textbf{Type}, B{:}(x{:}A)\textbf{Type})\ \textbf{Type} \\ \lambda & : & (A{:}\textbf{Type}, B{:}(x{:}A)\textbf{Type}, f{:}(x{:}A)B[x])\ \Pi[A, B] \\ \mathscr{E}_\Pi & : & (A{:}\textbf{Type}, B{:}(x{:}A)\textbf{Type}, C{:}(F{:}\Pi[A, B])\textbf{Type}, \\ & & \quad f{:}(g{:}(x{:}A)B[x])C[\lambda[A, B, g]], \\ & & \quad z{:}\Pi[A, B]) \\ & & C[z] \end{array}$$

[2] This conforms to the restriction considered in Luo (1999), where LF is used to specify type theories.

The corresponding computation rule is:

$$\mathscr{E}_\Pi[A,B,C,f,\lambda[A,B,g]] \;=\; f[g] \;:\; C[\lambda[A,B,g]]$$

The following simple example shows how local definitions may be used.

Example 3.3
The application operator for Π-types

$$\textbf{app} \;:\; (A{:}\textbf{Type}, B{:}(x{:}A)\textbf{Type}, F{:}\Pi[A,B], x{:}A)\; B[x]$$

can be defined by means of local definitions as follows:

$$\begin{aligned}\textbf{app}[A,B,F,a] \;=\; & \textbf{let } C[G{:}\Pi[A,B]] = B[a]{:}\textbf{Type in}\\ & \textbf{let } f[g{:}(x{:}A)B[x]] = g[a]{:}B[a] \textbf{ in } \mathscr{E}_\Pi[A,B,C,f,F]\end{aligned}$$

Or alternatively, parametric abstractions, defined as special forms of let-expressions, may be used to define the same application operator as follows:

$$\textbf{app}[A,B,F,a] \;=_{\text{df}}\; \mathscr{E}_\Pi[A,B,[G{:}\Pi[A,B]]B[a],[g{:}(x{:}A)B[x]]g[a],F]$$

With the above definition, we can show the expected equalities hold. For example, we can show that the usual β-equality holds for the computational equality:

$$\textbf{app}[A,B,\lambda[A,B,g],a] = g[a].$$

Furthermore, for propositional equality $=_{\Pi[A,B]}$ (e.g. the Leibniz equality, which can be defined when we have an impredicative universe of logical propositions, or Martin-Löf's equality type defined by introducing a single constructor $eq[a]$ of type $a =_{\Pi[A,B]} a$), we can show that the logical η-rule holds, i.e. the following logical proposition is provable:

$$\lambda[A,B,[x{:}A]\textbf{app}[A,B,F,x]] =_{\Pi[A,B]} F.$$

Example 3.4
The family of types of vectors of objects of type A can be introduced as follows, where N is the type of natural numbers as introduced above:

$$\begin{aligned}
Vec \;&:\; (A{:}\textbf{Type}, x{:}N)\;\textbf{Type}\\
nil \;&:\; (A{:}\textbf{Type})\; Vec[A,0]\\
cons \;&:\; (A{:}\textbf{Type}, n{:}N, a{:}A, l{:}Vec[A,n])\; Vec[A,succ[n]]\\
\mathscr{E}_V \;&:\; (A{:}\textbf{Type},\; C{:}(n{:}N, v{:}Vec[A,n])\textbf{Type},\\
&\qquad c{:}C[0,nil[A]],\; f{:}(n{:}N, x{:}A, v{:}Vec[A,n], y{:}C[n,v])C[succ[n],cons[A,x,v]],\\
&\qquad n{:}N,\; v{:}Vec[A,n])\\
&\quad C[n,v],
\end{aligned}$$

The corresponding computation rules are:

$$\begin{aligned}
&\mathscr{E}_V[A,C,c,f,0,nil[A]] \;=\; c \;:\; C[0,nil[A]],\\
&\mathscr{E}_V[A,C,c,f,succ[n],cons[A,n,a,v]]\\
&\quad = \; f[n,a,v,\mathscr{E}_V[A,C,c,f,n,v]] \;:\; C[succ[n],cons[A,n,a,v]].
\end{aligned}$$

Example 3.5
The W-types, $W[A,B]$ for a type A and a family of types B, can be introduced as follows:

$$\begin{array}{lll}
W & : & (A{:}\mathbf{Type}, B{:}(x{:}A)\mathbf{Type})\ \mathbf{Type} \\
sup & : & (A{:}\mathbf{Type}, B{:}(x{:}A)\mathbf{Type}, x{:}A, y{:}(v{:}B[x])W[A,B])\ W[A,B] \\
\mathscr{E}_W & : & (A{:}\mathbf{Type}, B{:}(x{:}A)\mathbf{Type}, C{:}(w{:}W[A,B])\mathbf{Type}, \\
& & \quad f{:}(x{:}A, y{:}(v{:}B[x])W[A,B], g{:}(v{:}B[x])C[y[v]])C[sup[A,B,x,y]], \\
& & \quad z{:}W[A,B]) \\
& & C[z]
\end{array}$$

The corresponding computation rule is:

$$\begin{array}{l}
\mathscr{E}_W[A,B,C,f,sup[A,B,x,y]] \\
\quad = \ f[x,y,[z{:}B[x]]\mathscr{E}_W[A,B,C,f,y[z]]] \ : \ C[sup[A,B,x,y]].
\end{array}$$

Special cases of W-types include the type of ordinals and various types of well-founded trees. Note that the notation of parametric abstraction is used in the computation rule above.

4 Typed operational semantics for PAL$^+$

In the next two sections, we study the meta-theory of PAL$^+$. In this section, we develop the typed operational semantics for PAL$^+$, which is taken as the basis for development of the meta-theory for PAL$^+$ in the next section.

Typed Operational Semantics (TOS) was developed for the type theory UTT in Goguen's thesis (Goguen, 1994), and a concise account of TOS for LF can be found inGoguen 91999). In Luo (2000), we have developed TOS for PAL$^+$ with only parametric abstractions (and without global definitions or let-expressions.) In this paper, we take let-expressions as basic and develop the TOS and meta-theory. As far as we know, this is the first treatment of meta-theory concerning such a calculus with basic let-expressions (and η-rules).

4.1 TOS

The typed operational semantics for PAL$^+$ has the following judgement forms:

- $\models \Gamma \to \Gamma'$ – context Γ has normal form Γ'.
- $\Gamma \models K \to K'$ – kind K is a well-typed and has normal form K' in context Γ.
- $\Gamma \models k \to k_0 \to k' : K$ – k, k_0, and k' are of kind K in Γ, and k has weak-head normal form k_0 and normal form k'.

Notation We shall use the following notations:

- For $\Delta \equiv \bar{x}{:}\bar{K}$ and $\Delta' \equiv \bar{x}'{:}\bar{K}'$ of the same length, we shall use the notation $\Gamma \models \Delta \to \Delta'$ to stand for the sequence of judgements $\Gamma \models K_1 \to K_1'$, $\Gamma, \Delta_1 \models K_2 \to K_2'$, ..., $\Gamma, \Delta_{n-1} \models K_n \to K_n'$.

Contexts

$$\frac{}{\models \langle\rangle \to \langle\rangle} \qquad \frac{\begin{array}{c}\models \Gamma \to \Gamma' \quad \Gamma \models K \to K' \\ x \notin FV(\Gamma) \textit{ and } x \in Var(Arity(K))\end{array}}{\models \Gamma, x{:}K \to \Gamma', x{:}K'}$$

$$\frac{\begin{array}{l}\models \Gamma \to \Gamma' \quad \Gamma \models \Delta \to \Delta' \quad \Gamma, \Delta \models T \to T' \quad \Gamma, \Delta \models t \to t_0 \to t' : T' \\ v \notin FV(\Gamma),\ v \in DV(l(\Delta)) \textit{ and } T \in Kind(0)\end{array}}{\models \Gamma, v[\Delta] = t{:}T \to \Gamma', v[\Delta'] = t'{:}T'}$$

Kinds

$$\frac{\models \Gamma \to \Gamma'}{\Gamma \models \mathbf{Type} \to \mathbf{Type}} \qquad \frac{\Gamma \models A \to A_0 \to A' : \mathbf{Type}}{\Gamma \models El(A) \to El(A')}$$

$$\frac{\Gamma \models \Delta \to \Delta' \quad \Gamma, \Delta \models T \to T' \quad T \in Kind(0)}{\Gamma \models (\Delta)T \to (\Delta')T'}$$

Variables

$$\frac{\Gamma \models K \to K' \quad \models \Gamma, x{:}K, \Gamma' \to \Gamma_1}{\Gamma, x{:}K, \Gamma' \models x \to x \to x : K'}$$

$$\frac{\begin{array}{l}\Gamma \models \Delta \to \Delta' \quad \Gamma, \Delta \models T \to T' \\ \Gamma, v[\Delta] = t{:}T, \Gamma' \models \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ v \to k_0 \to k' : (\Delta')T'\end{array}}{\Gamma, v[\Delta] = t{:}T, \Gamma' \models v \to k_0 \to k' : (\Delta')T'}$$

Instantiations

$$\frac{\begin{array}{l}\Gamma \models f \to x \to x : (\bar{x}{:}\bar{K})T \quad x \in Var \textit{ and } \bar{x}{:}\bar{K} \not\equiv \langle\rangle \\ \Gamma \models \bar{k} \to \bar{k}_0 \to \bar{k}' : \bar{K} \quad \Gamma \models [\bar{k}/\bar{x}]T \to T'\end{array}}{\Gamma \models f[\bar{k}] \to x[\bar{k}] \to x[\bar{k}'] : T'}$$

$$\frac{\begin{array}{l}\Gamma \models f \to \mathbf{let}\ v[\bar{x}{:}\bar{K}] = t{:}T\ \mathbf{in}\ v \to f' : (\bar{x}{:}\bar{K}')T' \quad \Gamma \models \bar{k} \to \bar{k}_0 \to \bar{k}' : \bar{K}' \\ \Gamma \models [\bar{k}/\bar{x}]T \to T'' \quad \Gamma \models [\bar{k}/\bar{x}]t \to t_0 \to t' : T'' \quad \bar{x}{:}\bar{K} \not\equiv \langle\rangle\end{array}}{\Gamma \models f[\bar{k}] \to t_0 \to t' : T''}$$

Fig. 8. Basic TOS rules for PAL$^+$.

- For $\bar{k}$, $\bar{k}_0$, $\bar{k}'$, and $\bar{K}$ of the same length, we shall use $\Gamma \models \bar{k} \to \bar{k}_0 \to \bar{k}' : \bar{K}$ to stand for the sequence of judgements

$$\begin{array}{l} \Gamma \models k_1 \to k_{01} \to k_1' : K_1, \\ \Gamma \models k_2 \to k_{02} \to k_2' : K_2', \\ ..., \\ \Gamma \models k_n \to k_{0n} \to k_n' : K_n', \\ \\ \Gamma \models \delta_i K_i \to K_i' \quad (i = 2, ..., n), \end{array}$$

  where δ_i $(i = 2, ..., n)$ is the substitution $[k_1, ..., k_{i-1}/x_1, ..., x_{i-1}]$.

The rules of TOS for PAL$^+$ are given in figures 8 and 9. For the rules in figure 9 for object let-expressions of the form **let** $v[\Delta] = t{:}T$ **in** k, we distinguish the cases

let-expressions for objects

$$\frac{\Gamma \models T \to T' \quad \Gamma \models t \to t_0 \to t' : T'}{\Gamma \models \mathbf{let}\ v = t{:}T\ \mathbf{in}\ v \to t_0 \to t' : T'}$$

$$\frac{\begin{array}{l}\Gamma, v = t{:}T \models k \to k_1 \to k_2 : K \quad k \not\equiv v \\ \Gamma \models \mathbf{let}\ v = t{:}T\ \mathbf{in}\ K \to K' \\ \Gamma \models [t/v]k \to k_0 \to k' : K'\end{array}}{\Gamma \models \mathbf{let}\ v = t{:}T\ \mathbf{in}\ k \to k_0 \to k' : K'}$$

$$\frac{\begin{array}{l}\Gamma \models \Delta \to \Delta' \quad \Gamma, \Delta \models T \to T' \quad \models \Gamma, v[\Delta] = t{:}T \to \Gamma_1 \quad \Delta \not\equiv \langle\rangle \\ \Gamma, \Delta_{\bar{x}} \models t \to t_0 \to t' : T' \quad t' \not\equiv g[\bar{x}]\ \mathit{such\ that}\ \bar{x} \notin FV(g)\end{array}}{\Gamma \models \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ v \to \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ v \to \mathbf{let}\ v[\Delta'] = t'{:}T'\ \mathbf{in}\ v : (\Delta')T'}$$

$$\frac{\begin{array}{l}\Gamma \models \Delta \to \Delta' \quad \Gamma, \Delta \models T \to T' \quad \models \Gamma, v[\Delta] = t{:}T \to \Gamma_1 \quad \Delta \not\equiv \langle\rangle \\ \Gamma, \Delta_{\bar{x}} \models t \to t_0 \to g[\bar{x}] : T' \quad \Gamma \models g \to g \to g : (\Delta')T'\end{array}}{\Gamma \models \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ v \to \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ v \to g : (\Delta')T'}$$

$$\frac{\begin{array}{l}\Gamma, v[\Delta] = t{:}T \models k \to k_1 \to k_2 : K \quad k \not\equiv v\ \mathit{and}\ \Delta \not\equiv \langle\rangle \\ \Gamma \models \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ K \to K' \\ \Gamma \models [\mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ v/v]k \to k_0 \to k' : K'\end{array}}{\Gamma \models \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ k \to k_0 \to k' : K'}$$

let-expressions for kinds

$$\frac{\Gamma, v[\Delta] = t{:}T \models K \to K_1 \quad \Gamma \models [\mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ v/v]K \to K'}{\Gamma \models \mathbf{let}\ v[\Delta] = t{:}T\ \mathbf{in}\ K \to K'}$$

Fig. 9. The TOS rules for let-expressions in PAL$^+$.

according to whether $k \equiv v$ and whether $\Delta \equiv \langle\rangle$. For example, when $k \equiv v$ and $\Delta \equiv \langle\rangle$, the let-expression $\mathbf{let}\ v = t{:}T\ \mathbf{in}\ v$ computes to the weak-head normal form and normal form of those of t (the first rule in figure 9).

4.2 Basic properties and completeness of the TOS

The TOS defined above has the basic properties concerning sub-derivations and variable occurrences in contexts. Furthermore, it has the properties as given by the following two lemmas, which are proved by induction on derivations of TOS judgements.

Lemma 4.1

- If $\models \Gamma \to \Gamma'$, then Γ and Γ' are contexts of the same length.
- If $\Gamma \models K \to K'$, then Γ is a context and K and K' are kind expressions of the same arity.
- If $\Gamma \models k \to k_0 \to k' : K$, then Γ is a context, K is a kind expression, and k, k_0 and k' are object expressions of the same arity.

Lemma 4.2
The following properties hold for the TOS:

1. (Determinacy)
   - If $\models \Gamma \to \Gamma'$ and $\models \Gamma \to \Gamma''$, then $\Gamma' \equiv \Gamma''$.
   - If $\Gamma \models K \to K'$ and $\Gamma \models K \to K''$, then $K' \equiv K''$.
   - If $\Gamma \models k \to k_1 \to k_2 : K$ and $\Gamma \models k \to k_1' \to k_2' : K'$, then $k_1 \equiv k_1'$, $k_2 \equiv k_2'$, and $K \equiv K'$.
2. (Weakening) If $\Gamma \models J$, $\models \Gamma_1 \to \Gamma_1'$, and Γ_1 contains all entries of Γ, then $\Gamma_1 \models J$, where J is of the form $K \to K'$ or $k \to k_0 \to k' : K$.
3. (Strengthening) Let $z \in Var \cup DV$ and Z be either $z{:}K$ (when $z \in Var$) or $z[\Delta] = t{:}T$ (when $z \in DV$).
   - If $\models \Gamma, Z, \Gamma' \to \Gamma_1$ and $z \notin FV(\Gamma')$, then $\models \Gamma, \Gamma' \to \Gamma_2$ for some Γ_2.
   - If $\Gamma, Z, \Gamma' \models K \to K'$ and $z \notin FV(\Gamma', K)$, then $\Gamma, \Gamma' \models K \to K'$.
   - If $\Gamma, Z, \Gamma' \models k \to k_0 \to k' : K$ and $z \notin FV(\Gamma', k)$, then $\Gamma, \Gamma' \models k \to k_0 \to k' : K$.

By induction on derivations of TOS, we can prove that it is complete with respect to PAL_0^+ – PAL$^+$ without the substitution rules in figures 2 and 5. We use $\vdash_0$ to represent the judgements in PAL_0^+.

Theorem 4.3 (*Completeness*)
- If $\models \Gamma \to \Gamma'$, then Γ **valid** in PAL_0^+.
- If $\Gamma \models K \to K'$, then $\Gamma \vdash_0 K$ **kind** and $\Gamma \vdash_0 K = K'$.
- If $\Gamma \models k \to k_0 \to k' : K$, then $\Gamma \vdash_0 k : K$, $\Gamma \vdash_0 k = k_0 : K$, $\Gamma \vdash_0 k = k' : K$, and $\Gamma \vdash_0 K = K$.

5 Meta-theoretic properties of PAL$^+$

In this section, we first define the notions of reduction, weak-head normal form, and normal form, and then, based on TOS, show that PAL$^+$ has the desirable properties such as Church–Rosser, Subject Reduction, and Strong Normalisation.

5.1 Reduction and weak-head normal and normal forms

Since we take let-expressions, rather than λ-abstractions, as basic expressions, the notion of reduction and the associated notions are new, as defined below.

Definition 5.1 (*reduction*)
The reduction relation is denoted by $\Gamma \vdash M \rhd N$, where Γ is a context and M and N are terms. Reduction is the reflexive and transitive closure of the one-step reduction ($\Gamma \vdash M \rhd_1 N$) defined inductively by the following rules:

- Basic rules (in the first three rules below, it is possible that $\Delta \equiv \langle\rangle$):

$$(v)\Gamma, v[\Delta] = t{:}T, \Gamma' \vdash v \rhd_1 \textbf{let } v[\Delta] = t{:}T \textbf{ in } v$$
$$(\beta)\Gamma \vdash (\textbf{let } v[\Delta_{\bar{x}}] = t{:}T \textbf{ in } v)[\bar{k}] \rhd_1 [\bar{k}/\bar{x}]t \qquad (l(\Delta) = l(\bar{k}))$$
$$(\eta)\Gamma \vdash \textbf{let } v[\Delta_{\bar{x}}] = t[\bar{x}]{:}T \textbf{ in } M \rhd_1 [t/v]M \qquad (\bar{x} \notin FV(t))$$
$$(d)\Gamma \vdash \textbf{let } v[\Delta] = t{:}T \textbf{ in } M \rhd_1 [\textbf{let } v[\Delta] = t{:}T \textbf{ in } v/v]M \qquad (M \not\equiv v,\ \Delta \not\equiv \langle\rangle)$$

- Congruence rules (note that we assume that the expressions involved be terms):

$$\frac{\Gamma \vdash A \rhd_1 B}{\Gamma \vdash El(A) \rhd_1 El(B)} \qquad \frac{\Gamma, \Delta \vdash T \rhd_1 T'}{\Gamma \vdash (\Delta)T \rhd_1 (\Delta)T'}$$

$$\frac{\Gamma, \Delta_{i-1} \vdash K_i \rhd_1 K_i'}{\Gamma \vdash (x_1{:}K_1, ..., x_n{:}K_n)T \rhd_1 (x_1{:}K_1, ..., x_i{:}K_i', ..., x_n{:}K_n)T}$$

$$\frac{\Gamma \vdash f \rhd_1 f'}{\Gamma \vdash f[\bar{k}] \rhd_1 f'[\bar{k}]} \qquad \frac{\Gamma \vdash k_i \rhd_1 k_i'}{\Gamma \vdash f[k_1, ..., k_n] \rhd_1 f'[k_1, ..., k_i', ...k_n]}$$

$$\frac{\Gamma, v[\Delta] = t{:}T \vdash M \rhd_1 M'}{\Gamma \vdash \textbf{let } v[\Delta] = t{:}T \textbf{ in } M \rhd_1 \textbf{let } v[\Delta] = t{:}T \textbf{ in } M'} \quad (M \not\equiv v)$$

$$\frac{\Gamma, \Delta \vdash t \rhd_1 t'}{\Gamma \vdash \textbf{let } v[\Delta] = t{:}T \textbf{ in } M \rhd_1 \textbf{let } v[\Delta] = t'{:}T \textbf{ in } M}$$

$$\frac{\Gamma, \Delta \vdash T \rhd_1 T'}{\Gamma \vdash \textbf{let } v[\Delta] = t{:}T \textbf{ in } M \rhd_1 \textbf{let } v[\Delta] = t{:}T' \textbf{ in } M}$$

$$\frac{\Gamma, \Delta_{i-1} \vdash K_i \rhd_1 K_i'}{\Gamma \vdash \textbf{let } v[\Delta] = t{:}T \textbf{ in } M \rhd_1 \textbf{let } v[\Delta'] = t{:}T \textbf{ in } M}$$

where, in the last rule, $\Delta \equiv x_1{:}K_1, ..., x_n{:}K_n$ and $\Delta' \equiv x_1{:}K_1, ..., x_i{:}K_i', ..., x_n{:}K_n$.

Remark The reduction relation respects the substitution operation.

Lemma 5.2 (*Adequacy for reduction*)

- If $\Gamma \models K \to K'$, then $\Gamma \vdash K \rhd K'$.
- If $\Gamma \models k \to k_0 \to k' : K$, then $\Gamma \vdash k \rhd k_0$ and $\Gamma \vdash k_0 \rhd k'$.

Definition 5.3 (*whnf and nf*)
A term M is in weak-head normal form (whnf) if

- $M \equiv x[k_1, ..., k_n]$ such that $x \in Var(n)$ for some $n \geqslant 0$; or
- $M \equiv \textbf{let } v[\Delta] = t{:}T \textbf{ in } v$ such that $\Delta \not\equiv \langle\rangle$.

A term M is in normal form, notation $M \in NF$, if

- $M \equiv x[k_1, ..., k_n]$ such that $x \in Var(n)$ for some $n \geqslant 0$ and $k_i \in NF$;
- $M \equiv \textbf{let } v[\Delta] = t{:}T \textbf{ in } v$ with $\Delta \equiv x_1{:}K_1, ..., x_n{:}K_n$, such that $\Delta \not\equiv \langle\rangle$, $K_i \in NF$, $t \in NF$, $T \in NF$, and $t \not\equiv g[\bar{x}]$ such that $\bar{x} \notin FV(g)$;
- $M \equiv \textbf{Type}$;
- $M \equiv El(A)$ such that $A \in NF$; or
- $M \equiv (\Delta)T$ with $\Delta \equiv x_1{:}K_1, ..., x_n{:}K_n$, such that $K_i \in NF$ and $T \in NF$.

Lemma 5.4 (*Adequacy of whnf and nf*)

1. For any term M, $M \in NF$ if and only if M has no reductions, i.e., for any context Γ and any term N, $DV(M) \subseteq DV(\Gamma)$ implies that $\Gamma \not\vdash M \rhd_1 N$.

2. If $\Gamma \models K \rightarrow K'$, then K' is in normal form. If $\Gamma \models k \rightarrow k_0 \rightarrow k' : K$, then k_0 is in whnf and k' and K are in normal form.

Proof

(1) By induction on the structure of M and (2) by induction on derivations in TOS. □

5.2 Subject reduction and normalisation

The subject reduction theorem captures both subject reduction and Church–Rosser. Therefore, a notion of parallel reduction is called for.

Definition 5.5 (*parallel reduction*)

The parallel reduction relation, $\Gamma \vdash M \Rightarrow N$, is defined as the least relation satisfying the following rules:

$$\frac{}{\Gamma \vdash x \Rightarrow x} \qquad \frac{v \in DV(\Gamma)}{\Gamma \vdash v \Rightarrow v}$$

$$\frac{\Gamma \vdash \Delta \Rightarrow \Delta' \quad \Gamma, \Delta \vdash t \Rightarrow t' \quad \Gamma, \Delta \vdash T \Rightarrow T'}{\Gamma, v[\Delta] = t{:}T, \Gamma' \vdash v \Rightarrow \textbf{let } v[\Delta'] = t'{:}T' \textbf{ in } v}$$

$$\frac{\Gamma \vdash \bar{k} \Rightarrow \bar{k}' \quad \Gamma, \Delta \vdash t \Rightarrow t' \quad l(\Delta) = l(\bar{k})}{\Gamma \vdash (\textbf{let } v[\Delta_{\bar{x}}] = t{:}T \textbf{ in } v)[\bar{k}] \Rightarrow [\bar{k}'/\bar{x}]t'}$$

$$\frac{\Gamma, \Delta_{\bar{x}} \vdash t \Rightarrow t'[\bar{x}] \quad \Gamma, v[\Delta_{\bar{x}}] = t{:}T \vdash M \Rightarrow M' \quad \bar{x} \notin FV(t')}{\Gamma \vdash \textbf{let } v[\Delta_{\bar{x}}] = t{:}T \textbf{ in } M \Rightarrow [t'/v]M'}$$

$$\frac{\begin{array}{c}\Gamma \vdash \Delta \Rightarrow \Delta' \quad \Gamma, \Delta \vdash t \Rightarrow t' \quad \Gamma, \Delta \vdash T \Rightarrow T' \\ \Gamma, v[\Delta] = t{:}T \vdash M \Rightarrow M' \quad M \not\equiv v \textit{ and } \Delta \not\equiv \langle\rangle\end{array}}{\Gamma \vdash \textbf{let } v[\Delta] = t{:}T \textbf{ in } M \Rightarrow [\textbf{let } v[\Delta'] = t'{:}T' \textbf{ in } v/v]M'}$$

$$\frac{\Gamma \vdash f \Rightarrow f' \quad \Gamma \vdash \bar{k} \Rightarrow \bar{k}'}{\Gamma \vdash f[\bar{k}] \Rightarrow f'[\bar{k}']}$$

$$\frac{\begin{array}{c}\Gamma \vdash \Delta \Rightarrow \Delta' \quad \Gamma, \Delta \vdash t \Rightarrow t' \quad \Gamma, \Delta \vdash T \Rightarrow T' \\ \Gamma, v[\Delta] = t{:}T \vdash M \Rightarrow M'\end{array}}{\Gamma \vdash \textbf{let } v[\Delta] = t{:}T \textbf{ in } M \Rightarrow \textbf{let } v[\Delta'] = t'{:}T' \textbf{ in } M'}$$

We omit the obvious rules for kinds and contexts.

Lemma 5.6

Parallel reduction has the following properties:

1. $\Gamma \vdash M \Rightarrow M$ for any context Γ and term M.
2. If $\Gamma \vdash M \rhd_1 N$, then $\Gamma \vdash M \Rightarrow N$.
3. If $\Gamma \vdash M \Rightarrow N$, then $\Gamma \vdash M \rhd N$.

Remark Parallel reduction also respects the substitution operation.

Lemma 5.7 (*parallel subject reduction*)

- If $\models \Gamma \to \Gamma_0$ and $\vdash \Gamma \Rightarrow \Gamma'$, then $\vdash \Gamma' \Rightarrow \Gamma_0$.
- If $\Gamma \models K \to K_0$, $\vdash \Gamma \Rightarrow \Gamma'$, and $\Gamma \vdash K \Rightarrow K'$, then $\Gamma' \models K' \to K_0$.
- If $\Gamma \models k \to k_0 \to k' : K$, $\vdash \Gamma \Rightarrow \Gamma'$, and $\Gamma \vdash k \Rightarrow k_1$, then for some k_1' and k_{01}, $\Gamma' \models k_1 \to k_1' \to k' : K$, $\Gamma \vdash k_0 \Rightarrow k_{01}$, and $\Gamma \models k_{01} \to k_1' \to k' : K$.

Proof
By induction on derivations in TOS. □

Corollary 5.8 (*subject reduction*)

- If $\Gamma \models K \to K'$ and $\Gamma \models K \rhd_1 K_1$, then $\Gamma \models K_1 \to K'$.
- If $\Gamma \models k \to k_0 \to k' : K$ and $\Gamma \vdash k \rhd_1 k_1$, then $\Gamma \models k_1 \to k_1' \to k' : K$ for some k_1' such that $\Gamma \vdash k_0 \rhd k_1'$.

Proof
By Lemmas 5.7, 5.2 and 5.6. □

Corollary 5.9 (*Church–Rosser*)

- If $\Gamma \models K \to K'$, $\Gamma \vdash K \rhd K_1$, and $\Gamma \vdash K \rhd K_2$, then $\Gamma \vdash K_1 \rhd K'$ and $\Gamma \vdash K_2 \rhd K'$.
- If $\Gamma \models k \to k_0 \to k' : K$, $\Gamma \vdash k \rhd k_1$, and $\Gamma \vdash k \rhd k_2$, then $\Gamma \vdash k_1 \rhd k'$ and $\Gamma \vdash k_2 \rhd k'$.

Proof
By Corollary 5.8 and Lemma 5.2. □

The following shows that TOS only types strongly normalisable terms. For any term M, we say that M is strongly normalisable in context Γ, notation $M \in SN(\Gamma)$, if for any term N, $\Gamma \vdash M \rhd_1 N$ implies that $N \in SN(\Gamma)$.

Lemma 5.10 (*strong normalisation*)

- If $\Gamma \models K \to K'$, then $K \in SN(\Gamma)$.
- If $\Gamma \models k \to k_0 \to k' : K$, then $k \in SN(\Gamma)$.

Proof
By induction on derivations in TOS. We briefly consider two cases. First, consider the first rule for object let-expressions (the first rule in figure 9). By induction hypothesis, T, $t \in SN(\Gamma)$. We show that, if

$$\textbf{let } v = t{:}T \textbf{ in } v \ \rhd_1 \ k,$$

then $k \in SN(\Gamma)$, by considering all possible one-step reductions leading to k. In this case, it must be either (1) **let** $v = t{:}T$ **in** $v \rhd_1 t \equiv k$ by the basic reduction rule (η) or (β), or (2) it is from a congruence rule for reduction because $t \rhd_1 t_1$ or $T \rhd_1 T_1$. For (1), $k \equiv t \in SN(\Gamma)$; for (2), since T and t are both strongly normalisable in Γ, so is any reduct from T or t, and therefore k is strongly normalisable (from the argument of (1)).

Next, we briefly consider a more difficult case, the second instantiation rule (the last rule in figure 8). By induction hypothesis, we have f, $\bar{k}$, $[\bar{k}/\bar{x}]t \ \in SN(\Gamma)$. If $f[\bar{k}] \rhd_1 k$, there are three possible subcases, of which we only consider the case

$k \equiv f_1[\bar{k}]$ such that $\Gamma \vdash f \rhd_1 f_1$. By subject reduction (Corollary 5.8), for some f_1', we have

$$\Gamma \models f_1 \to f_1' \to f' : (\bar{x}{:}\bar{K}')T' \quad \textit{and} \quad \Gamma \vdash \textbf{let } v[\Delta] = t{:}T \textbf{ in } v \rhd f_1'.$$

Then, by case analysis of f_1' being a whnf (by Lemma 5.4), we can show that $f_1[\bar{k}] \in SN(\Gamma)$. $\square$

Remark Note that the above results of Church–Rosser, subject reduction and strong normalisation are for the TOS of PAL$^+$, but not for PAL$^+$ itself. To show that these properties hold for PAL$^+$, we need to show the soundness theorem.

5.3 Soundness of TOS

To prove the soundness of the TOS wrt PAL$^+$, we first prove the following lemma about admissibility of substitution and instantiation. This lemma is proved by induction on the following measure on kinds:

- $|\textbf{Type}| = |El(A)| = 0$.
- $|(x_1{:}K_1, ..., x_n{:}K_n)T| = |K_1| + ... + |K_n| + n$, where $n \geqslant 1$.
- $|\textbf{let } v[\Delta] = t{:}T \textbf{ in } K| = |K|$.

The measure extends to pure contexts as well.

Lemma 5.11

1. Let Z be a context entry. When Z is of the form $z{:}K$, k is an object expression such that $\Gamma \models k \to k_0 \to k' : K'$ and $\Gamma \models K \to K'$. When Z is of the form $z[\Delta] = t{:}T$, $k \equiv \textbf{let } z[\Delta] = t{:}T \textbf{ in } z$. Then we have
   - If $\models \Gamma, Z, \Gamma' \to \Gamma_1$, then $\models \Gamma, [k/z]\Gamma' \to \Gamma_2$ for some Γ_2.
   - If $\Gamma, Z, \Gamma' \models K \to K_1$, then $\Gamma, [k/z]\Gamma' \models [k/z]K \to K_2$ for some K_2.
   - If $\Gamma, Z, \Gamma' \models k_0 \to k_1 \to k_2 : K$, then $\Gamma, [k/z]\Gamma' \models [k/z]k_0 \to k_1' \to k_2' : K'$, $\Gamma, [k/z]\Gamma' \models [k/z]k_1 \to k_1' \to k_2' : K'$, and $\Gamma, [k/z]\Gamma' \models [k/z]K \to K'$ for some k_1', k_2' and K'.
2. If $\Gamma \models f \to f_0 \to f' : (\bar{x}{:}\bar{K})T$ and $\Gamma \models \bar{k} \to \bar{k}_0 \to \bar{k}' : \bar{K}$, then $\Gamma \models f[\bar{k}] \to t_0 \to t' : T'$ and $\Gamma \models [\bar{k}/\bar{x}]T \to T'$ for some t_0, t' and T'.

Theorem 5.12 (*Soundness*)

- If Γ **valid**, then $\models \Gamma \to \Gamma'$ for some Γ'.
- If $\Gamma \vdash K$ **kind**, then $\Gamma \models K \to K'$ for some K'.
- If $\Gamma \vdash K_1 = K_2$, then $\Gamma \models K_1 \to K'$ and $\Gamma \models K_2 \to K'$ for some K'.
- If $\Gamma \vdash k : K$, then $\Gamma \models K \to K'$ and $\Gamma \models k \to k_0 \to k' : K'$ for some K', k_0, and k'.
- If $\Gamma \vdash k_1 = k_2 : K$, then $\Gamma \models K \to K'$, $\Gamma \models k_1 \to k_{10} \to k' : K'$, and $\Gamma \models k_2 \to k_{20} \to k' : K'$, for some K', k', k_{10}, and k_{20}.

Proof
By induction on derivations in PAL$^+$. We consider several cases.

- First, consider the following substitution rule in figure 2:

$$\frac{\Gamma, x{:}K, \Gamma' \vdash k' : K' \quad \Gamma \vdash k : K}{\Gamma, [k/x]\Gamma' \vdash [k/x]k' : [k/x]K'}$$

  By induction hypothesis, $\Gamma, x{:}K, \Gamma' \models k' \rightarrow k'_0 \rightarrow k'' : K''$ and $\Gamma, x{:}K, \Gamma' \models K' \rightarrow K''$, for some K'', k'_0, and k''. By Lemma 5.11(1), we have $\Gamma, [k/x]\Gamma' \models [k/x]k' \rightarrow k'_1 \rightarrow k'_2 : K_1$ and $\Gamma, [k/x]\Gamma' \models [k/x]K' \rightarrow K_1$, for some k'_1, k'_2 and K_1.
- Consider the following instantiation rule in figure 3:

$$\frac{\Gamma \vdash f : (\bar{x}{:}\bar{K})T \quad \Gamma \vdash \bar{k} : \bar{K}}{\Gamma \vdash f[\bar{k}] : [\bar{k}/\bar{x}]T}$$

  By induction hypothesis, determinacy (Lemma 4.2(1)) and inversion, we have $\Gamma \models f \rightarrow f_0 \rightarrow f' : (\bar{x}{:}\bar{K}')T'$ and $\Gamma \models \bar{k} \rightarrow \bar{k}_0 \rightarrow \bar{k}' : \bar{K}'$, with $\Gamma \models \bar{K} \rightarrow \bar{K}'$ and $\Gamma, \bar{x}{:}\bar{K} \models T \rightarrow T'$, for some f_0, f', $\bar{K}'$, T', $\bar{k}_0$ and $\bar{k}'$. By Lemma 5.11(2), we have $\Gamma \models f[\bar{k}] \rightarrow t_0 \rightarrow t' : T'$ and $\Gamma \models [\bar{k}/\bar{x}]T' \rightarrow T''$ for some t_0, t' and T''. Therefore, for this case, we only have to show that $\Gamma \models [\bar{k}/\bar{x}]T \rightarrow T''$. But by Lemma 5.11(1), $\Gamma \models [\bar{k}/\bar{x}]T \rightarrow T'''$ for some T'''. By adequacy (Lemma 5.2) and subject reduction (Corollary 5.8), $\Gamma \models [\bar{k}/\bar{x}]T' \rightarrow T'''$. By determinacy (Lemma 4.2(1)), $T'' \equiv T'''$. So we have $\Gamma \models [\bar{k}/\bar{x}]T \rightarrow T''$.
- Consider the following (let_β) rule in figure 7:

$$\frac{\Gamma, v[\bar{x}{:}\bar{K}] = t{:}T \ \textbf{valid} \quad \Gamma \vdash \bar{k} : \bar{K}}{\Gamma \vdash (\textbf{let}\ v[\bar{x}{:}\bar{K}] = t{:}T\ \textbf{in}\ v)[\bar{k}] = [\bar{k}/\bar{x}]t : [\bar{k}/\bar{x}]T}$$

  By induction hypothesis, determinacy (Lemma 4.2(1)) and properties of sub-derivations, we have $\models \Gamma \rightarrow \Gamma'$, $\Gamma \models \bar{K} \rightarrow \bar{K}'$, $\Gamma, \bar{x}{:}\bar{K} \models T \rightarrow T'$, $\Gamma, \bar{x}{:}\bar{K} \models t \rightarrow t_0 \rightarrow t' : T'$, and $\Gamma \models \bar{k} \rightarrow \bar{k}_0 \rightarrow \bar{k}' : \bar{K}'$, for some $\Gamma', \bar{K}', T', t_0, t', \bar{k}_0$ and $\bar{k}'$. We show that, for some T'', t_1 and t'',

  1. $\Gamma \models [\bar{k}/\bar{x}]T \rightarrow T''$,
  2. $\Gamma \models [\bar{k}/\bar{x}]t \rightarrow t_1 \rightarrow t'' : T''$, and
  3. $\Gamma \models (\textbf{let}\ v[\bar{x}{:}\bar{K}] = t{:}T\ \textbf{in}\ v)[\bar{k}] \rightarrow t_1 \rightarrow t'' : T''$.

  For the first, applying Lemma 5.11(1) suffices to show the existence of T''. For the second, by Lemma 5.11(1), $\Gamma \models [\bar{k}/\bar{x}]t \rightarrow t_1 \rightarrow t'' : T''_1$ and $\Gamma \models [\bar{k}/\bar{x}]T \rightarrow T''_1$ for some t_1, t'' and T''_1. By determinacy (Lemma 4.2(1)), $T''_1 \equiv T''$.
  For the third, we need to consider two cases according to whether $\bar{x}{:}\bar{K} \equiv \langle\rangle$. If $\bar{x}{:}\bar{K} \not\equiv \langle\rangle$, by either the third or the fourth rule in figure 9, for some f',

$$\Gamma \models \textbf{let}\ v[\bar{x}{:}\bar{K}] = t{:}T\ \textbf{in}\ v \rightarrow \textbf{let}\ v[\bar{x}{:}\bar{K}] = t{:}T\ \textbf{in}\ v \rightarrow f' : (\bar{x}{:}\bar{K}')T'.$$

  Therefore, by the second instantiation rule (the last rule in Figure 8), $\Gamma \models (\textbf{let}\ v[\bar{x}{:}\bar{K}] = t{:}T\ \textbf{in}\ v)[\bar{k}] \rightarrow t_1 \rightarrow t'' : T''$.
  If $\bar{x}{:}\bar{K} \equiv \langle\rangle$, we only have to show that $\Gamma \models \textbf{let}\ v = t{:}T\ \textbf{in}\ v \rightarrow t_1 \rightarrow t'' : T''$, and for this, use of the first rule in Figure 9 suffices. □

With soundness and completeness of the TOS for PAL$^+$ and the relationship between reduction and TOS, we can easily show that the system PAL$^+$ has nice meta-theoretic properties. These include admissibility results of the structural rules (e.g. the substitution rules), and computational properties for the reduction relation such as Church–Rosser, Subject Reduction and Strong Normalisation for well-typed terms.

Remark When PAL$^+$ is extended with new constants of the object type theories, we remark that the techniques developed in Goguen (1994) can be used to prove the meta-theoretic results of the object type theories such as UTT.

6 Conclusions

We have presented and studied PAL$^+$, a logical framework based on parameterisation and definitions rather than lambda-abstraction. Further extensions of meta-features such as coercive subtyping (e.g. see Luo, 1999) may be considered.

PAL$^+$ is developed partly as an underlying framework for implementing proof development systems. Most of the proof systems (e.g. those based on type theory like ALF (Magnusson & Nordström, 1994), Coq (Barras *et al.* , 2000), Lego (Luo & Pollack, 1992), NuPRL (Constable *et al.* , 1986) and Plastic (Callaghan & Luo, 2001)), have some form of definition mechanism. Taking definition (and parameterisation) as basic, the proposed lambda-free framework should lead to a better understanding of the underlying theories. We also expect that the simplicity and directness gained would benefit implementation as well as the user (e.g. it is expected that the use of de Bruijn indices would be simplified, and the treatment of meta-variables may be dealt with using the simple method as proposed in Luo (1997) and implemented in Plastic (Callaghan & Luo, 2001)). Paul Callaghan at Durham has implemented a prototype of PAL$^+$, based on his implementation of LF in the system Plastic. We have done some experiments on proof development (e.g., about inductive types and universes) based on the prototype implementation. A better understanding of what we can gain in implementations requires further research and a real development of a proof system.

The development of meta-theory here uses the TOS tool heavily, which shows that TOS is a robust approach that can be adapted to other calculi. Among other related work, Severi and Poll have considered meta-theory of adding definitions into PTS (Severi & Poll, 1994), but they do not consider let-expressions as basic and PTS does not have η-rules either. Another interesting aspect is to consider categorical theories corresponding to PAL$^+$, in a similar way as Cartmell's notion of contextual categories (Cartmell, 1978, 1986) corresponds to Martin-Löf's logical framework.

Another aspect this paper has not discussed is the use of type theories as logical frameworks following the 'judgement-as-types' approach (Harper *et al.*, 1987). We think that the idea to develop lambda-free logical frameworks can similarly be considered and should benefit the users of systems based on the principle of judgement-as-types.

Acknowledgements

This work is based on some notes I wrote in 1997 of the same title, which I did not finish, and a further development of my note in LFM'00 (Luo, 2000). It is partly supported by UK EPSRC grants GR/K79130 and GR/M75518, and the EU TYPES WG grant 29001. I am grateful to Paul Callaghan, Healfdene Goguen, Conor McBride, Bengt Nordström, Randy Pollack, and others, who have read this paper or its earlier versions and given many useful suggestions and comments. Callaghan's prototype implementation of PAL$^+$ has been extremely useful. I am also grateful to the discussions on this topic I have had with members of the Computer-Assisted Reasoning Group at Durham.

References

Barras, B. *et al.* (2000) *The Coq Proof Assistant Reference Manual* (*Version 6.3.1*). INRIA-Rocquencourt.

Callaghan, P. and Luo, Z. (2001) An implementation of LF with coercive subtyping and universes. *J. Automated Reasoning*, **27**(1), 3–27.

Cartmell, J. (1978) *Generalized algebraic theories and contextual category.* PhD thesis, University of Oxford.

Cartmell, J. (1986) Generalized algebraic theories and contextual category. *Ann. Pure Appl. Logic*, **32**.

Constable, R. L. *et al.* (1986) *Implementing Mathematics with the NuPRL Proof Development System.* Prentice-Hall.

de Bruijn, N. G. (1980) A survey of the project AUTOMATH. In: Hindley, J. and Seldin, J., editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press.

Goguen, H. (1994) *A typed operational semantics for type theory.* PhD thesis, University of Edinburgh.

Goguen, H. (1999) Soundness of typed operational semantics for the logical framework. *Typed Lambda Calculi and Applications* (*TLCA'99*).

Harper, R., Honsell, F. and Plotkin, G. (1987) A framework for defining logics. *Proc. 2nd Ann. Symp. on Logic in Computer Science.*

Luo, Z. (1994) *Computation and Reasoning: A Type Theory for Computer Science.* Oxford University Press.

Luo, Z. (1997) *Meta-variables and existential judgements.* Notes.

Luo, Z. (1999). Coercive subtyping. *J. Logic & Computation*, **9**(1), 105–130.

Luo, Z. (2000) PAL$^+$: a lambda-free logical framework. *Proc. Int. Workshop on Logical Frameworks and Meta-languages* (*LFM'2000*), Santa Barbara, CA.

Luo, Z. and Pollack, R. (1992) *LEGO Proof Development System: User's Manual.* LFCS Report ECS-LFCS-92-211, Department of Computer Science, University of Edinburgh.

Magnusson, L. and Nordström, B. (1994) The ALF proof editor and its proof engine. In: Barendregt, H. and Nipkow, T., editors, *Types for Proof and Programs: Lecture Notes in Computer Science 806.* Springer-Verlag.

Nordström, B., Petersson, K. and Smith, J. (1990) *Programming in Martin-Löf's Type Theory: An Introduction.* Oxford University Press.

Severi, P. and Poll, E. (1994) Pure type systems with definitions. *Proc. LFCS'94: Lecture Notes in Computer Science 813.* Springer-Verlag