A Modular Hierarchy of Logical Frameworks

(article starts on next page)

A Modular Hierarchy of Logical Frameworks

Robin Adams

University of Manchester
robin.adams@ma.man.ac.uk

Abstract. We present a modular method for building logical frameworks as a set of *features* that are defined and behave independently of one another. We show how several existing logical frameworks can be so constructed.

1 Introduction

2 Logical Frameworks and Features

A *logical framework* consists of the following components:

1. Disjoint, countably infinite sets of *variables* and *constants*. There must be at least one such set of each, but there may be more than one; for example, we may provide ourselves with sets of n-ary variables for each natural number n.
2. A number of *syntactic classes* of *expressions*, defined in a BNF-style grammar by a set of *constructors*, each of which forms a member of one class from members of other classes. For example, the definition of the class of terms may contain the clause
$$\text{Term } M ::= \cdots \mid [x : A]M \mid \cdots \qquad (1)$$
This indicates that "[:] " is a constructor that takes a variable, a kind and a term, and returns a term. We allow constructors to bind the variables that they use; for example, the above abstraction constructor binds the variable x within the term M. We shall not go into the details of variable binding, but shall take the notions of free and bound variable, α-conversion, and capture-avoiding substitution as read. We shall always identify our expressions up to α-convertibility.
3. Three of the syntactic classes are distinguished as being the classes of *signature declarations*, *context declarations* and *judgement bodies*. Each signature declaration is specified to be either a declaration of a particular constant, or of none. Similarly, each context declaration is specified to be either a declaration of a particular variable or of none. We shall always give these assignments at the same time as writing the grammar clause defining the declaration: we write
$$\delta \text{ of } x$$

to indicate that δ is a context declaration of the variable x, or

$$\delta \text{ of none}$$

to indicate that δ is a context declaration of no variable; similarly for signature declarations.
For example, in a system in which we can declare constants of a particular kind

$$c : A$$

and declare equalities to hold between two terms of the same kind

$$M = N : A \ ,$$

the class of signature declarations is defined by the grammar

$$\text{Signature declaration } \delta ::= c : A \text{ of } c \mid M = M : A \text{ of none}$$

We now define a *signature* to be a finite sequence of signature declarations, such that no two declarations are of the same constant. The *domain* of the signature Σ, $\mathrm{dom}\,\Sigma$, is then defined to be the sequence consisting of the constants declared in Σ, in order. For example, with the class of signature declarations given by the grammar (1), the signature

$$A : \mathbf{Type},\ a : \mathrm{El}(a),\ b : \mathrm{El}(a),\ a = b : \mathrm{El}(a)$$

has domain

$$A, a, b$$

Similarly, we define a *context* to be a finite sequence of context declarations, no two of the same variable, and we define its domain similarly.
Finally, we define a *judgement* to be a string of one of two forms: either

$$\Sigma \text{ sig}$$

or

$$\Gamma \vdash_\Sigma J$$

where Σ is a signature, Γ a context, and J a judgement body.

4. The final component of a logical framework is a set of *rules of deduction* which define the set of *derivable* judgements.

A *feature* shall be a collection of additional elements that can be 'bolted on' to the above set-up.

3 The Basic Framework BF

As is to be expected, **BF** is a very simple system. It allows: the declaration of variable and constant types; the declaration of variables and constants of a previously declared type; and the assertion that a variable or constant has the type with which it was declared, or is itself a type.

The grammar of **BF** is as follows:

$$\begin{array}{rl} \text{Term} & a ::= x \mid c \\ \text{Kind} & A ::= \mathbf{Type} \mid \mathrm{El}(a) \end{array}$$

A *signature* of **BF** is a sequence of *clauses*

$$c : A$$

where c is a constant and A a type.

A *context* of **BF** is a sequence of *declarations*

$$x : A$$

where x is a variable and A a type.

There are four judgement forms in **BF**:

- Σ sig, to say that Σ is a valid signature.
- $\Gamma \vdash_\Sigma$ valid, to say that, under the signature Σ, Γ is a valid context.
- $\Gamma \vdash_\Sigma A$ kind, to say that, under the signature Σ and context Γ, A is a kind.
- $\Gamma \vdash_\Sigma a : A$, to say that, under the signature Σ and context Γ, a is an object of kind A.

The rules of deduction of **BF** are given in Figure 1.

$$\frac{}{\langle\rangle \text{ sig}} \qquad \frac{\vdash_\Sigma A \text{ kind}}{\Sigma, c : A \text{ sig}}\,(c \notin \mathrm{dom}\,\Sigma)$$

$$\frac{\Sigma \text{ sig}}{\vdash_\Sigma \text{ valid}} \qquad \frac{\Gamma \vdash_\Sigma A \text{ kind}}{\Gamma, x : A \vdash_\Sigma \text{ valid}}\,(x \notin \mathrm{dom}\,\Gamma)$$

$$\frac{\Gamma \vdash_\Sigma \text{ valid}}{\Gamma \vdash_\Sigma c : A}\,(c : A \in \Sigma) \qquad \frac{\Gamma \vdash_\Sigma \text{ valid}}{\Gamma \vdash_\Sigma x : A}\,(x : A \in \Gamma)$$

$$\frac{\Gamma \vdash_\Sigma \text{ valid}}{\Gamma \vdash_\Sigma \mathbf{Type} \text{ kind}} \qquad \frac{\Gamma \vdash_\Sigma a : \mathbf{Type}}{\Gamma \vdash_\Sigma \mathrm{El}(a) \text{ kind}}$$

Fig. 1. The basic framework **BF**

4 Features

The first, and most important, of our features are those which allow *parametrization* of variables and constants. Parametrization provides a common core, above which the different forms of abstraction (λ-abstraction with typed or untyped domains, and with β- or $\beta\eta$-conversion, as well as PAL$^+$-style abstraction by let-definition) can be built as conservative extensions.

We define a series of features: **SPar** (1), **SPar** (2), **SPar** (3), ..., and **LPar** (1), **LPar** (2), **LPar** (3), ... These extend one another in the manner shown in Figure 2.

$$\begin{tikzcd}
 & & & \mathbf{LPar}\,(\omega) & \\
 & & \mathbf{LPar}\,(3) \arrow[ur, dotted] & & \mathbf{SPar}\,(\omega) \arrow[ul, hook] \\
 & \mathbf{LPar}\,(2) \arrow[ur, hook] & & \mathbf{SPar}\,(3) \arrow[ul, hook] \arrow[ur, dotted] & \\
\mathbf{LPar}\,(1) \arrow[ur, hook] & & \mathbf{SPar}\,(2) \arrow[ul, hook] \arrow[ur, hook] & & \\
 & \mathbf{SPar}\,(1) \arrow[ul, hook] \arrow[ur, hook] & & & \\
\mathbf{BF} \arrow[ur, hook] & & & &
\end{tikzcd}$$

Fig. 2. The initial fragment of the modular hierarchy

Define the set of *arities* by the grammar

$$\text{Arity } \alpha ::= (\alpha, \ldots, \alpha)$$

We write **0** for the arity (), **n** for the arity

$$\overbrace{(\mathbf{0}, \ldots, \mathbf{0})}^{n}$$

Define the *order* of each arity α, $\mathrm{Ord}\,(\alpha)$, as follows:

$$\begin{aligned}\mathrm{Ord}\,(\mathbf{0}) &= 0\\ \mathrm{Ord}\,((\alpha_1, \ldots, \alpha_n)) &= \max(\mathrm{Ord}\,(\alpha_1)\,, \ldots, \mathrm{Ord}\,(\alpha_n)) + 1\end{aligned}$$

Parameters in Small Types, $\mathbf{SPar}\,(n)$ We give ourselves disjoint, countably infinite sets $\mathcal{V}^{\alpha}$ of α-ary variables for each arity α of order $\leq n$, with $\mathcal{V}^{\mathbf{0}}$ being the set of variables that we have been using in **BF**. Likewise, we introduce sets $\mathcal{C}^{\alpha}$ of α-ary constants, with $\mathcal{C}^{\mathbf{0}}$ being the set of constants of **BF**.

For each arity

$$\alpha \equiv (\alpha_1, \ldots, \alpha_m)$$

of order $\leq n$, we define an α-ary *variable sequence* to be a sequence of distinct variables

$$(x_1, \ldots, x_m)$$

where x_i is an α_i-ary variable.

We define an α-ary *pure context* by recursion on α as follows. An α-ary pure context is a string of the form

$$(x_1 : (\mathit{\Delta}_1)\,\mathrm{El}(a_1), \ldots, x_m : (\mathit{\Delta}_m)\,\mathrm{El}(a_m))$$

where each x_i is an α_i-ary variable, all distinct, $\mathit{\Delta}_i$ an α_i-ary pure context, and a_i a term. Its *domain* is $(x_1, \ldots, x_m)$; note that this is an α-ary variable sequence.

We define an α-ary *abstraction* to be a string of the form

$$[\boldsymbol{x}]M$$

where $\boldsymbol{x}$ is an α-ary variable sequence, and M a term. We take each member of $\boldsymbol{x}$ to be bound within M in this abstraction, and we define free and bound variables and identify all our expressions up to α-conversion in the usual manner. We write $\hat{}M$, $\hat{}N$, ... for arbitrary abstractions. Just as for variables, if $\alpha \equiv (\alpha_1, \ldots, \alpha_m)$, we define an α-ary abstraction sequence to be a sequence $\hat{}\boldsymbol{M} \equiv (\hat{}M_1, \ldots, \hat{}M_m)$, where $\hat{}M_i$ is an α_i-ary abstraction. In this case, we do not insist they be distinct.

We add the following clause to the grammar: For each arity

$$\alpha \equiv (\alpha_1, \ldots, \alpha_m)$$

of order $\leq n$,

$$z(\hat{}\boldsymbol{M})$$

is a term, where z is an α-ary variable or constant, and $\hat{}\boldsymbol{M}$ an α-ary abstraction sequence. This clause subsumes the grammar of **BF**, for $x()$ and $c()$ are terms when x is a $\mathbf{0}$-ary variable and c a $\mathbf{0}$-ary constant.

We also allow declarations of the form

$$c : (\mathit{\Delta})A$$

in the signature, where c is an α-ary constant, Δ an α-ary pure context, and A a type; and those of the form

$$x : (\Delta)\,\mathrm{El}(a)$$

in the context, where x is an α-ary variable, Δ an α-ary pure context, and a a term. Again, these subsume those of **BF**.

We define the operation of *instantiation* as follows. We are going to define a term

$$\{\hat{}M_1/x_1, \ldots, \hat{}M_m/x_m\}M$$

where M is a term, and, for $i = 1, \ldots, m$, $\hat{}M_i \equiv [\boldsymbol{y_i}]M_i$ is an α_i-ary abstraction, and x_i an α_i-ary variable.

$$\{\hat{}\boldsymbol{M}/\boldsymbol{x}\}z(\hat{}\boldsymbol{N}) \equiv z(\{\hat{}\boldsymbol{M}/\boldsymbol{x}\}\hat{}\boldsymbol{N})$$
$$\text{(if } z \text{ is a constant or a variable not in } \boldsymbol{x}\text{)}$$
$$\{\hat{}\boldsymbol{M}/\boldsymbol{x}\}x_i(\hat{}\boldsymbol{N}) \equiv \{\{\hat{}\boldsymbol{M}/\boldsymbol{x}\}\hat{}\boldsymbol{N}/\boldsymbol{y_i}\}M_i$$

If $\hat{}\boldsymbol{M}$ is an α-ary abstraction sequence, and Δ an α-ary pure context, we define a set of judgements

$$\Gamma \Vdash_\Sigma \hat{}\boldsymbol{M} :: \Delta$$

(read: under signature Σ and context Γ, $\hat{}\boldsymbol{M}$ *satisfies* Δ). The definition is as follows. Let

$$\Delta \equiv x_1 : (\Delta_1)\,\mathrm{El}(a_1), \ldots, x_m : (\Delta_m)\,\mathrm{El}(a_m)$$

and let

$$\hat{}M_i \equiv [\boldsymbol{y_i}]M_i$$

By α-conversion, we may assume $\boldsymbol{y_i} \equiv \mathrm{dom}\,\Delta_i$.

$$\Gamma \Vdash_\Sigma \hat{}\boldsymbol{M} :: \Delta$$

is the following set of judgements:

$$\Gamma, \Delta_1 \vdash_\Sigma M_1 : \mathrm{El}(a_1)$$
$$\Gamma, \{\hat{}M_1/x_1\}\Delta_2 \vdash_\Sigma M_2 : \mathrm{El}(\{\hat{}M_1/x_1\}a_2)$$
$$\Gamma, \{\hat{}M_1/x_1, \hat{}M_2/x_2\}\Delta_3 \vdash_\Sigma M_3 : \mathrm{El}(\{\hat{}M_1/x_1, \hat{}M_2/x_2\}a_3)$$
$$\vdots$$
$$\Gamma, \{\hat{}M_1/x_1, \ldots, \hat{}M_{m-1}/x_{m-1}\}\Delta_m \vdash_\Sigma M_m : \mathrm{El}(\{\hat{}M_1/x_1, \ldots, \hat{}M_{m-1}/x_{m-1}\}a_m)$$

In the case $m = 0$, we take $\Gamma \Vdash_\Sigma \hat{}\boldsymbol{M} :: \Delta$ (i.e. $\Gamma \Vdash_\Sigma \langle\rangle :: \langle\rangle$) to be the single judgement

$$\Gamma \vdash_\Sigma \text{valid}$$

The rules of deduction in **SPar** (n) are as follows:

$$\frac{\Delta \vdash_\Sigma \text{valid}}{\Sigma, c : (\Delta)\mathbf{Type}\ \text{sig}}\ (c \notin \mathrm{dom}\,\Sigma) \qquad \frac{\Delta \vdash_\Sigma a : \mathbf{Type}}{\Sigma, c : (\Delta)\,\mathrm{El}(a)\ \text{sig}}\ (c \notin \mathrm{dom}\,\Sigma)$$

$$\frac{\Gamma, \Delta \vdash_\Sigma a : \mathbf{Type}}{\Gamma, x : (\Delta)\, \mathrm{El}(a) \vdash_\Sigma \text{valid}}\ (x \notin \mathrm{dom}\, \Gamma)$$

$$\frac{\Gamma \Vdash_\Sigma \hat{\boldsymbol{M}} :: \Delta}{\Gamma \vdash_\Sigma c[\hat{\boldsymbol{M}}] : \{\hat{\boldsymbol{M}} / \mathrm{dom}\, \Delta\} A}\ (c : (\Delta) A \in \Sigma)$$

$$\frac{\Gamma \Vdash_\Sigma \hat{\boldsymbol{M}} :: \Delta}{\Gamma \vdash_\Sigma x[\hat{\boldsymbol{M}}] : \mathrm{El}(\{\hat{\boldsymbol{M}} / \mathrm{dom}\, \Delta\} a)}\ (x : (\Delta)\, \mathrm{El}(a) \in \Gamma)$$

Finally, $\mathbf{SPar}\,(\omega)$ is defined to be the union of all the features $\mathbf{SPar}\,(n)$.

Parameters in Large Types, $\mathbf{LPar}\,(n)$ The feature $\mathbf{LPar}\,(n)$ allows variables to have types of the form $(\Delta)\mathbf{Type}$ as well as $(\Delta)\,\mathrm{El}(a)$. The details are similar to $\mathbf{SPar}\,(n)$.

For each arity $\alpha \equiv (\alpha_1, \dots, \alpha_m)$ of order $\leq n$:

An *α-ary pure context* is a string of the form

$$(x_1 : (\Delta_1) A_1, \dots, x_m : (\Delta_m) A_m)$$

where each x_i is an α_i-ary variable, all distinct, Δ_i is an $alpha_i$-ary pure context, and A_i is either $\mathbf{Type}$ or $\mathrm{El}(a_i)$ for some term a_i. Its *domain* is defined to be the α_i-ary variable sequence $x_1, \dots, x_m$.

Again, we define an *α-ary abstraction* to be a string of the form $[\boldsymbol{x}]M$, where $\boldsymbol{x}$ is an α-ary variable sequence and M a term. And we add to the grammar the clause: if z is an α-ary variable or constant, and $\hat{\boldsymbol{M}}$ an α-ary abstraction sequence, then $z[\hat{\boldsymbol{M}}]$ is a term.

We allow declarations of the form $c : (\Delta)A$ in the signature and $x : (\Delta)A$ in the context, where c is an α-ary constant, x an α-ary variable, Δ an α-ary pure context, and A either $\mathbf{Type}$ or $\mathrm{El}(a)$ for some term a.

We define instantiation and satisfaction as in $\mathbf{SPar}\,(n)$.

The rules of deduction of $\mathbf{LPar}\,(n)$ are as follows:

$$\frac{\Delta \vdash_\Sigma \text{valid}}{\Sigma, c : (\Delta)\mathbf{Type}\ \text{sig}}\ (c \notin \mathrm{dom}\, \Sigma) \qquad \frac{\Delta \vdash_\Sigma a : \mathbf{Type}}{\Sigma, c : (\Delta)\, \mathrm{El}(a)\ \text{sig}}\ (c \notin \mathrm{dom}\, \Sigma)$$

$$\frac{\Gamma, \Delta \vdash_\Sigma \text{valid}}{\Gamma, x : (\Delta)\mathbf{Type} \vdash_\Sigma \text{valid}}\ (x \notin \mathrm{dom}\, \Gamma) \qquad \frac{\Gamma, \Delta \vdash_\Sigma a : \mathbf{Type}}{\Gamma, x : (\Delta)\, \mathrm{El}(a) \vdash_\Sigma \text{valid}}\ (x \notin \mathrm{dom}\, \Gamma)$$

$$\frac{\Gamma \Vdash_\Sigma \hat{\boldsymbol{M}} :: \Delta}{\Gamma \vdash_\Sigma c[\hat{\boldsymbol{M}}] : \{\hat{\boldsymbol{M}} / \mathrm{dom}\, \Delta\} A}\ (c : (\Delta) A \in \Sigma) \qquad \frac{\Gamma \Vdash_\Sigma \hat{\boldsymbol{M}} :: \Delta}{\Gamma \vdash_\Sigma x[\hat{\boldsymbol{M}}] : \{\hat{\boldsymbol{M}} / \mathrm{dom}\, \Delta\} A}\ (x : (\Delta) A \in \Gamma)$$

Again, $\mathbf{LPar}\,(\omega)$ is the union of all the features $\mathbf{LPar}\,(n)$.

4.1 Lambda Abstraction

We can now add in traditional λ-abstraction. We can make these abstractions typed or untyped (i.e. explicitly include the domain or not), and we can choose to use β or $\beta\eta$-conversion. These two choices lead to four features that can be added to a framework. We shall denote them $\lambda^{\mathrm{t}}_{\beta}$, $\lambda^{\mathrm{ut}}_{\beta}$, $\lambda^{\mathrm{t}}_{\beta\eta}$, $\lambda^{\mathrm{ut}}_{\beta\eta}$. We shall give here the details of $\lambda^{\mathrm{t}}_{\beta}$; the others are very similar.

We shall describe here a feature $\lambda^{\mathrm{t}}_{\beta}$ to be built on top of $\mathbf{BF} + \mathbf{LPar}\,(\omega)$. It would be easy to change the details to give a feature that could be added to $\mathbf{BF} + \mathbf{LPar}\,(n)$, $\mathbf{BF} + \mathbf{SPar}\,(n)$, or $\mathbf{BF} + \mathbf{SPar}\,(\omega)$.

We shall have classes of *α-ary terms* and *α-ary kinds*. We take the terms and kinds of $\mathbf{BF} + \mathbf{LPar}\,(\omega)$ to be the $\mathbf{0}$-ary terms and $\mathbf{0}$-ary kinds, respectively, and introduce new classes for every other arity α.

We add the following clauses to the grammar:

Terms

- Every α-ary variable or constant is an α-ary term.
- If M is an $(\alpha_1, \ldots, \alpha_n)$-ary term, x an α_0-ary variable, and A an α_0-ary type, then
$$[x : (\Delta)A]M$$
is an $(\alpha_0, \alpha_1, \ldots, \alpha_n)$-ary term.
- If M is an $(\alpha_0, \alpha_1, \ldots, \alpha_n)$-ary term, and N an α_0-ary term, then $M[N]$ is an α_0-ary term.

Types

- If x is an α_0-ary variable, A an α_0-ary type, and B an $(\alpha_1, \ldots, \alpha_m)$-ary type, then
$$(x : A)B$$
is an $(\alpha_0, \alpha_1, \ldots, \alpha_m)$-ary type.

Recall that, as the grammar of $\mathbf{BF}$ is still present, the following clauses are still in the grammar:

- $\mathbf{Type}$ is a $\mathbf{0}$-ary type.
- If M is a $\mathbf{0}$-ary term, then $\mathrm{El}(M)$ is a $\mathbf{0}$-ary type.

There are two redundancies in the feature $\lambda^{\mathrm{t}}_{\beta}$. The first: let c be an α-ary constant, where $\alpha \equiv ((\alpha_{11}, \ldots, \alpha_{1k_1}), \ldots, (\alpha_{m1}, \ldots, \alpha_{mk_m}))$. Let

$$c : (x_1 : (\Delta_1)A_1, \ldots, x_m : (\Delta_m)A_m)A$$

be in the signature, where

$$\Delta_i \equiv (x_{i1} : (\Delta_{i1})A_{i1}, \ldots, x_{ik_i} : (\Delta_{ik_i})A_{ik_i})A_i$$

Then we identify the term

$$c[[\boldsymbol{x_1}]M_1, \ldots, [\boldsymbol{x_m}]M_m]$$

with the base term

$$\begin{aligned}c&[[x_{11} : (\Delta_{11})A_{11}] \cdots [x_{1k_1} : (\Delta_{1k_1})A_{1k_1}]M_1] \cdots \\ &[[x_{m1} : (\Delta_{m1})A_{m1}] \cdots [x_{mk_m} : (\Delta_{mk_m})A_{mk_m}]M_m]\end{aligned}$$

Similarly, if x is an $(\alpha_1, \ldots, \alpha_m)$-ary variable, and

$$x : (x_1 : (\Delta_1)A_1) \cdots (x_m : (\Delta_m)A_m)A$$

is in the context, then we identify the term

$$x[[\boldsymbol{x_1}]M_1, \ldots, [\boldsymbol{x_m}]M_m]$$

with the base term

$$\begin{aligned}x&[[x_{11} : (\Delta_{11})A_{11}] \cdots [x_{1k_1} : (\Delta_{1k_1})A_{1k_1}]M_1] \cdots \\ &[[x_{m1} : (\Delta_{m1})A_{m1}] \cdots [x_{mk_m} : (\Delta_{mk_m})A_{mk_m}]M_m]\end{aligned}$$

We introduce new judgement bodies: K kind and $M : K$, where M is an α-ary term and K an α-ary kind.

We define the relations of β-reduction, β-conversion, etc. on our classes of terms in the usual manner, based on the contraction

$$([x : (\Delta)A]M)[N] \rightsquigarrow [N/x]M$$

where $[N/x]M$ denotes the result of substituting the α-ary term N for the α-ary variable x throughout the term M, relabelling bound variables to avoid capture.

The rules of deduction in $\lambda^{\mathsf{t}}_{\beta}$ are now:

$$\frac{\Gamma, x : A \vdash_\Sigma B \text{ kind}}{\Gamma \vdash_\Sigma (x : A)B \text{ kind}}$$

$$\frac{\vdash_\Sigma A \text{ kind}}{\Sigma, c : A \text{ sig}}\,(c \notin \operatorname{dom} \Sigma) \qquad \frac{\Gamma \vdash_\Sigma A \text{ kind}}{\Gamma, x : A \vdash_\Sigma \text{valid}}\,(x \notin \operatorname{dom} \Gamma)$$

$$\frac{\Gamma, x : A \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma [x : A]M : (x : A)B} \qquad \frac{\Gamma \vdash_\Sigma M : (x : A)B \quad \Gamma \vdash_\Sigma N : A}{\Gamma \vdash_\Sigma M[N] : [N/x]B}$$

$$\frac{\Gamma \vdash_\Sigma M : A \quad \Gamma \vdash_\Sigma B \text{ kind}}{\Gamma \vdash_\Sigma M : B}\,(A =_\beta B)$$

Global Definition of Constants, **cdef** Depends on **SPar** (ω).

$$\text{Signature Declaration } \gamma ::= \cdots \mid c^{\alpha}[\Delta^{\alpha}] := M : A$$

If $c[\Delta] := M : A$ is in the signature, the following is a reduction rule:

$$c[\hat{\boldsymbol{N}}] \rightsquigarrow_{\delta_c} \{\hat{\boldsymbol{N}}/\operatorname{dom}\Delta\}$$

$$\frac{\Delta \vdash_{\Sigma} M : A}{\Sigma, c[\Delta] := M : A \text{ sig}}\,(c \notin \operatorname{dom}\Sigma) \qquad \frac{\Gamma \Vdash_{\Sigma} \hat{\boldsymbol{N}} :: \Delta}{\Gamma \vdash_{\Sigma} c[\hat{\boldsymbol{N}}] : \{\hat{\boldsymbol{N}}/\operatorname{dom}\Delta\}A}\,(c[\Delta] := M : A \in \Sigma)$$

$$\frac{\Gamma \vdash_{\Sigma} M : A \quad \Gamma \vdash_{\Sigma} B \text{ kind}}{\Gamma \vdash_{\Sigma} M : B}\,(\Gamma \vdash_{\Sigma} A =_{\delta_c} B)$$

Global Definition of Variables, **vdef** Depends on **SPar** (ω).

$$\text{Context Declaration } \delta ::= \cdots \mid x^{\alpha}[\Delta^{\alpha}] := M : A$$

If $x^{\alpha}[\Delta^{\alpha}] := M^{\beta} : A^{\beta}$ is in the context, the following is a reduction rule:

$$x[\hat{\boldsymbol{N}}] \rightsquigarrow_{\delta_v} \{\hat{\boldsymbol{N}}/\operatorname{dom}\Delta\}M$$

$$\frac{\Gamma, \Delta \vdash_{\Sigma} M : A}{\Gamma, x[\Delta] := M : A \vdash_{\Sigma} \text{valid}}\,(x \notin \operatorname{dom}\Gamma) \qquad \frac{\Gamma \Vdash_{\Sigma} \hat{\boldsymbol{N}} :: \Delta}{\Gamma \vdash_{\Sigma} x[\hat{\boldsymbol{N}}] : \{\hat{\boldsymbol{N}}/\operatorname{dom}\Delta\}A}\,(x[\Delta] := M : A \in \Gamma)$$

$$\frac{\Gamma \vdash_{\Sigma} M : A \quad \Gamma \vdash_{\Sigma} B \text{ kind}}{\Gamma \vdash_{\Sigma} M : B}\,(\Gamma \vdash_{\Sigma} a =_{\delta} b)$$

Fig. 3. Miscellaneous features

Local Definitions, **let** Depends on **vdef**.

$$\begin{array}{ll} \text{Term} & M ::= \cdots \mid \text{let}\, x^\alpha[\Delta^\alpha] := M : A \,\text{in}\, M \\ \text{Kind} & A ::= \cdots \mid \text{let}\, x^\alpha[\Delta^\alpha] := M : A \,\text{in}\, A \end{array}$$

$$\text{let}\, v[\Delta] = M : A \,\text{in}\, N \rightsquigarrow \{[\text{dom}\,\Delta]M/v\}N$$
$$\text{let}\, v[\Delta] = M : A \,\text{in}\, K \rightsquigarrow \{[\text{dom}\,\Delta]M/v\}K$$

$$\frac{\Gamma, v[\Delta] = M : A \vdash_\Sigma K \text{ kind}}{\Gamma \vdash_\Sigma \text{let}\, v[\Delta] = M : A \,\text{in}\, K \text{ kind}} \qquad \frac{\Gamma, v[\Delta] = M : A \vdash_\Sigma N : K}{\Gamma \vdash_\Sigma \text{let}\, v[\Delta] = M : A \,\text{in}\, N : \text{let}\, v[\Delta] = M : A \,\text{in}\, K}$$

$$\frac{\Gamma \vdash_\Sigma M : A \quad \Gamma \vdash_\Sigma B \text{ kind}}{\Gamma \vdash_\Sigma M : B}\,(A =_\delta B)$$

Judgemental Equality, **eq** Depends on **SPar** (ω).

$$\text{Judgement body } J ::= \cdots \mid M = M : A \mid A = A$$

$$\frac{\Gamma \vdash_\Sigma M : A \quad \Gamma \vdash_\Sigma N : A}{\Gamma \vdash_\Sigma M = N : A}\,(M = N) \qquad \frac{\Gamma \vdash_\Sigma A \text{ kind} \quad \Gamma \vdash_\Sigma B \text{ kind}}{\Gamma \vdash_\Sigma A = B}$$

Fig. 4. Miscellaneous features

$$\frac{}{\langle\rangle \text{ sig}} \qquad \frac{\vdash_\Sigma A \text{ kind}}{\Sigma, c : A \text{ sig}}\,(c \notin \text{dom}\,\Sigma)$$

$$\frac{\Sigma \text{ sig}}{\vdash_\Sigma \text{ valid}} \qquad \frac{\Gamma \vdash_\Sigma A \text{ kind}}{\Gamma, x : A \vdash_\Sigma \text{ valid}}\,(x \notin \text{dom}\,\Gamma)$$

$$\frac{\Gamma \vdash_\Sigma \text{ valid}}{\Gamma \vdash_\Sigma c : A}\,(c : A \in \Sigma) \qquad \frac{\Gamma \vdash_\Sigma \text{ valid}}{\Gamma \vdash_\Sigma x : A}\,(x : A \in \Gamma)$$

$$\frac{\Gamma \vdash_\Sigma \text{ valid}}{\Gamma \vdash_\Sigma \textbf{Type} \text{ kind}} \qquad \frac{\Gamma \vdash_\Sigma M : \textbf{Type}}{\Gamma \vdash_\Sigma \text{El}(M) \text{ kind}} \qquad \frac{\Gamma \vdash_\Sigma M = N : \textbf{Type}}{\Gamma \vdash_\Sigma \text{El}(M) = \text{El}(N)}$$

$$\frac{\Gamma \vdash_\Sigma M : A}{\Gamma \vdash_\Sigma M = M : A} \qquad \frac{\Gamma \vdash_\Sigma M = N : A}{\Gamma \vdash_\Sigma N = M : A} \qquad \frac{\Gamma \vdash_\Sigma M = N : A \quad \Gamma \vdash_\Sigma N = P : A}{\Gamma \vdash_\Sigma M = P : A}$$

$$\frac{\Gamma \vdash_\Sigma A \text{ kind}}{\Gamma \vdash_\Sigma A = A} \qquad \frac{\Gamma \vdash_\Sigma A = B}{\Gamma \vdash_\Sigma B = A} \qquad \frac{\Gamma \vdash_\Sigma A = B \quad \Gamma \vdash_\Sigma B = C}{\Gamma \vdash_\Sigma A = C}$$

$$\frac{\Gamma \vdash_\Sigma M : A \quad \Gamma \vdash_\Sigma A = B}{\Gamma \vdash_\Sigma M : B} \qquad \frac{\Gamma \vdash_\Sigma M = N : A \quad \Gamma \vdash_\Sigma A = B}{\Gamma \vdash_\Sigma M = N : B}$$

Fig. 5. Basic framework with judgemental equality, **BF**′

4.2 Other Features

We present a summary of other features in Figures 3 and 4. Each of these features depends on $\mathbf{SPar}\,(\omega)$; it would be easy enough to write a version dependent on $\mathbf{SPar}\,(n)$ for some finite n.

We could alternatively have built judgemental equality into the hierarchy from the beginning. Define an alternative version of $\mathbf{BF}$, $\mathbf{BF}'$, with the rules of deduction in Figure 5

We can now define alternative versions of all the features we have presented so far. For example, we can define the feature $\mathbf{cdef}'$, an alternative version of $\mathbf{cdef}$, to consist of the rules of deduction

$$\frac{\Delta \vdash_{\Sigma} M : A}{\Sigma, c[\Delta] = M : A \text{ sig}}$$

$$\frac{\Gamma \Vdash_{\Sigma} \hat{\boldsymbol{N}} :: \Delta}{\Gamma \vdash_{\Sigma} c[\hat{\boldsymbol{N}}] : \{\hat{\boldsymbol{N}} / \operatorname{dom} \Delta\} A} \, (c[\Delta] = M : A \in \Sigma) \qquad \frac{\Gamma \Vdash_{\Sigma} \hat{\boldsymbol{N}} = \hat{\boldsymbol{P}} :: \Delta}{\Gamma \vdash_{\Sigma} c[\hat{\boldsymbol{N}}] = c[\hat{\boldsymbol{P}}] : \{\hat{\boldsymbol{N}} / \operatorname{dom} \Delta\} A} \, (c[\Delta] = M : A \in$$

$$\frac{\Gamma \Vdash_{\Sigma} \hat{\boldsymbol{N}} :: \Delta}{\Gamma \vdash_{\Sigma} c[\hat{\boldsymbol{N}}] = \{\hat{\boldsymbol{N}} / \operatorname{dom} \Delta\} M : \{\hat{\boldsymbol{N}} / \operatorname{dom} \Delta\} A} \, (c[\Delta] = M : A \in \Sigma)$$

Conjecture 1. *Let $\{F_1, \ldots, F_n\}$ be a set of features closed under dependency. Then the derivable judgements of*

$$\mathbf{BF} + F_1 + \cdots + F_n + \mathbf{eq}$$

are the same as those of

$$\mathbf{BF}' + F_1' + \cdots + F_n'$$

This conjecture is provable for $\{F_1, \ldots, F_n\} \subseteq \{\mathbf{SPar}\,(n) \mid n \in \mathbb{N}\} \cup \{\mathbf{LPar}\,(n) \mid n \in \mathbb{N}\}$, for, then, in each system, $\Gamma \vdash_{\Sigma} M = N : A$ is derivable iff $\Gamma \vdash_{\Sigma} M : A$ is derivable and $M \equiv N$; and $\Gamma \vdash_{\Sigma} A = B$ is derivable iff $\Gamma \vdash_{\Sigma} A$ kind is derivable and $A \equiv B$.

Miscellaneous Features

5 Existing Logical Frameworks

$$\begin{aligned}
\text{PAL} &= \mathbf{BF} + \mathbf{LPar}\,(1) + \mathbf{cdef} \\
\text{AUT-68} &\simeq \mathbf{BF} + \mathbf{SPar}\,(\omega) + \lambda^{\mathrm{t}}_{\beta} + \mathbf{LPar}\,(1) + \mathbf{cdef} \\
\text{AUT-QE} &= \mathbf{BF} + \mathbf{LPar}\,(\omega) + \lambda^{\mathrm{t}}_{\beta} + \mathbf{cdef} \\
\text{ELF} &= \mathbf{BF} + \mathbf{SPar}\,(\omega) + \lambda^{\mathrm{t}}_{\beta} \\
\text{Martin-Löf's Theory of Types} &= \mathbf{BF} + \mathbf{LPar}\,(\omega) + \lambda^{\mathrm{ut}}_{\beta\eta} + \mathbf{eq} \\
\text{LF} &= \mathbf{BF} + \mathbf{LPar}\,(\omega) + \lambda^{\mathrm{t}}_{\beta\eta} + \mathbf{eq}
\end{aligned}$$

To build PAL$^+$ in the framework, there are two possibilities. Firstly, we could write a feature that introduces classes of α-ary terms and kinds for every arity α, in a similar manner to $\lambda^{\mathrm{t}}_{\beta}$, but the only such terms are the α-ary variables and constants. Then we could build on top of this a features similar to **vdef** and **let**, but allowing global and local definitions of any arity term and kind. Putting these three features on top of $\mathbf{BF} + \mathbf{LPar}\,(\omega) + \mathbf{eq}$ yields a framework equivalent to PAL$^+$.

Alternatively, we could build features similar to **vdef** and **let** on top of $\mathbf{BF} + \mathbf{LPar}\,(\omega) + \mathbf{eq} + \lambda^{\mathrm{t}}_{\beta\eta}$, including a redundancy that identifies $[x_1 : A_1] \cdots [x_n ; A_n] M$ with let $v[x_1 : A_1, \ldots, x_n : A_n] = M : A inv$, where A is an inferred kind for M.

6 Use of Frameworks

We show how to build an arbitrary first-order theory in $\mathbf{BF} + \mathbf{SPar}\,(2)$.

The signature consists of:

- term : **Type**
- For each n-ary function symbol F in the language, the declaration

$$F : (x_1 : \mathrm{El(term)}, \ldots, x_n : \mathrm{El(term)})\,\mathrm{El(term)}$$

- prop : **Type**
- For each n-ary predicate symbol P in the language, the declaration

$$P : (x_1 : \mathrm{El(term)}, \ldots, x_n : \mathrm{El(term)})\,\mathrm{El(prop)}$$

- $\rightarrow : (x : \mathrm{El(prop)}, y : \mathrm{El(prop)})\,\mathrm{El(prop)}$
- $\forall : (p : (x : \mathrm{El(term)})\,\mathrm{El(prop)})\,\mathrm{El(prop)}$
- $\mathrm{Prf} : (x : \mathrm{El(prop)})\mathbf{Type}$
- $\rightarrow I : (p, q : \mathrm{El(prop)}, H : (x : \mathrm{El(Prf}[p]))\,\mathrm{El(Prf}[q]))\,\mathrm{El(Prf}[\rightarrow [p, q]])$
- $\rightarrow E : (p, q : \mathrm{El(prop)}, H_1 : \mathrm{El(Prf}[\rightarrow [p, q]]), H_2 : \mathrm{El(Prf}[p]))\,\mathrm{El(Prf}[q])$
- $\forall I : (p : (x : \mathrm{El(term)})\,\mathrm{El(prop)}, H : (x : \mathrm{El(term)})\,\mathrm{El(Prf}[p[x]]))\,\mathrm{El(Prf}[\forall[p]])$
- $\forall E : (p : (x : \mathrm{El(term)})\,\mathrm{El(prop)}, t : \mathrm{El(term)}, H : \mathrm{El(Prf}[\forall[p]]))\,\mathrm{El(Prf}[p[t]])$

Theorem 2. *1. There is a bijection ρ between the terms with free variables among $x_1, \ldots, x_n$ in the first-order language, and the terms M such that*

$$x_1 : \mathrm{El(term)}, \ldots, x_n : \mathrm{El(term)} \vdash_{\Sigma} M : \mathrm{El(term)}$$

2. There is a bijection σ between the formulas with free variables among $x_1, \ldots, x_n$ in the first-order language, and the terms M such that

$$x_1 : \mathrm{El(term)}, \ldots, x_n : \mathrm{El(term)} \vdash_{\Sigma} M : \mathrm{El(prop)}$$

3. Let ϕ, ψ_1, …, ψ_m be formulas with free variables among $x_1, \ldots, x_n$. Then ϕ is provable from hypothese ψ_1, …, ψ_m iff there is a term M such that

$$x_1 : \mathrm{El(term)}, \ldots, x_n : \mathrm{El(term)}, y_1 : \mathrm{El(Prf}[\sigma(\psi_1)]), \ldots, y_m : \mathrm{El(Prf}[\sigma(\psi_m)]) \vdash_{\Sigma} M : \mathrm{El(Prf}[\sigma(\phi)])$$

Notice that the correspondance between the entities of the object logic and the terms of the logical framework is a bijection up to *identity* (that is, α-conversion), not up to convertibility; indeed, in a framework whose only features are **SPar** (n) and **LPar** (n), there is no such thing as convertibility. This theorem is much easier to prove than most adequacy theorems, because the correspondence between the framework and the object logic is so much closer than in a traditional logical framework.

We build W-types within $\mathbf{BF} + \mathbf{LPar}\,(2) + \mathbf{eq}$. We suppress instances of El, and use η-contractions; e.g. we write $W[A, B]$ for $W[A, [x : A]B[x]]$.

$$\begin{aligned}
W &: (A : \mathbf{Type}, B : (A)\mathbf{Type})\mathbf{Type}, \\
\mathrm{sup} &: (A : \mathbf{Type}, B : (A)\mathbf{Type}, a : A, b : (B[a])W[A, B])W[A, B] \\
E_W &: (A : \mathbf{Type}, B : (A)\mathbf{Type}, C : (W[A, B])\mathbf{Type}, \\
&\quad f : (x : A, y : (B[x])W[A, B], g : (v : B[x])C[y[v]])C[\mathrm{sup}[A, B, x, y]], z : W[A, B])C[z], \\
&(A : \mathbf{Type}, B : (A)\mathbf{Type}, C : (W[A, B])\mathbf{Type}, \\
&f : (x : A, y : (B[x])W[A, B], g : (v : B[x])C[y[v]])C[\mathrm{sup}[A, B, x, y]], a : A, b : (B[a])W[A, B]) \\
&\quad E_W[A, B, C, f, g, \mathrm{sup}[A, B, a, b]] = f[a, b, [v : B[x]]E_W[A, B, C, f, g, y[v]]] : C[\mathrm{sup}[A, B, a, b]]
\end{aligned}$$