Arity Polymorphism and Dependent Types

Eugenio Moggi (DISI, Univ. of Genova, Italy)
http://www.disi.unige.it/person/MoggiE/

Arity Polymorphism some motivations:
- FML [JBM98@JFP], FLC [Jay00];
- Zip calculus [Tul00@MPC]

Dependent Types and Stratification:
- TS settings [Jac91@PhD,Jac99@book];
- DML [Xi98@PhD,XP99@POPL]

Limited form of Inductive Types:
- Pattern matching suffices, Prime Covering [Coq92@TYPES]
- Extensionality & Decidable equality of types coexist

Motivations for Arity Polymorphism

- the type of map one would like to have in FML

$$\forall n\colon Arity.\forall F\colon Functor(n).\forall X,Y\colon Type^n.(X_\iota \rightarrow Y_\iota \mid \iota\colon n) \rightarrow F(X) \rightarrow F(Y)$$

- the type of zip (for lists) in the Zip calculus

$$\forall n\colon Arity.\forall X\colon Type^n.(list\ X_\iota \mid \iota\colon n) \rightarrow list\ (X_\iota \mid \iota\colon n)$$

- example of Arity Polymorphism in Pascal

```
MatrixMultiply ( const m,n,p: nat;
                 A:array [m,n] of real; B:array [n,p] of real;
                 var C:array [m,p] of real) {C:=A*B}
```

types do not depend on indexes ι

Some Remarks

Arities $n\colon Arity$ are sets, like types $X\colon Type$.

Arities for zip must be finite, but arities for map can be arbitrary.

Types, such as X_ι, may depend on indexes $\iota\colon n$.

$Type$ must be closed under quant. $(\tau_i \mid \iota\colon n)$ over n
and $\forall n\colon Arity.\tau$ over $Arity$.

$Kind$ must be close under quant. over n, e.g. $Type^n$.

Questions

What closure properties should the *universe* $Arity$ have?

Types have NO computational content, i.e. are irrelevant for the dynamic semantics! What is the computational content of arities and indexes?

What are the differences between the universes $Arity$ and $Type$?

What is equality of types (arities and indexes) in the presence of a fix-point combinator $Y : \forall X : Type. (X \to X) \to X$?

Desiderata

- Decidable typing. Expresiveness.

* Extensional equality (a semanticist's desiderata).

* Limited type annotation (a programmer's desiderata).

Type Systems - TS [Jac91@PhD,Jac99@book], [Bar92@HB,Geu93@PhD]

$s \in \mathsf{S}$ sort, $c, C \in \mathsf{C}$ constant, $\Gamma ::= \emptyset \mid \Gamma, x\colon A\colon s$ context

$$A, B, M, N \;\; ::= \;\; x \mid s \mid \forall x\colon A.B \mid c(\overline{M}) \mid \lambda x\colon A.M \mid M\,N$$

Well-formed context $\Gamma \vdash$, set $\Gamma \vdash A\colon s$, element $\Gamma \vdash M\colon A\colon s$.

TS-settings: phase-distinction, type-dependency

- $s_1 < s_2$: a s_2-element M may depend on a s_1-variable x.
$<$ preorder.

- $s_1 \prec s_2$: a s_2-set B may depend on a s_1-variable x
$\prec \subseteq <$ and downward closed, i.e. $s < s_1 \prec s_2$ implies $s \prec s_2$.

$$\Gamma, x\colon A\colon s_1 \vdash M\colon B\colon s_2 \qquad \frac{\Gamma \vdash M\colon A\colon s}{\Gamma_{<s} \vdash M\colon A\colon s} \qquad \frac{\Gamma \vdash A\colon s}{\Gamma_{\prec s} \vdash M\colon A\colon s}$$

TS-features: axioms, $\forall$-closure, constants

- axiom $s_2 \in s_1$, i.e. s_2 is an s_1-set.
$s_2 \in s_1$ implies $s_1 \prec s_2$, and $s \prec s_2 \in s_1$ implies $s < s_1$.

We consider only functional TS, where well-formed sets and elements have exactly one sort.

- $\forall(s_1, s_2)$, i.e. s_2-sets are closed under quantification over a s_1-set.

$$\frac{\Gamma, x{:}\, A{:}\, s_1 \vdash M{:}\, B{:}\, s_2}{\Gamma \vdash \lambda x{:}\, A.M{:}\, (\forall x{:}\, A.B){:}\, s_2} \qquad \frac{\Gamma \vdash M{:}\, (\forall x{:}\, A.B){:}\, s_2 \quad \Gamma \vdash N{:}\, A{:}\, s_1}{\Gamma \vdash M\, N{:}\, B[x{:}= N]{:}\, s_2}$$

$\forall \subseteq <$. Write $A \to B$ for $\forall x{:}\, A.B$, when $x \notin \mathsf{FV}(B)$.

- $C{:}\, (\Gamma)s$ family of s-sets. $C{:}\, (x_\iota{:}\, A_\iota{:}\, s_\iota \mid \iota{:}\, m)s$ implies $s_\iota \prec s$.
- $c{:}\, (\Gamma)A{:}\, s$ family of s-elements. $c{:}\, (x_\iota{:}\, A_\iota{:}\, s_\iota \mid \iota{:}\, m)A{:}\, s$ implies $s_\iota < s$.

* Inductive types. Existential types.

TS settings

$\lambda \rightarrow$	$* = Type$
$\lambda \prod$	$* = Type \succ Type$
$ML, \;\; F$	$* = Type \succ Kind$ $\square = Kind$
DML	$* = Type \succ Kind, Index$ $\square = Kind > Index$ $Index \succ Index$
Zip	$* = Type \succ Kind, Arity$ $\mathsf{D} = Arity \succ Kind$ $\square = Kind \succ Kind > Arity$

$$\boxed{\begin{array}{l} * = Type \succ Kind, Arity \\ \square = Kind \succ Kind', Kind > Arity \\ Arity \succ Kind' \\ Kind' \end{array}}$$

TS for system F and DML

	$Type$	$Kind$
$Type$	$<, \forall$	
$Kind$	$\prec, \ni, \forall$	$<, \forall$

$$\tau ::= \delta \mid X \mid \tau_1 \to \tau_2 \mid \forall X : Type . \tau$$

	$Type$	$Kind$	$Index$
$Type$	$<, \forall$		
$Kind$	$\prec, \ni, \forall$	$<$	
$Index$	$\prec, \forall$	$<$	$\prec$

$$\begin{aligned} \tau &::= \delta \mid X \mid \tau_1 \to \tau_2 \mid \forall X : Type . \tau \\ & \quad \delta(\bar{\iota}) \mid \prod x : n . \tau \\ \iota &::= x \mid f(\bar{\iota}) \\ n &::= b \mid 0 \mid 1 \mid p(\bar{\iota}) \end{aligned}$$

TS for FML and revised Zip calculus

	$Type$	$Kind$	$Arity$	$Kind'$
$Type$	$<, \forall$			
$Kind$	$\prec, \ni, \forall$	$<, \forall$		
$Arity$	$\prec, \forall$	$<, \forall$	$<$	
$Kind'$	$\prec, \forall$	$\prec, \forall$	$\prec, \ni$	$<$

in [Tul00@MPC]

$Type = *$

$Kind = \Box$

$Arity = \mathsf{D}$

$Kind' = \Box$

DML is more structured: it distinguishes between index types and index propositions. But the dependencies are correctly represented.

DML is parameterized over a domain of constraints. The appropriate one for comparison with arity polymorphism is that of linear inequalities over integer / natural numbers.

In DML indexes have NO computational content.
there is an index-erasing function $||\cdot||$ s.t.

$$||\delta(\bar{\iota})|| = \delta \quad ||\prod x{:}\, n.\tau|| = ||\tau|| \quad ||e\,\iota|| = ||\lambda x{:}\, n.e|| = ||e||$$

Prop. $\Gamma_\iota; \Gamma_e \vdash e{:}\, \tau$ in DML implies $||\Gamma_e|| \vdash ||e||{:}\, ||\tau||$ in ML

Stratified Expressions: s-sets and s-elements

$Kind'$ $Arity$ $\Gamma_n ::= \emptyset \mid \Gamma_n, x\colon Arity$

$n ::= x \mid 0 \mid s\, n$

0 is the empty set, and $s\,n$ is the disjoint union $1 + n$.

$Arity$ n $\Gamma_\iota ::= \emptyset \mid \Gamma_\iota, x\colon n$

$\iota ::= x \mid 0' \mid s'\,\iota \mid l\iota\,\iota$ where $l\iota ::= (\iota | x\colon n) \mid Nil_n \mid \iota, l\iota$

$Kind$ $K ::= Type \mid K^n$ $\Gamma_u ::= \emptyset \mid \Gamma_u, x\colon K$

$\tau, u ::= x \mid lu \mid u\,\iota \mid \tau_1 \to \tau_2 \mid \forall x\colon K.\tau \mid \prod(lu) \mid \forall x\colon Arity.\tau$

where $lu ::= (u\colon K | x\colon n) \mid Nil_K \mid u, lu$

$Type$ τ $\Gamma_e ::= \emptyset \mid \Gamma_e, x\colon \tau$

$e ::= x \mid \mu x\colon \tau.e \mid \lambda x\colon \tau.e \mid e_1\, e_2 \mid \lambda x\colon K.e \mid e\,u \mid le \mid e\,\iota \mid Le \mid e\,n$

where $le ::= (e | x\colon n) \mid nil \mid e, le$ and $Le ::= (e | x\colon Arity) \mid e, Le$

Stratification, Computational content

Stratification allows to define well-formedness, normal-forms and equality in stages (as in $F\omega$), rather than by mutual recursion. As in $F\omega$, equality $e_1 = e_2$ is not used for type-checking $e\colon\tau$.

Unlike DML, arities n and indexes ι have computational content, since le and Le can perform case analysis.

Inductive types

0 and $s\,n$ are simple forms of inductive types. For them pattern matching (see [Coq92@TYPES]) is more convenient and as expressive as the corresponding elimination rules. This is because these inductive types have a prime covering, which is "the most refined" among the coverings.

Prime covering still exist when arities are $n ::= x \mid 0 \mid 1 \mid n_1 + n_2 \mid n_1 \times n_2$.

For FLC [Jay00] the arities are $n ::= x \mid 0 \mid 1 \mid n_1 + n_2$.

$Arity$ is not an inductive type (i.e. the type of natural numbers) for the sorts $Kind'$, $Arity$ and $Kind$, i.e. for these sorts we cannot define elements by induction on the natural numbers. This choice ensures existence of prime coverings and extensionality, but it limits expressiveness, e.g. the following functions are not definable

- $+\colon Arity \to Arity \to Arity$ s.t. $0 + y = y \quad (s\,x) + y = s\,(x + y)$
- $f\colon Arity \to Type \to Type$ s.t. $f(0, X) = X \quad f(s\,x, X) = X \to f(x, X)$
  rank 1 types over X
- $k\colon Arity \to Type \to Type$ s.t. $k(0, X) = X \quad k(s\,x, X) = k(x, X) \to X$
  rank increasing types over X

$Arity$ is an inductive type for the sort $Type$. Recursive definitions $\mu x.e$ and pattern matching suffice to recover induction.

CBV operational semantics

$n_v ::= 0 \mid s\, n_v$ arity values

$\iota_v ::= 0' \mid s'\, \iota_v$ index values

$e ::= x \mid \mu x.e \mid \lambda x.e \mid e_1\, e_2 \mid le \mid e\, \iota \mid Le \mid e\, n$ expressions, where
$le ::= (e|x{:}\, n) \mid nil \mid e, le$ and $Le ::= (e|x) \mid e, Le$

$v ::= \lambda x.e \mid lv \mid Le$ values, where $lv ::= nil \mid v, lv$

$$\frac{}{nil \Rightarrow nil} \qquad \frac{e \Rightarrow v \quad le \Rightarrow lv}{e, le \Rightarrow v, lv} \qquad \frac{n \Rightarrow 0}{(e|x{:}\, n) \Rightarrow nil} \qquad \frac{\begin{array}{l} n \Rightarrow s\, n_v \\ e[0'] \Rightarrow v \quad (e[s'\, x]|x{:}\, n_v) \Rightarrow lv \end{array}}{(e[x]|x{:}\, n) \Rightarrow v, lv}$$

$$\frac{e \Rightarrow v, lv \quad \iota \Rightarrow 0'}{e\, \iota \Rightarrow v} \qquad \frac{e \Rightarrow v, lv \quad \iota \Rightarrow s'\, \iota_v \quad lv\, \iota_v \Rightarrow v'}{e\, \iota \Rightarrow v'}$$

$$\frac{\begin{array}{l} e \Rightarrow (e'[x]|x) \\ n \Rightarrow n_v \quad e'[n_v] \Rightarrow v \end{array}}{e\, n \Rightarrow v} \qquad \frac{\begin{array}{l} e \Rightarrow e', Le \\ n \Rightarrow 0 \quad e' \Rightarrow v \end{array}}{e\, n \Rightarrow v} \qquad \frac{\begin{array}{l} e \Rightarrow e', Le \\ n \Rightarrow s\, n_v \quad Le\, n_v \Rightarrow v \end{array}}{e\, n \Rightarrow v}$$

Well-formed $\Gamma_n \vdash n\colon Arity \quad \Gamma_n \vdash K\colon Kind$

$$\frac{\Gamma_n \vdash}{\Gamma_n \vdash x\colon Arity}\ \Gamma_n(x) = Arity \qquad \frac{\Gamma_n \vdash}{\Gamma_n \vdash 0\colon Arity} \qquad \frac{\Gamma_n \vdash n\colon Arity}{\Gamma_n \vdash s\, n\colon Arity}$$

$$\frac{\Gamma_n \vdash}{\Gamma_n \vdash Type\colon Kind} \qquad \frac{\Gamma_n \vdash n\colon Arity \quad \Gamma_n \vdash K\colon Kind}{\Gamma_n \vdash K^n\colon Kind}$$

Well-formed $\Gamma_n;\Gamma_\iota \vdash \quad \Gamma_n;\Gamma_\iota \vdash \iota\colon n \quad \Gamma_n;\Gamma_\iota \vdash l\iota\colon m{\Rightarrow}n$

$$\frac{\Gamma_n \vdash}{\Gamma_n;\emptyset \vdash} \qquad \frac{\Gamma_{n\iota} \vdash n\colon Arity}{\Gamma_{n\iota}, x\colon n \vdash} \qquad \frac{\Gamma_{n\iota} \vdash}{\Gamma_{n\iota} \vdash x\colon n}\ \Gamma_\iota(x) = n$$

$$\frac{\Gamma_{n\iota} \vdash n\colon Arity}{\Gamma_{n\iota} \vdash 0'\colon s\, n} \qquad \frac{\Gamma_{n\iota} \vdash \iota\colon n}{\Gamma_{n\iota} \vdash s'\,\iota\colon s\, n} \qquad \frac{\Gamma_{n\iota} \vdash l\iota\colon m{\Rightarrow}n \quad \Gamma_{n\iota} \vdash \iota\colon m}{\Gamma_{n\iota} \vdash l\iota\ \iota\colon n}$$

$$\frac{\Gamma_{n\iota}, x\colon m \vdash \iota\colon n}{\Gamma_{n\iota} \vdash (\iota|x\colon m)\colon m{\Rightarrow}n} \qquad \frac{\Gamma_{n\iota} \vdash n\colon Arity}{\Gamma_{n\iota} \vdash Nil_n\colon 0{\Rightarrow}n} \qquad \frac{\Gamma_{n\iota} \vdash \iota\colon n \quad \Gamma_{n\iota} \vdash l\iota\colon m{\Rightarrow}n}{\Gamma_{n\iota} \vdash \iota, l\iota\colon s\, m{\Rightarrow}n}$$

Well-formed $\Gamma_n; \Gamma_\iota; \Gamma_u \vdash u\colon K$

$$\frac{\Gamma_{n\iota} \vdash}{\Gamma_{n\iota}; \emptyset \vdash} \qquad \frac{\Gamma_{n\iota u} \vdash K\colon Kind}{\Gamma_{n\iota u}, x\colon K \vdash} \qquad \frac{\Gamma_{n\iota u} \vdash}{\Gamma_{n\iota u} \vdash x\colon K}\ \Gamma_u(x) = K$$

$$\frac{\Gamma_{n\iota u} \vdash \tau_1, \tau_2\colon Type}{\Gamma_{n\iota u} \vdash \tau_1 \to \tau_2\colon Type} \qquad \frac{\Gamma_{n\iota u}, x\colon K \vdash \tau\colon Type}{\Gamma_{n\iota u} \vdash \forall x\colon K.\tau\colon Type}$$

$$\frac{\Gamma_{n\iota}, x\colon m; \Gamma_u \vdash u\colon K}{\Gamma_{n\iota u} \vdash (u\colon K | x\colon m)\colon K^m} \qquad \frac{\Gamma_{n\iota u} \vdash u\colon K^m \quad \Gamma_{n\iota} \vdash \iota\colon m}{\Gamma_{n\iota u} \vdash u\,\iota\colon K}$$

$$\frac{\Gamma_{n\iota u} \vdash K\colon Kind}{\Gamma_{n\iota u} \vdash Nil_K\colon K^0} \qquad \frac{\Gamma_{n\iota u} \vdash u\colon K \quad \Gamma_{n\iota u} \vdash lu\colon K^m}{\Gamma_{n\iota u} \vdash u, lu\colon K^{s\,m}}$$

$$\frac{\Gamma_{n\iota u} \vdash lu\colon Type^m}{\Gamma_{n\iota u} \vdash \prod(lu)\colon Type} \qquad \frac{\Gamma_n, x\colon Arity; \Gamma_{\iota u} \vdash \tau\colon Type}{\Gamma_{n\iota u} \vdash \forall x\colon Arity.\tau\colon Type}$$

Well-formed $\Gamma \vdash e\colon \tau$ and equality of types

As in $F\omega$ we need a type-conversion rule

$$\frac{\Gamma \vdash e\colon \tau_1 \quad \Gamma_{n\iota u} \vdash \tau_1 = \tau_2\colon Type}{\Gamma \vdash e\colon \tau_2}$$

thus we must define $\Gamma_{n\iota u} \vdash u_1 = u_2\colon K$, and also $\Gamma_{n\iota} \vdash \iota_1 = \iota_2\colon n$

1. define prime contexts Γ^* and the prime covering $C(\Gamma)$ of a context Γ, i.e. the most refined covering of Γ [Coq92@TYPES]

2. define (long) normal forms $[\![\iota]\!]$ and $[\![u]\!]$ for ι and u well-formed in a prime context [reduction free normalization]

3. define equality for ι and u well-formed in an arbitrary context.

Prime Coverings

- $\{S_p\colon \Gamma_p \to \Gamma | p \in P\}$ covering $\overset{\Delta}{\Longleftrightarrow}$ coproduct in the category of contexts.

- $C(\Gamma)$ prime covering of Γ, i.e. the most refined covering of Γ (in an extensive category of contexts).

- prime contexts $\Gamma^*_\iota ::= \emptyset \mid \Gamma^*_\iota, x\colon n^*$ where $n^* ::= x$
- normal forms in prime contexts $\widehat{\iota} ::= x \mid 0' \mid s'\,\widehat{\iota}$

- $C(x\colon n) = \{\Gamma^*_{\iota,p}.\widehat{\iota}_p | p \in P\}$ is defined by induction on the structure of n

$x\colon n$	$\Gamma^*_{\iota,p}.\ \widehat{\iota}_p$	$p \in P$
$x\colon y$	$x\colon y.\ x$	
$x\colon 0$		$\_ \in 0$
$x\colon s\,n$	$\emptyset.\ 0'$ $\Gamma^*_\iota.\ s'\,\widehat{\iota}$	 $\Gamma^*_\iota.\widehat{\iota} \in C(x\colon n)$

Prop. $\Gamma_n; x\colon n \vdash$ and $\Gamma^*_\iota.\widehat{\iota} \in C(x\colon n)$ imply $\Gamma_n; \Gamma^*_\iota \vdash \widehat{\iota}\colon n$

Normal forms in prime contexts $\Gamma_n; \Gamma_\iota^* \vdash \widehat{\iota}\colon n$

$\widehat{\iota} ::= x \mid 0' \mid s'\,\widehat{\iota}$

Prop. $\Gamma_n; \Gamma_\iota^* \vdash \iota\colon n$ implies $\Gamma_n; \Gamma_\iota^* \vdash [\![\iota]\!]\colon n$

Prop. $\Gamma_n; \Gamma_\iota^* \vdash l\iota\colon m \Rightarrow n$ and $\Gamma_n; \Gamma_\iota^* \vdash \widehat{\iota}\colon m$ imply $\Gamma_n; \Gamma_\iota^* \vdash [\![l\iota]\!](\widehat{\iota})\colon n$

ι	$[\![\iota]\!]$
x	x
$0'$	$0'$
$s'\,\iota$	$s'\,[\![\iota]\!]$
$l\iota\,\iota$	$[\![l\iota]\!]([\![\iota]\!])$

$l\iota$	$\widehat{\iota}$	$[\![l\iota]\!](\widehat{\iota})$
$(\iota[x] \vert x\colon m)$	$\widehat{\iota}$	$[\![\iota[\widehat{\iota}]]\!]$
$\iota, l\iota$	$0'$	$[\![\iota]\!]$
$\iota, l\iota$	$s'\,\widehat{\iota}$	$[\![l\iota]\!](\widehat{\iota})$
$\_$	$\_$	$fail$

$[\![\iota]\!]$ / $[\![l\iota]\!](\_)$ are defined by lexicographic induction on $(\#\iota, |\iota|)$ / $(\#l\iota, |l\iota|)$

ι	$\#\iota$	$\lvert\iota\rvert$
x	0	1
$0'$	0	1
$s'\,\iota$	$\#\iota$	$\lvert\iota\rvert + 1$
$l\iota\,\iota$	$\#l\iota + \#\iota$	$\lvert l\iota\rvert + \lvert\iota\rvert + 1$

$l\iota$	$\#l\iota$	$\lvert l\iota\rvert$
Nil_n	1	0
$(\iota \vert x\colon m)$	$\#\iota + 1$	0
$\iota, l\iota$	$\#\iota + \#l\iota$	$\lvert l\iota\rvert + 1$

Normal forms in prime contexts $\Gamma_n; \Gamma_\iota^*; \Gamma_u \vdash \hat{u} \colon K$

$\hat{\tau}, \hat{u} ::= x\,\bar{\hat{\iota}} \mid \widehat{lu} \mid \hat{\tau}_1 \to \hat{\tau}_2 \mid \forall x\colon K.\hat{\tau} \mid \prod \widehat{lu} \mid \forall x\colon Arity.\hat{\tau}$
where $\widehat{lu} ::= (\hat{u}\colon K | x\colon n^*) \mid Nil_K \mid \hat{u}, \widehat{lu}$
Prop. $\Gamma_n; \Gamma_\iota^*; \Gamma_u \vdash u\colon K$ implies $\Gamma_n; \Gamma_\iota^*; \Gamma_u \vdash [\![u]\!]_{\Gamma_u}\colon K$

u	$[\![u]\!]_{\Gamma_u}$
x	$long(x, \Gamma_u(x))$
$u\,\iota$	$[\![u]\!]_{\Gamma_u} * [\![\iota]\!]$
$\tau_1 \to \tau_2$	$[\![\tau_1]\!]_{\Gamma_u} \to [\![\tau_2]\!]_{\Gamma_u}$
$\forall x\colon K.\tau$	$\forall x\colon K.[\![\tau]\!]_{\Gamma_u, x:K}$
$\prod(lu)$	$\prod([\![lu]\!]_{\Gamma_u})$
$\forall x\colon Arity.\tau$	$\forall x\colon Arity.[\![\tau]\!]_{\Gamma_u}$

lu	$[\![lu]\!]_{\Gamma_u}$
$(u\colon K\|x\colon y)$	$([\![u]\!]_{\Gamma_u}\colon K\|x\colon y)$
$(u\colon K\|x\colon 0)$	Nil_K
$(u[x]\colon K\|x\colon s\,m)$	$[\![u[0']]\!]_{\Gamma_u}$, $[\![(u[s'\,x]\colon K\|x\colon m)]\!]_{\Gamma_u}$
Nil_K	Nil_K
u, lu	$[\![u]\!]_{\Gamma_u}, [\![lu]\!]_{\Gamma_u}$

$[\![u]\!]_{\Gamma_u}$ is defined by induction on $|\Gamma_u| + |u|$

u	$\|u\|$
x	1
$(u\colon K\|x\colon m)$	$\|u\| + \|m\|$
Nil_K	1
u, lu	$\|u\| + \|lu\|$
$u\,\iota$	$\|u\| + 1$

u	$\|u\|$
$\tau_1 \to \tau_2$	$\|\tau_1\| + \|\tau_2\| + 1$
$\forall x\colon K.\tau$	$\|K\| + \|\tau\| + 1$
$\prod(lu)$	$\|lu\| + 1$
$\forall x\colon Arity.\tau$	$\|\tau\| + 1$

Expansion of variables to long normal-form

K	$long(x, K)$
$Type$	x
K^m	$long(x\,z\colon K \mid z\colon m)$
m	$long(x\,\iota[z]\colon K \mid z\colon m)$
y	$(long(x, K)[x\colon= x\,\iota[z]] \mid z\colon y)$
0	Nil_K
$s\,m$	$long(x, K)[x\colon= x\,\iota[0']], long(x\,\iota[s'\,z]\colon K \mid z\colon m)$

Application of long normal-forms

$\widehat{lu}$	$\widehat{\iota}$	$\widehat{lu} * \widehat{\iota}$
$(\widehat{u}\colon K \mid x\colon n^*)$	z	$\widehat{u}[x\colon= z]$
$\widehat{u}, \widehat{lu}$	$0'$	$\widehat{u}$
$\widehat{u}, \widehat{lu}$	$s'\,\widehat{\iota}$	$\widehat{lu} * \widehat{\iota}$
_	_	$fail$

Equality of types $\Gamma_n; \Gamma_\iota; \Gamma_u \vdash \tau_1 = \tau_2 \colon Type$

$\Gamma_n; \Gamma_\iota; \Gamma_u \vdash \tau_1 = \tau_2 \colon Type$ holds $\overset{\Delta}{\Longleftrightarrow}$

- $\Gamma_n; \Gamma_\iota; \Gamma_u \vdash \tau_1, \tau_2 \colon Type$ well-formed

- $[\![\tau_1[S]]\!]_{\Gamma_u} \equiv [\![\tau_2[S]]\!]_{\Gamma_u}$ for every substitution $S \colon \Gamma_\iota^* \to \Gamma_\iota$ in $C(\Gamma_\iota)$

By the properties of $C(\Gamma_\iota)$ and $[\![u]\!]_{\Gamma_u}$ one has

- $\Gamma_n; \Gamma_\iota^*; \Gamma_u \vdash \tau_i[S] \colon Type$ well-formed
- $[\![\tau_1[S]]\!]_{\Gamma_u}$ defined
- $\Gamma_n; \Gamma_\iota^*; \Gamma_u \vdash [\![\tau_i[S]]\!]_{\Gamma_u} \colon Type$ well-formed.

Well-formed $\Gamma \vdash$ $\quad \Gamma \vdash e\colon \tau$

$$\frac{\Gamma_{n\iota u} \vdash}{\Gamma_{n\iota u}; \emptyset \vdash} \qquad \frac{\Gamma \vdash \tau\colon Type}{\Gamma, x\colon \tau \vdash} \qquad \frac{\Gamma \vdash}{\Gamma \vdash x\colon \tau}\ \Gamma_e(x) = \tau \qquad \frac{\Gamma, x\colon \tau \vdash e\colon \tau}{\Gamma \vdash \mu x\colon \tau . e\colon \tau}$$

$$\frac{\Gamma, x\colon \tau_1 \vdash e\colon \tau_2}{\Gamma \vdash \lambda x\colon \tau_1 . e\colon \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash e_1\colon \tau_1 \to \tau_2 \quad \Gamma \vdash e_2\colon \tau_1}{\Gamma \vdash e_1\, e_2\colon \tau_2}$$

$$\frac{\Gamma_{n\iota u}, x\colon K; \Gamma_e \vdash e\colon \tau}{\Gamma \vdash \lambda x\colon K . e\colon \forall x\colon K . \tau} \qquad \frac{\Gamma \vdash e\colon \forall x\colon K . \tau \quad \Gamma_{n\iota u} \vdash u\colon K}{\Gamma \vdash e\, u\colon \tau[x\colon = u]}$$

$$\frac{\Gamma \vdash e\colon \tau_1 \quad \Gamma_{n\iota u} \vdash \tau_1 = \tau_2\colon Type}{\Gamma \vdash e\colon \tau_2} \qquad \frac{\Gamma_{n\iota}, x\colon n; \Gamma_{ue} \vdash e\colon \tau}{\Gamma \vdash (e | x\colon n)\colon \prod(\tau | x\colon n)} \qquad \frac{\Gamma \vdash e\colon \prod(lu) \quad \Gamma_{n\iota} \vdash \iota\colon n}{\Gamma \vdash e\colon lu\, \iota}$$

$$\frac{\Gamma \vdash}{\Gamma \vdash nil\colon \prod(Nil_{Type})} \qquad \frac{\Gamma \vdash e\colon \tau \quad \Gamma \vdash le\colon \prod(lu)}{\Gamma \vdash e, le\colon \prod(\tau, lu)}$$

$$\frac{\Gamma_n, x\colon Arity; \Gamma_{\iota ue} \vdash e\colon \tau}{\Gamma \vdash (e | x\colon Arity)\colon \forall x\colon Arity . \tau} \qquad \frac{\Gamma \vdash e\colon \forall x\colon Arity . \tau \quad \Gamma_n \vdash n\colon Arity}{\Gamma \vdash e\, n\colon \tau[x\colon = n]} \qquad \frac{\Gamma \vdash e\colon \tau[0] \quad \Gamma \vdash Le\colon \forall x\colon Arity . \tau[s\, x]}{\Gamma \vdash e, Le\colon \forall x\colon Arity . \tau[x]}$$

Uniqueness of types: case e, Le

Prop. $\Gamma_n, x{:}\,Arity; \Gamma_\iota^*, y_i{:}\,x; \Gamma_u[x] \vdash \hat{u}_1[x, y_i], \hat{u}_2[x, y_i]{:}\,K[x]$ and $[\![\hat{u}_1[s\,x, s'\,y_i]]\!]_{\Gamma_{u[s\,x]}} \equiv [\![\hat{u}_2[s\,x, s'\,y_i]]\!]_{\Gamma_{u[s\,x]}}$ imply $\hat{u}_1[x, y_i] \equiv \hat{u}_2[x, y_i]$.

Hint. $\hat{u}_i^*[x, y_i] \triangleq [\![\hat{u}_i[s\,x, s'\,y_i]]\!]_{\Gamma_{u[s\,x]}}$ is obtained from $\hat{u}_i$, x and the y_i
by the obvious substitutions, and
by replacing $(\hat{u}[x, y_i, y]{:}\,K|y{:}\,x)$ with $(\hat{u}[y{:}= 0'])^*[x, y_i], (\hat{u}^*[x, y_i, y]{:}\,K|y{:}\,x)$.
Thus $\hat{u}_i$ is determined by $\hat{u}_i^*$, x and the y_i.

Type inference algorithm $Ty(\Gamma, e)$

The algorithm proceeds by induction on $|\Gamma| + |e|$ and uses the algorithms for computing normal forms, and the fact above.

Example: Generic Matrix-Multiplication

- $Vec(m\colon Arity) \triangleq Real^m$ real-valued vector of size m
- $Mat(m,n\colon Arity) \triangleq (Real^n)^m$ real-valued $m \times n$ matrix
- $GSP\colon \forall m\colon Arity.Vec(m) \to Vec(m) \to Real$ scalar product
- $GMM\colon \forall m,n,p\colon Arity.Mat(m,n) \to Mat(n,p) \to Mat(m,p)$

```
fun GSP 0 _ _ = 0.0
  | GSP (s m) (x,U) (y,V) = x*y+(GSP m U V);
fun GMM m n p A B i j = GSP n (A i k|k:n) (B k j|k:n);
```

Counter-example: Generic Merge

- $GM\colon \forall m,n\colon Arity.Vec(m) \to Vec(n) \to Vec(m+n)$ order-preserving merge of (ordered) sequences. Problems: $m+n$ is not definable, but one can use existential types [XP00@POPL].

Example: Generic Zip

- $GZ\colon \forall m\colon Arity.\forall X\colon Type^{s\,m}.\prod(list\ \ X\,\iota \mid \iota\colon s\,m) \to list\ \ \prod(X\,\iota \mid \iota\colon s\,m)$
generic zip of an m-tuple of lists

```
fun GZ 0 (t,Nil) (l,nil) = let fun f(x:t) = x,nil in map f l
  | GZ (s m) (t,tt) (l,ll) = let fun f(x:t,xx:\/tt) = x,xx
                             in map f (zip l (GZ m tt ll));
fun map f [] = [] | map (x::lx) = (f x)::(map f lx);
fun zip [] [] = [] | zip (x::lx) (y::ly) = (x,y)::(zip lx ly);
```

Example: simpler and more refined Generic Zip [MBJ99@CTCS]

- $GZ\colon\ \forall m,n\colon Arity.\forall X\colon (Type^{n})^{m}.$
$\prod(\prod(X\,\iota\,j \mid j\colon n) \mid \iota\colon m) \to \prod(\prod(X\,\iota\,j \mid \iota\colon m) \mid j\colon n)$

```
fun GZ m n X x = ((x i j | i:m) | j:n)
```

does not raise exceptions

Summary

Type system for arity polymorphism whose main features are:
- four level stratification: $Type \succ Kind > Arity \succ Kind'$;
- limited for of inductive types (with prime coverings);
- $Arity$ is an inductive type only for e;
- decidable typing and extensional equality of types;
- arities n and indexes ι have computational content (unlike DML).

Further extensions/improvements

- reacher arity expressions $n ::= x \mid 0 \mid 1 \mid n_1 + n_2 \mid n_1 \times n_2$
  (there are non-trivial computationally-expensive isomorphisms);
- existential types, reduced type annotation [Xi98@PhD, XP99@POPL];
- higher-order kinds $K_1 \to K_2$ and $\forall x{:}\, Arity.K$;
- recursive types ($\mu X{:}\, K.u$).

Unexplored connections

- polykinded types for polytypic values [Hin00@MPC]:
  meta-level induction on K/u (similar to definition of logical relation).

References

[Bar92@HB]: typed lambda calculi
[Jac91@PhD,Jac99@book]: categorical logic and type theory
[Geu93@PhD]: PTS with $\beta\eta$-conversion

[Coq92@TYPES]: pattern matching in type theory, covering

[JBM98@JFP]: Functorial ML
[MBJ99@CTCS]: traversals and zips
[Jay00]: Functorial Lambda-Calculus
[Tul00@MPC]: Zip calculus

[Xi98@PhD, XP99@POPL]: DML

[Hin00@POPL,Hin00@MPC]: new approach to polytypic programming

[Bar92@HB] H.P. Barendregt. Lambda calculi with types. In D. M. Gabbai Samson Abramski and T. S. E. Maiboum, editors, *Handbook of Logic in Computer Science*. Oxford University Press, Oxford, 1992.

[Coq92@TYPES] T. Coquand. Pattern matching with dependent types. In *Workshop on Types for Proofs and Programs*, Båstad, Sweden, 1992.

[Geu93@PhD] H. Geuvers. *Logics and Type Systems*. PhD thesis, Computer Science Institute, Katholieke Universiteit Nijmegen, 1993.

[Hin00@POPL] R. Hinze. A new approach to generic functional programming. In *POPL'00*, Boston, January 2000.

[Hin00@MPC] R. Hinze. Polytypic values possess polykinded types. In *MPC'00*, volume ?? of *LNCS*. Springer Verlag, 2000.

[Jac91@PhD] B. Jacobs. *Categorical type theory*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1991.

[Jac99@book] B. Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Elsevier, 1999.

[Jay00] C.B. Jay. Functorial lambda-calculus. 2000.

[JBM98@JFP] B. Jay, G. Bellè, and E. Moggi. Functorial ML. *JFP*, 8(6), 1998.

[MBJ99@CTCS] E. Moggi, G. Bellè, and C.B. Jay. Monads, shapely functors and traversals. In *CTCS'99*, volume 29 of *ENTCS*. Elsevier, 1999.

[Tul00@MPC] M. Tullsen. The zip calculus. In *MPC'00*, volume ?? of *LNCS*. Springer Verlag, 2000.

[Xi98@PhD] H. Xi. *Dependent Types in Practical Programming*. PhD thesis, School of Computer Science, CMU, 1998.

[XP99@POPL] H. Xi and F. Pfenning. Dependent types in practical programming. In *POPL'99*, San Antonio, January 1999.