Polymorphic variants in Objective Caml 3
(draft)

Jacques Garrigue

July 7, 2000

This short note describes theoretical background for polymorphic variants, as they are implemented in Objective Caml version 3.00 [LDG+00]. This differs from the original Objective Label approach [Gar98] in subtle ways.

1 Types and kinds

The types are just the traditional ML-style ones. τ stands for monotypes, and σ for polytypes.

$$\begin{array}{rcll} \tau & ::= & \alpha & \text{type variable} \\ & | & u & \text{base type} \\ & | & \tau \to \tau & \text{function type} \\ \sigma & ::= & \forall\alpha\ldots\alpha.\tau & \text{polytypes} \end{array}$$

However, a type is to be understood in the context of a kinding environment K. K is a set of bindings $\alpha :: (L, U, T, F)$, describing the possible variant values admitted for the type α, where

- L is a finite subset of constructor names $\mathcal{L}$. This is the lower bound on the constructors handled by this type.
- U is either a finite subset of $\mathcal{L}$, or $\mathcal{L}$ itself. This is the upper bound on the constructors handled by this type.
- T is a function from constructor names $\mathcal{L}$ to finite sets of monotypes. Each of these type sets is to be understood as a conjunctive type required for any value injected through a constructor.
- F is a set of supplementary free type variables. It has no specific meaning, but is needed to obtain the principal type property, and allow complete type inference.

A kind is well formed if

1. the lower bound is included in the upper bound, that is $L \subseteq U$.
2. for each l in L, the set $T(l)$ is a singleton.

By definition we suppose that kinding environments contain only well formed kinds. We write respectively $\mathrm{K_T}$, $\mathrm{K_L}$, $\mathrm{K_U}$ and $\mathrm{K_F}$ for the functions from type variables to each component of the associated kind in the kinding environment K. Notice that all type variables do not necessarily have a kind, only those that represent variant types do.

Free variables $\mathrm{FV_K}(\sigma)$ of a polytype σ under a kinding environment K are defined as the minimum set statisfying the following equations. (We write $\mathrm{FV}(\sigma)$ as K will always be clear from the context.)

$$\begin{array}{rcl} \mathrm{FV}(\forall\alpha_1\ldots\alpha_n.\tau) & = & \mathrm{FV}(\tau) \setminus \{\alpha_1,\ldots,\alpha_n\} \\ \mathrm{FV}(\alpha) & = & \{\alpha\} \cup \mathrm{FV}(\mathrm{K_T}(\alpha)) \cup \mathrm{K_F}(\alpha) \\ \mathrm{FV}(u) & = & \emptyset \\ \mathrm{FV}(\tau_1 \to \tau_2) & = & \mathrm{FV}(\tau_1) \cup \mathrm{FV}(\tau_2) \end{array}$$

A type substitution θ, extended as usual on monotypes and polytypes, is *admissible* under for the kinding environments K and K′, written $\mathrm{K} \vdash \theta : \mathrm{K}'$, if for all α in Dom(K), $\theta(\alpha)$ is a type variable α' and it satisfies the following properties.

1. $\alpha' \in \mathrm{Dom}(\mathrm{K}')$
2. $\mathrm{K_L}(\alpha) \subseteq \mathrm{K'_L}(\alpha')$
3. $\mathrm{K'_U}(\alpha') \subseteq \mathrm{K_U}(\alpha)$
4. for each l in $\mathrm{K'_U}(\alpha')$, $\theta(\mathrm{K_T}(\alpha)(l)) \subseteq \mathrm{K'_T}(\alpha')(l)$
5. $\mathrm{FV}(\theta(\mathrm{K_T}(\alpha))) \cup \mathrm{FV}(\theta(\mathrm{K_F}(\alpha))) \subseteq \mathrm{FV}(\mathrm{K'_T}(\alpha')) \cup \mathrm{K'_F}(\alpha')$

Condition 1 ensures that all kinded variables are mapped to kinded variables. Conditions 2 and 3 ensure that lower bound and upper bound are instanciated correctly (respectively covariant and contravariant). Condition 4 ensures that all type constraints on possible constructors are kept. Finally condition 5 ensures that no free variables are forgotten by instanciation (this would break the principal type property).

2 Terms and typing rules

Expressions are

$$
\begin{array}{rcll}
e & ::= & x \mid \mathsf{fun}\ x \to e \mid e\ e & \text{core lambda} \\
 & \mid & (l\ e) & \text{variant} \\
 & \mid & \mathsf{function}\ r \mid \ldots \mid r & \text{pattern matching} \\
r & ::= & (l\ x) \to e & \text{decomposition} \\
 & \mid & (l\_ \mid \ldots \mid l\_\ \mathsf{as}\ x) \to e & \text{dispatch}
\end{array}
$$

Type judgments have the two forms

$$
\begin{array}{l}
\mathrm{K};\Gamma \vdash e : \tau \\
\mathrm{K};\Gamma \vdash r : \alpha \to \tau
\end{array}
$$

where K is a well-formed kinding environment and Γ is a set of bindings $x : \sigma$ from term variables to polytypes.

Rules are written in the syntax-directed style.

Variable

$$\frac{\mathrm{K} \vdash \theta : \mathrm{K} \quad \mathrm{Dom}(\theta) \subseteq B}{\mathrm{K};\Gamma, x : \forall B.\tau \vdash x : \theta(\tau)}$$

Abstraction

$$\frac{\mathrm{K};\Gamma, x : \tau \vdash e : \tau'}{\mathrm{K};\Gamma \vdash \mathsf{fun}\ x \to e : \tau \to \tau'}$$

Application

$$\frac{\mathrm{K};\Gamma \vdash e_1 : \tau \to \tau' \quad \mathrm{K},\Gamma \vdash e_2 : \tau}{\mathrm{K};\Gamma \vdash e_1\ e_2 : \tau'}$$

Let

$$\frac{\mathrm{K};\Gamma \vdash e_1 : \tau_1 \quad \mathrm{K};\Gamma, x : \forall B.\tau_1 \vdash e_2 : \tau_2}{\mathrm{K};\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau_2}\ B = \mathrm{FV}(\tau_1) \setminus \mathrm{FV}(\Gamma)$$

Variant

$$\frac{\mathrm{K};\Gamma \vdash e : \tau \quad l \in \mathrm{K_L}(\alpha) \quad \tau \in \mathrm{K_T}(\alpha)(l)}{\mathrm{K};\Gamma \vdash (l\ e) : \alpha}$$

Matching

$$\frac{\begin{array}{l}\mathrm{K};\Gamma \vdash r_i : \alpha_i \to \tau \quad \mathrm{K_T}(\alpha_i) \subseteq \mathrm{K_T}(\alpha) \quad (1 \leq i \leq n) \\ \mathrm{K_F}(\alpha_i) \subseteq \mathrm{FV}(\mathrm{K_T}(\alpha)) \cup \mathrm{K_F}(\alpha) \quad \mathrm{K_U}(\alpha) \subseteq \bigcup_{i=1}^{n} \mathrm{K_U}(\alpha_i)\end{array}}{\mathrm{K};\Gamma \vdash \mathsf{function}\ r_1 \mid \ldots \mid r_n : \alpha \to \tau}$$

Decompose

$$\frac{\mathrm{K}; \Gamma, x : \tau_1 \vdash e : \tau_2 \quad l \in \mathrm{K_U}(\alpha) \quad \tau_1 \in \mathrm{K_T}(\alpha)(l)}{\mathrm{K}; \Gamma \vdash (l\ x) \to e : \alpha \to \tau_2}$$

Dispatch

$$\frac{\begin{array}{l}\mathrm{K}; \Gamma, x : \alpha_2 \vdash e : \tau \\ l_i \in \mathrm{K_U}(\alpha_1) \quad l_i \in \mathrm{K_L}(\alpha_2) \quad \mathrm{K_T}(\alpha_2)(l_i) \subseteq \mathrm{K_T}(\alpha_1)(l_i) \quad (1 \leq i \leq n)\end{array}}{\mathrm{K}; \Gamma \vdash (l_{1\_} \mid \ldots \mid l_{n\_}\ \mathsf{as}\ x) \to e : \alpha \to t}$$

3 Unification

We define unfication on monotypes through rules of the form

$$\frac{\varphi}{\varphi'}$$

φ are preconditions, φ' conclusions. φ is a *unification problem* composed both of *kinding constraints* $\alpha :: (L, U, T, F)$ and *equality constraints* $\tau_1 \doteq \tau_2$ connected by $\wedge$. Both $\wedge$ and $\doteq$ commute. A variable may not be kinded by more than one kinding constraint.

An equality constraint is said to be *solved* in φ if it is of the form $\alpha \doteq \tau$ and α only appears as strict subterm in other constraints of φ. A kinding constraint is *solved* if its kind is well-formed. A unification problem φ is solved if all its constraints are solved.

If a unification problem φ has all its kinding constraints solved, then we extract K from these constraints, and a solution to φ is a substitution $\mathrm{K} \vdash \theta : \mathrm{K}'$ such that for all equality constraint $\tau_1 \doteq \tau_2$ in φ, we have $\theta(\tau_1) = \theta(\tau_2)$.

Incompatible

$$\frac{\varphi \wedge \tau_1 \doteq \tau_2}{\perp} \text{ when } sort_\varphi(\tau_1) \neq sort_\varphi(\tau_2)$$

Redundancy

$$\frac{\varphi \wedge \tau \doteq \tau}{\varphi}$$

Function

$$\frac{\varphi \wedge \tau_1 \to \tau_2 \doteq \tau_1' \to \tau_2'}{\varphi \wedge \tau_1 \doteq \tau_1' \wedge \tau_2 \doteq \tau_2'}$$

Substitution

$$\frac{\varphi \wedge \alpha \doteq \tau}{\varphi[\tau/\alpha] \wedge \alpha \doteq \tau} \text{ when } \alpha :: (L, U, T, F) \notin \varphi \text{ and } \alpha \in \mathrm{FV}(\varphi)$$

Variant

$$\frac{\varphi \wedge \alpha_1 :: (L_1, U_1, T_1, F_1) \wedge \alpha_2 :: (L_2, U_2, T_2, F_2) \wedge \alpha_1 \doteq \alpha_2}{\varphi \wedge \alpha :: (L_1 \cup L_2, U_1 \cap U_2, T_1 \sqcup T_2, F_1 \cup F_2) \wedge \alpha_1 \doteq \alpha \wedge \alpha_2 \doteq \alpha} \ \alpha \text{ fresh}$$

Variant field

$$\frac{\varphi \wedge \alpha :: (L, U, T, F) \qquad l \in L \quad T(l) \doteq \{\tau_1, \ldots, \tau_n\}}{\varphi \wedge \alpha :: (L, U, T, F) \wedge \tau_1 \doteq \tau_2 \wedge \ldots \wedge \tau_1 \doteq \tau_n}$$

Bad kind

$$\frac{\varphi \wedge \alpha :: (L, U, T, F) \qquad L \not\subseteq U}{\perp}$$

$sort_\varphi$ is a partial function from types to $\{u_1, \ldots, u_n, \to, v\}$, left undefined on non-kinded variables:

$$\begin{array}{rcll} sort_\varphi(u) & = & u & u \text{ a base type} \\ sort_\varphi(\tau_1 \to \tau_2) & = & \to & \\ sort_\varphi(\alpha) & = & v & \alpha :: (L, U, T, F) \in \varphi \end{array}$$

$\sqcup$ is pointwise union on functions, that is $(T_1 \sqcup T_2)(l) = T_1(l) \cup T_2(l)$.

Proposition 1 *Rewriting a unification problem leads either to $\perp$ or to a solved problem.*

Proposition 2 *From a solved unification problem φ one can directly read a pair of a kinding environment* K *and a substitution* θ*, such that* $\mathrm{K} \vdash \theta : \mathrm{K}$.

A substitution $\mathrm{K} \vdash \theta_1 : \mathrm{K}_1$ is said to be more general than $\mathrm{K} \vdash \theta_2 : \mathrm{K}_2$ if there is a substitution $\mathrm{K}_1 \vdash \theta : \mathrm{K}_2$ such that $\theta_2 = \theta \circ \theta_1$.

Proposition 3 *The solution* $\mathrm{K}' \vdash \theta : \mathrm{K}'$ *read from the solved form obtained by rewriting a unification problem of basis* K *is also* $\mathrm{K} \vdash \theta : \mathrm{K}'$*, and it is the most general unifier for this problem.*

4 Type reconstruction

Type reconstruction is done by translating a typing problem $\mathrm{K}; \Gamma \triangleright e : \tau$ into a unification problem φ, and solving it to obtain a substitution $\mathrm{K} \vdash \theta : \mathrm{K}'$.

The algorithm does not depend on a specific evaluation order, constraint only needing to be solved eagerly for let nodes. In order to support this freedom, types in Γ have to be marked with the kinding environment under which they where generalized, written $\forall_{\mathrm{K}}$. Γ_{K} denotes the marking of all $\forall$ symbols in Γ with K.

$$
\begin{array}{rcl}
\mathrm{K}; \Gamma \triangleright e : \tau & = & (\Gamma_{\mathrm{K}} \triangleright e : \tau) \wedge \mathrm{K} \\
\Gamma, x : \forall_{\mathrm{K}} \alpha_1 ... \alpha_n . \tau_1 \triangleright x : \tau & = & \tau = \theta(\tau_1) \wedge \bigwedge_{\alpha_i :: (L_i, U_i, T_i, F_i) \in \mathrm{K}} \beta_i :: (L_i, U_i, \theta(T_i), \theta(F_i)) \\
 & & \text{where } \theta = \{\alpha_1 \mapsto \beta_1, \ldots, \alpha_n \mapsto \beta_n\} \\
\Gamma \triangleright \mathsf{fun}\ x \to e : \tau & = & (\Gamma, x : \alpha_1 \triangleright e : \alpha_2) \wedge \tau \doteq \alpha_1 \to \alpha_2 \\
\Gamma \triangleright e_1\ e_2 : \tau & = & (\Gamma \triangleright e_1 : \alpha \to \tau) \wedge (\Gamma \triangleright e_2 : \alpha) \\
\Gamma \triangleright \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau & = & \varphi \wedge (\Gamma, x : \forall_{\mathrm{K}}(\mathrm{FV}(\theta(\alpha)) \setminus \mathrm{FV}(\theta(\Gamma))).\theta(\alpha) \triangleright e_2 : \tau) \\
 & & \text{where } \varphi = (\Gamma \triangleright e_1 : \alpha) \text{ and } K \vdash \theta : K = \mathrm{solve}(\varphi) \\
\Gamma \triangleright (l\ e) : \tau & = & (\Gamma \triangleright e : \alpha_1) \wedge \alpha :: (\{l\}, \mathcal{L}, \{l \mapsto \{\alpha\}\}, \emptyset) \wedge \tau \doteq \alpha \\
\Gamma \triangleright \mathsf{function}\ r_1 \mid \ldots \mid r_n : \tau & = & \bigwedge_{i=1}^{n} (\Gamma \triangleright r_i : \tau) \wedge \tau \doteq \alpha_1 \to \alpha_2 \wedge \alpha_1 :: (\emptyset, U, \emptyset, \emptyset) \\
 & & \text{where } U \text{ is the set of all variant tags in } r_1 \ldots r_n \\
\Gamma \triangleright (l\ x) \to e : \tau & = & (\Gamma, x : \alpha \triangleright e : \alpha_2) \wedge \tau \doteq \alpha_1 \to \alpha_2 \wedge \alpha_1 :: (\emptyset, \mathcal{L}, \{l \mapsto \{\alpha\}\}, \emptyset) \\
\Gamma \triangleright (l_1\_ \mid \ldots \mid l_n\_\ \mathsf{as}\ x) : \tau & = & (\Gamma, x : \alpha \triangleright e : \alpha_2) \wedge \tau \doteq \alpha_1 \to \alpha_2 \wedge \alpha_1 :: (\emptyset, \mathcal{L}, T, \emptyset) \wedge \alpha :: (L, \mathcal{L}, T, \emptyset) \\
 & & \text{where } L = \{l_1, \ldots, l_n\} \text{ and } T = \{l_1 \mapsto \{\beta_1\}, \ldots, l_n \mapsto \{\beta_n\}\}
\end{array}
$$

A solution to a typing proble $\mathrm{K}; \Gamma \triangleright e : \tau$ is a substitution $\mathrm{K} \vdash \theta : \mathrm{K}'$ such that $\mathrm{K}'; \theta(\Gamma) \vdash e : \theta(\tau)$ is derivable.

Proposition 4 *If* $\mathrm{K}; \Gamma \triangleright e : \tau$ *can be reduced to* $\mathrm{K} \vdash \theta : \mathrm{K}'$ *by the type reconstruction algorithm,* $\mathrm{K}'; \theta(\Gamma) \vdash e : \theta(\tau)$ *is derivable, and* $\theta(\tau)$ *principal; otherwise it reduces to* $\perp$ *and there is no solution.*

References

[Gar98] Jacques Garrigue. Programming with polymorphic variants. In *ML Workshop*, Baltimore, September 1998.

[LDG+00] Xavier Leroy, Damien Doligez, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. *The Objective Caml system release 3.00, Documentation and user's manual.* Projet Cristal, INRIA, April 2000.