# C\ TYPE SYSTEM

WILL VEATCH

We define the System $F_\omega^{AL}$, the basis for the C\ language. System $F_\omega^{AL}$ is system $F_\omega$ extended with affine and linear kinds. That is, we have have kinds $\star$ for unrestricted types, $\bullet$ for affine types, and $\circ$ for linear types, with the following subkinding relation: $\star \leq \bullet \leq \circ$. Alternatively, System $F_\omega^{AL}$ is System $F^\circ$, described in [1], extended with type functions and an affine kind. We take inspiration from [3, 2].

| | |
|---|---|
| $\kappa ::= \star \mid \bullet \mid \circ \mid \kappa \Rightarrow \kappa$ | kinds |
| $\tau ::= \alpha \mid \tau \xrightarrow{\kappa} \tau \mid \forall \alpha{:}\kappa.\, \tau \mid \lambda(\alpha : \kappa).\, \tau \mid \tau\ \tau$ | types |
| $e ::= x \mid \lambda^\kappa(x : \tau).\, e \mid e\ e \mid \Lambda(\alpha : \kappa).\, v \mid e[\tau]$ | expressions |
| $v ::= \lambda^\kappa(x : \tau).\, e \mid \Lambda(\alpha : \kappa).\, v$ | values |

*Remark* 0.1 (Type and Data Constructors). Type constructors, and even type functions, do not need any kind constraints. Their kinds can simply be of the form $k_1 \Rightarrow \cdots \Rightarrow k_n$. This is because the type constructor does not have any information about then usage of variables passed to the data constructor for a given type. Constraints likely must be present in data constructors. This choice allows for better kind polymorphism.

Consider the `Either` type. In our language we would have

$$\texttt{type Either} = \lambda(a : \kappa_1).\, \lambda(b : \kappa_2).\, (a|b) : \kappa_3$$

We could add the constraints $\kappa_1 \leq \kappa_3, \kappa_2 \leq \kappa_3$. The constructors would then have the following type signatures:

$$\texttt{Left} : \kappa_1 \leq \kappa_3, \kappa_2 \leq \kappa_3 \Rightarrow \forall a{:}\kappa_1.\, \forall b{:}\kappa_2.\, a \xrightarrow{\kappa_4} (\texttt{Either}\ a\ b) : \kappa_3$$
$$\texttt{Right} : \kappa_1 \leq \kappa_3, \kappa_2 \leq \kappa_3 \Rightarrow \forall a{:}\kappa_1.\, \forall b{:}\kappa_2.\, b \xrightarrow{\kappa_4} (\texttt{Either}\ a\ b) : \kappa_3$$

However, suppose $\kappa_1 = \star$ and $\kappa_2 = \circ$. Then $(\texttt{Either}\ a\ b) : \circ$. So $(\texttt{Left}\ x) : \_ : \circ$ and $(\texttt{Right}\ \ y) : \_ : \circ$. However, it should be that $(\texttt{Left}\ x) : \star$. Thus, we do not constrain the type constructor, only the data constructors:

$$\texttt{Left} : \kappa_1 \leq \kappa_3 \Rightarrow \forall a{:}\kappa_1.\, \forall b{:}\kappa_2.\, a \xrightarrow{\kappa_4} (\texttt{Either}\ a\ b) : \kappa_3$$
$$\texttt{Right} : \kappa_2 \leq \kappa_3 \Rightarrow \forall a{:}\kappa_1.\, \forall b{:}\kappa_2.\, b \xrightarrow{\kappa_4} (\texttt{Either}\ a\ b) : \kappa_3$$

---

*Date*: August 29, 2024.

$$\frac{\Gamma \vdash \tau : \star}{\Gamma \vdash \tau : \bullet}\ [\textsf{K-Sub}_1] \qquad \frac{\Gamma \vdash \tau : \bullet}{\Gamma \vdash \tau : \circ}\ [\textsf{K-Sub}_2] \qquad \frac{}{\vdash \cdot}\ [\textsf{Empty}]$$

$$\frac{\vdash \Gamma \qquad \alpha \notin \Gamma}{\vdash \Gamma, \alpha : \kappa}\ [\textsf{U-TExt}] \qquad \frac{\alpha : \kappa \in \Gamma}{\Gamma \vdash \alpha : \kappa}\ [\textsf{U-TVar}] \qquad \frac{\Gamma, \alpha : \kappa \vdash \tau : \kappa'}{\Gamma \vdash \forall \alpha{:}\kappa.\, \tau : \kappa'}\ [\textsf{All}]$$

$$\frac{\Gamma \vdash \tau_1 : \kappa_1 \qquad \Gamma \vdash \tau_2 : \kappa_2}{\Gamma \vdash \tau_1 \xrightarrow{\kappa} \tau_2 : \kappa}\ [\textsf{K-Arr}] \qquad \frac{\Gamma, \alpha : \kappa_1 \vdash \tau : \kappa_2}{\Gamma \vdash \lambda(\alpha : k_1).\, \tau : \kappa_1 \Rightarrow \kappa_2}\ [\textsf{K-Lam}]$$

$$\frac{\Gamma \vdash \tau_1 : \kappa_2 \Rightarrow \kappa_1 \qquad \Gamma \vdash \tau_2 : \kappa_2}{\Gamma \vdash \tau_1 \tau_2 : \kappa_1}\ [\textsf{K-App}]$$

$$\frac{\Gamma_1, \Gamma_2; \Delta; \Xi \vdash e : \tau' \qquad \Gamma_1 \vdash \tau : \star}{\Gamma_1, x : \tau, \Gamma_2; \Delta; \Xi \vdash e : \tau'}\ [\textsf{U-Wkg}]$$

$$\frac{\Gamma; \Delta_1, \Delta_2; \Xi \vdash e : \tau' \qquad \Gamma; \Delta_1 \vdash \tau : \bullet}{\Gamma; \Delta_1, x : \tau, \Delta_2; \Xi \vdash e : \tau'}\ [\textsf{A-Wkg}]$$

$$\cdot \Cup \cdot = \cdot\ [\textsf{AU-Empty}] \qquad \frac{\Delta_1 \Cup \Delta_2 = \Delta \qquad x \notin \Delta}{\Delta_1, x : \tau \Cup \Delta_2 = \Delta, x : \tau}\ [\textsf{AU-Left}]$$

$$\frac{\Delta_1 \Cup \Delta_2 = \Delta \qquad x \notin \Delta}{\Delta_1 \Cup \Delta_2, x : \tau = \Delta, x : \tau}\ [\textsf{AU-Right}] \qquad \cdot \Cup \cdot = \cdot\ [\textsf{LU-Empty}]$$

$$\frac{\Xi_1 \Cup \Xi_2 = \Xi \qquad x \notin \Xi}{\Xi_1, x : \tau \Cup \Xi_2 = \Xi, x : \tau}\ [\textsf{LU-Left}] \qquad \frac{\Xi_1 \Cup \Xi_2 = \Xi \qquad x \notin \Xi}{\Xi_1 \Cup \Xi_2, x : \tau = \Xi, x : \tau}\ [\textsf{LU-Right}]$$

$$\frac{x : \tau \in \Gamma}{\Gamma; \cdot; \cdot \vdash x : \tau}\ [\textsf{U-Var}] \qquad \Gamma; x : \tau; \Xi \vdash x : \tau\ [\textsf{A-Var}]$$

$$\Gamma; \Delta; x : \tau \vdash x : \tau\ [\textsf{L-Var}]$$

$$\frac{\Gamma \vdash \tau : \star \qquad x \notin \Gamma; \Delta; \Xi}{[\Gamma; \Delta; \Xi], x : \tau \Supset \Gamma, x : \tau; \Delta; \Xi}\ [\textsf{U-Ext}]$$

$$\frac{\Gamma \vdash \tau : \bullet \qquad x \notin \Gamma; \Delta; \Xi}{[\Gamma; \Delta; \Xi], x : \tau \Supset \Gamma; \Delta, x : \tau; \Xi}\ [\textsf{A-Ext}]$$

$$\frac{\Gamma \vdash \tau : \circ \qquad x \notin \Gamma; \Delta; \Xi}{[\Gamma; \Delta; \Xi], x : \tau \Supset \Gamma; \Delta; \Xi, x : \tau}\ [\textsf{L-Ext}]$$

$$\frac{[\Gamma; \cdot; \cdot], x : \tau_1 \Supset \Gamma'; \Delta'; \Xi' \qquad \Gamma'; \Delta'; \Xi' \vdash e : \tau_2}{\Gamma \vdash \lambda^\star(x : \tau_1).\, e : \tau_1 \xrightarrow{\star} \tau_2} \text{[U-Lam]}$$

$$\frac{[\Gamma; \Delta; \cdot], x : \tau_1 \Supset \Gamma'; \Delta'; \Xi' \qquad \Gamma'; \Delta'; \Xi' \vdash e : \tau_2}{\Gamma; \Delta \vdash \lambda^\bullet(x : \tau_1).\, e : \tau_1 \xrightarrow{\bullet} \tau_2} \text{[A-Lam]}$$

$$\frac{[\Gamma; \Delta; \Xi], x : \tau_1 \Supset \Gamma'; \Delta'; \Xi' \qquad \Gamma'; \Delta'; \Xi' \vdash e : \tau_2}{\Gamma; \Delta, \Xi \vdash \lambda^\circ(x : \tau_1).\, e : \tau_1 \xrightarrow{\circ} \tau_2} \text{[L-Lam]}$$

$$\frac{\Gamma; \Delta_1; \Xi_1 \vdash e_1 : \tau_1 \xrightarrow{\kappa} \tau_2 \qquad \Gamma; \Delta_2; \Xi_2 \vdash e_2 : \tau_1 \qquad \Delta_1 \Cup \Delta_2 = \Delta \qquad \Xi_1 \Cup \Xi_2 = \Xi}{\Gamma; \Delta; \Xi \vdash e_1 e_2 : \tau_2} \text{[App]}$$

$$\frac{\Gamma, \alpha : \kappa; \Delta; \Xi \vdash v : \tau}{\Gamma; \Delta; \Xi \vdash \Lambda(\alpha : \kappa).\, v : \forall \alpha{:}\kappa.\, \tau} \text{[TLam]}$$

$$\frac{\Gamma; \Delta; \Xi \vdash e : \forall \alpha{:}\kappa.\, \tau' \qquad \Gamma \vdash \tau : \kappa}{\Gamma; \Delta; \Xi \vdash e[\tau] : \tau'[\tau/\alpha]} \text{[TApp]}$$

Evaluation rules:

$$(\lambda^\kappa(x : \tau).\, e)v \mapsto e[v/x] \qquad (\Lambda(\alpha : \kappa).\, v)[\tau] \mapsto v[\tau/\alpha] \qquad \frac{e_1 \mapsto e_1'}{e_1 e_2 \mapsto e_1' e_2}$$

$$\frac{e \mapsto e'}{ve \mapsto ve'} \qquad \frac{e \mapsto e'}{e[\tau] \mapsto e'[\tau]}$$

Existential types may be encoded in the current system. We (admissibly) adopt existential types as primitives.

| | |
|---|---|
| $\tau ::= \ldots \mid \exists \alpha{:}\kappa.\, \tau'$ | types |
| $e ::= \ldots \mid \mathsf{pack}(\tau, e) \mid \mathsf{let}\,\mathsf{pack}(\tau, e) = e_1\,\mathsf{in}\,e_2$ | expressions |
| $v ::= \ldots \mid \mathsf{pack}(\tau, v)$ | values |

$$\frac{\Gamma, \alpha : \kappa \vdash \tau' : \kappa' \qquad \kappa \leq \kappa'}{\Gamma \vdash \exists \alpha{:}\kappa.\, \tau' : \kappa'}\ [\mathsf{Exts}]$$

$$\frac{\Gamma \vdash \tau : \kappa \qquad \Gamma, \alpha : \kappa \vdash \tau' : \star \qquad \kappa \leq \star \qquad \Gamma, \Gamma'; \cdot; \cdot \vdash e : \tau'[\tau/\alpha]}{\Gamma, \Gamma'; \cdot; \cdot \vdash \mathsf{pack}(\tau, e) : \exists \alpha{:}\kappa.\, \tau'}\ []$$

$$\frac{\Gamma \vdash \tau : \kappa \qquad \Gamma, \alpha : \kappa \vdash \tau' : \bullet \qquad \kappa \leq \bullet \qquad \Gamma, \Gamma'; \Delta; \cdot \vdash e : \tau'[\tau/\alpha]}{\Gamma, \Gamma'; \Delta; \cdot \vdash \mathsf{pack}(\tau, e) : \exists \alpha{:}\kappa.\, \tau'}\ []$$

$$\frac{\Gamma \vdash \tau : \kappa \qquad \Gamma, \alpha : \kappa \vdash \tau' : \circ \qquad \kappa \leq \circ \qquad \Gamma, \Gamma'; \Delta; \Xi \vdash e : \tau'[\tau/\alpha]}{\Gamma, \Gamma'; \Delta; \Xi \vdash \mathsf{pack}(\tau, e) : \exists \alpha{:}\kappa.\, \tau'}\ []$$

$$\frac{\begin{array}{c}\Gamma, \Gamma'; \cdot; \cdot \vdash e_1 : \exists \alpha{:}\kappa.\, \tau' \qquad [\Gamma, \alpha : \kappa, \Gamma'; \cdot; \cdot], x : \tau' \vdash e_2 : \tau_2 \\ \Gamma \vdash \tau_2 : \star \qquad \Gamma, \alpha : \kappa \vdash \tau' : \kappa' \qquad \kappa' \leq \star\end{array}}{\Gamma, \Gamma'; \cdot; \cdot \vdash (\mathsf{let\,pack}(\alpha, x) = e_1\ \mathsf{in}\ e_2) : \tau_2}\ []$$

$$\frac{\begin{array}{c}\Gamma, \Gamma'; \Delta; \cdot \vdash e_1 : \exists \alpha{:}\kappa.\, \tau' \qquad [\Gamma, \alpha : \kappa, \Gamma'; \Delta; \cdot], x : \tau' \vdash e_2 : \tau_2 \\ \Gamma \vdash \tau_2 : \bullet \qquad \Gamma, \alpha : \kappa \vdash \tau' : \kappa' \qquad \kappa' \leq \bullet\end{array}}{\Gamma, \Gamma'; \Delta; \cdot \vdash (\mathsf{let\,pack}(\alpha, x) = e_1\ \mathsf{in}\ e_2) : \tau_2}\ []$$

$$\frac{\begin{array}{c}\Gamma, \Gamma'; \Delta; \Xi \vdash e_1 : \exists \alpha{:}\kappa.\, \tau' \qquad [\Gamma, \alpha : \kappa, \Gamma'; \Delta; \Xi], x : \tau' \vdash e_2 : \tau_2 \\ \Gamma \vdash \tau_2 : \circ \qquad \Gamma, \alpha : \kappa \vdash \tau' : \kappa' \qquad \kappa' \leq \circ\end{array}}{\Gamma, \Gamma'; \Delta; \Xi \vdash (\mathsf{let\,pack}(\alpha, x) = e_1\ \mathsf{in}\ e_2) : \tau_2}\ []$$

Evaluation rules:

$$\mathsf{let\,pack}(\alpha, x) = (\mathsf{pack}(\tau, v))\ \mathsf{in}\ e_2 \mapsto e_2[\tau, v/\alpha, x] \qquad \frac{e \mapsto e'}{\mathsf{pack}(\tau, e) \mapsto \mathsf{pack}(\tau, e')}$$

$$\frac{e_1 \mapsto e_1'}{\mathsf{let\,pack}(\alpha, x) = e_1\ \mathsf{in}\ e_2 \mapsto \mathsf{let\,pack}(\alpha, x) = e_1'\ \mathsf{in}\ e_2}$$

## References

[1] Karl Mazurak, Aa Bb, and S. Zdancewic. "Lightweight linear types in System F°". In: Jan. 2010, pp. 77–88. DOI: 10.1145/1708016.1708027.

[2] J. Garrett Morris. "The best of both worlds: linear functional programming without compromise". In: *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*. ICFP'16. ACM, Sept. 2016. DOI: 10.1145/2951913.2951925.

[3] Jesse Tov and Riccardo Pucella. "Practical Affine Types". In: vol. 46. Jan. 2011. DOI: 10.1145/1926385.1926436.