The Next 700 Modal Type Assignment Systems

Andreas Abel
Department of Computer Science and Engineering
Gothenburg University

We exhibit a generic modal type system for simply-typed lambda-calculus that subsumes linear and relevance typing, strictness analysis, variance (positivity) checking, and other modal typing disciplines. By identifying a common structure in these seemingly unrelated non-standard type systems, we hope to gain better understanding and a means to combine several analyses into one. This is work in progress.

Our modal type assignment system is parametrized by a (partially) ordered monoid \((P, \cdot, 1, \leq)\) with a partial, monotone binary operation \(\cdot^+\), and a default element \(p_0 \in P\). Types \(T, U\) include at least a greatest type \(\top\) and function types \(Q \to T\), and form a partial ordering under subtyping \(T \leq T'\) with partial meet \(T \land T'\). Modal types \(Q ::= pT\) support composition \(pQ\) and partial meet \(Q \land Q'\) defined by \(p(qT) = (pq)T\) and \(pT \land qT = (p + q)T\). Subtyping \(pT \leq p'T\) holds if \(p \leq p'\) and \(T \leq T'\).

For typing contexts \(\Gamma, \Delta\), which are total functions from term variables to modal types, modality composition \(p\Gamma\), subsumption \(\Gamma \leq \Delta\), and meet \(\Gamma \land \Delta\) are defined pointwise. Finite contexts \(x_1 : Q_1, \ldots, x_n : Q_n\) are represented as \(\Gamma(x_i) = Q_i\), and \(\Gamma(y) = p_0\top\) for \(y \neq x_i\).

Judgements \(\Gamma \vdash t : T\) and \(\Gamma \vdash t : Q\) are given by the following (linear) typing rules:

\[
\begin{align*}
\text{HYP} & \quad x : pT \vdash x : T \\
\text{ABS} & \quad \frac{p \leq 1 \quad \frac{\Gamma, x : Q \vdash t : T}{\frac{\Gamma \vdash \lambda x t : Q \to T}{\Delta \vdash u : Q} \quad \text{APP}}}{\Gamma \vdash t : Q \to T \quad \frac{\Delta \vdash u \vdash \Gamma}{\frac{\Gamma \vdash t : U}{\frac{\gamma \vdash \Gamma \vdash t : U \quad \frac{\Delta \vdash t : T}{\frac{\gamma \vdash \Delta \vdash t : T}{\gamma \vdash \Gamma \vdash t : U}}}}}} \\
\text{SUB} & \quad \frac{\Gamma \leq \Delta \quad \frac{\Delta \vdash t : T}{\gamma \vdash \Gamma \vdash t : T}}{\gamma \vdash \Gamma \vdash t : U}
\end{align*}
\]

The default modality \(p_0\) controls weakening: We can use the subsumption rule \(\text{SUB}\) with \(\left(\Gamma, x : pT\right) \leq \Gamma\) which holds if \(p \leq p_0\) (as then \(pT \leq p_0\top = \Gamma(x)\)). Meaningful instances of our modal type assignment system abound, here are a few:

1. **Simple typing**: \(P = \{1\}\) with \(1 + 1 = 1\) and \(t\) well-typed if \(\Gamma \vdash t : T \neq \top\).

2. **Quantitative typing**: Take some \(P \subseteq P(\mathbb{N})\) closed under \(p \cdot q = \{nm \mid n \in p, m \in q\}\) and define \(p \leq q\) as \(p \supseteq q\) and \(p + q\) as \(\bigcap\{r \in P \mid r \supseteq \{n + m \mid n \in p, m \in q\}\}\). If \(0 := \{0\} \in P\), it is a zero.

The rule \(\text{MOD}\) has an intuitive reading in quantitative typing: If \(t\) produces a \(T\) from resources \(\Gamma\), we can produce \(p\) times \(T\) from the \(p\)-fold resources \(p\Gamma\). Subsumption \(\text{SUB}\) may allow us to produce less (or the same) from more (or the same) resources. A modal function type \(pU \to T\) requires \(p\)-fold \(U\) to deliver one \(T\).

Instances of quantitative typing include:

(a) **Linear typing**: \([\square]P = \{0, 1\}\) with unit \(\square = \{1\}\) and default \(p_0 = 0\) forbidding weakening with linear variables \(x : \square T\) (as \(\square \not\leq p_0\)). Contraction is also forbidden as \(\square + 1\) is undefined.

(b) **Affine typing**: \(P = \{0, 1\}\) with unit \(\square = \{0, 1\}\), allowing weakening as \(\square \leq p_0 = 0\).

(c) **Relevant typing**: \(P = \{0, 1\}\) with unit \(\square = \mathbb{N} \setminus \{0\}\), allowing contraction as \(1 + 1 = 1\).
(d) **Linear and unrestricted hypotheses:** $P = \{!, \_1\}$ with $\_1 = \{1\}$ and $p_0 = ! = \_N$. Allows weakening and contraction for $x : ! T$.

(e) **Strictness typing:** $P = \{l, s\}$ with $l = \_N$ and unit strict $s = \_N \setminus \emptyset$.

We cannot weaken with strict variables. As $p + q = s$ iff $p = s$ or $q = s$, one strict occurrence of a variable $x$ suffices to classify a function $\lambda x : T : sT \rightarrow T'$ as strict, whereas a function is lazy only if all occurrences of parameter $x$ are lazy.

3. **Variance (positivity):** $P = \{\emptyset, +, -, \pm\} = P\{+1, -1\}$ with unit $+ = \{+1\}$ denoting positive occurrence, $- = \{-1\}$ negative occurrence, $\pm = \{+1, -1\}$ mixed occurrence, and $p_0 = \emptyset$ no occurrence. With $p \leq q$ iff $p \supseteq q$ and $pq = \{ij \mid i \in p, j \in q\}$ and $p + q = p \cup q$ we obtain **variance typing** aka **positivity checking** for type-level lambda calculi [1].

We can go further and give up the distinction between types and modal types, leading to the types $T, U ::= \top \mid U \rightarrow T \mid pT \mid \ldots$ quotiented by $p(qT) = (pq)T$. This makes modal types first class, and we can simplify the hypothesis rule to

$\begin{array}{c}
x : T \vdash x : T \\
\end{array}$

HYP.

Thus, we subsume further type systems:

1. **Linear typing with exponential:** As [2], but now $! T$ is a valid type.

2. **Nakano’s modality for recursion** [3]: Basic modalities are later $\triangleright$ and always $\Box$ with $\Box \cdot p = \Box$, generating the modalities $P = \{\triangleright^n, \Box^n \mid n \in \_N\}$ with unit $1 = \triangleright^0$ and partial order $\triangleright^n \leq \triangleright^m$ for $k \leq l \leq m$. Since $x : U \rightarrow T$, $y : U \vdash xy : T$ entails $x : \triangleright(U \rightarrow T), y : \triangleright U \vdash xy : \triangleright T$ by MOD, idiomatic application $\lambda x \lambda y.xy : \triangleright(U \rightarrow T) \rightarrow \triangleright U \rightarrow \triangleright T$ is definable.

Acknowledgments. Thanks to the anonymous referees, who helped improving the quality of this abstract through their feedback. This work was supported by Vetenskapsrådet through the project **Termination Certificates for Dependently-Typed Programs and Proofs via Refinement Types**.

References


