Principal Typings Demystified

what they are,
why you want them,
and why your type system doesn't have them

Joe Wells

Heriot-Watt University

Computing & Elec. Eng. Dept.

`http://www.cee.hw.ac.uk/~jbw/`

A Perspective Change via New Notation

Consider the λ-term $P = (\lambda yx.x)$.

Traditional presentations of the simply typed lambda calculus (`STLC`) can derive this judgement:

$$\emptyset \vdash P : \beta \to \alpha \to \alpha$$

Instead, we will write such statements like this:

$$P : \underbrace{(\emptyset \vdash \beta \to \alpha \to \alpha)}_{\textit{typing}}$$

This perspective change will help prevent wrong thinking.

We say the typing $t_1 = (\emptyset \vdash \beta \to \alpha \to \alpha)$ can be *assigned* to P.

More Typings for the Example (1)

(Memory from previous overheads)

$$P = (\lambda yx.x) \qquad t_1 = (\emptyset \vdash \beta \to \alpha \to \alpha)$$

The term P can also be assigned the typing

$$t_2 = (\emptyset \vdash \gamma \to \gamma \to \gamma)$$

which can be obtained from t_1 by a *substitution*:

$$S_2 \qquad = \{(\alpha \mapsto \gamma), (\beta \mapsto \gamma)\}$$

$$S_2(t_1) = t_2$$

More Typings for the Example (2)

(Memory from previous overheads)

$$P = (\lambda yx.x) \qquad t_1 = (\emptyset \vdash \beta \to \alpha \to \alpha)$$

Another typing for $P = (\lambda yx.x)$ is

$$t_3 = (\{z : \delta \to \delta\} \vdash (\delta \to \delta) \to \delta \to \delta)$$

which can be obtained from t_1 by a substitution and a *weakening*:

$$\begin{aligned} S_3 &= \{(\alpha \mapsto \delta), (\beta \mapsto \delta \to \delta)\} \\ W_3 &= \{z : \delta \to \delta\} \end{aligned}$$

$$W_3(S_3(t_1)) = t_3$$

Introducing Principal Typings

- In fact, it turns out that every typing t that can be assigned to P in STLC can be obtained from t_1 by applying some substitution S and some weakening W.

  So t_1 is called a *principal typing* for P.

  If $t_2 = W(S(t_1))$, then t_2 is often called an *instance* of t_1.

- STLC has principal typings, i.e., every typable term has one (see Hindley [1997] for references).

- In general, a principal typing in a system $\mathbf{S}$ for a term M is a typing assignable to M which *somehow* represents all other typings in $\mathbf{S}$ that can be assigned to M.

  Part of this talk is about clarifying the "*somehow*".

- Principal typings turn out to be very useful when finding types for untyped programs, as the following example shows.

An Example Problem of Finding Types (1)

Consider these λ-terms:

$$
\begin{aligned}
M &= \lambda z.NP \\
N &= \lambda w.w(wz) \\
P &= \lambda yx.x
\end{aligned}
$$

A bottom-up analysis algorithm $\texttt{Inf}$ for $\texttt{STLC}$ would do this on M:

$$
\texttt{Inf}(M) \quad = \quad \begin{array}{c} \texttt{Cmb}_\lambda(z) \\ | \\ \texttt{Inf}(NP) \end{array} \quad = \quad \begin{array}{c} \texttt{Cmb}_\lambda(z) \\ | \\ \texttt{Cmb}_@ \\ \diagup \quad \diagdown \\ \texttt{Inf}(N) \qquad \texttt{Inf}(P) \end{array}
$$

The subalgorithms $\texttt{Cmb}_\lambda$ and $\texttt{Cmb}_@$ are used to combine the results from recursively processing the subterms.

An Example Problem of Finding Types (2)

(Memory from previous overheads)

$$M = \lambda z.NP$$
$$N = \lambda w.w(wz)$$
$$P = \lambda yx.x$$

$$\mathtt{Inf}(M) = \left(\begin{array}{c} \mathtt{Cmb}_\lambda(z) \\ | \\ \mathtt{Cmb}_@ \\ \diagup \quad \diagdown \\ \mathtt{Inf}(N) \qquad \mathtt{Inf}(P) \end{array} \right)$$

Suppose $\mathtt{Inf}$ is designed by a naïve person like me and $\mathtt{Inf}(P) = t_2 = (\emptyset \vdash \gamma \to \gamma \to \gamma)$. What happens?

Typing NP needs a typing for P like $t_4 = (\emptyset \vdash (\delta \to \delta) \to \delta \to \delta)$.

Can $\mathtt{Cmb}_@$ figure out from t_2 that t_4 is also assignable to P?

An Example Problem of Finding Types (3)

(Memory from previous overheads)

$$t_2 = (\emptyset \vdash \gamma \to \gamma \to \gamma)$$
$$t_4 = (\emptyset \vdash (\delta \to \delta) \to \delta \to \delta)$$

If all we know about a term Q in STLC is that it can be assigned t_2, can we conclude that we can also assign it t_4?

No, because here is a term which can be assigned t_2 but not t_4:

$$Q = \lambda yx.(\lambda z.x)(\lambda w.wx(wyx))$$

The types of x and y are forced to be the same by the applications (wx) and (wy). The term Q differs from P only by the addition of some constraining subterms which have no computational effect.

Principal Typings Solve the Example Problem

(Memory from previous overheads)

$$N = \lambda w.w(wz)$$
$$P = \lambda yx.x$$

$$t_1 = (\emptyset \vdash \beta \to \alpha \to \alpha)$$
$$t_2 = (\emptyset \vdash \gamma \to \gamma \to \gamma)$$
$$t_4 = (\emptyset \vdash (\delta \to \delta) \to \delta \to \delta)$$

$$\mathtt{Inf}(NP) = \left(\begin{array}{c} \mathtt{Cmb}_{@} \\ \diagup \quad \diagdown \\ \mathtt{Inf}(N)\,\mathtt{Inf}(P) \end{array} \right)$$

Suppose instead of $\mathtt{Inf}(P) = t_2$ we make $\mathtt{Inf}(P) = t_1$, the principal typing of P. Can $\mathtt{Cmb}_{@}$ deduce that it can legally use t_4, the typing needed for NP?

By principality, $t_4 = W(S(t_1))$ for some S and W. Is this enough?

Yes, because for any substitution or weakening X, if $M' : t$ is derivable in $\mathtt{STLC}$, then $M' : X(t)$ is also derivable.

PT Example with Intersection Types (1)

Consider this λ-term:

$$M = \lambda xyz.x(yz)$$

In many systems with intersection types, the principal typing of M is:

$$t_1 = (\emptyset \vdash (\alpha \to \beta) \to (\gamma \to \alpha) \to \gamma \to \beta)$$

Another:

$$t_2 = (\emptyset \vdash ((\alpha_1 \cap \alpha_2) \to (\delta \times \delta)) \to ((\gamma_1 \to \alpha_1) \cap (\gamma_2 \to \alpha_2)) \to (\gamma_1 \cap \gamma_2) \to (\delta \times \delta))$$

There is an *expansion* E and a substitution S such that $t_2 = S(E(t_1))$ (see van Bakel [1995] for references).

Other operations used in obtaining other typings from the principal typing in systems with intersection types include *coverings* and *liftings*.

PT Example with Intersection Types (2)

(Memory from previous overheads)

$$
\begin{aligned}
M &= \lambda xyz.x(yz) \\
t_1 &= (\emptyset \vdash (\alpha \to \beta) \to (\gamma \to \alpha) \to \gamma \to \beta) \\
t_2 &= (\emptyset \vdash \quad ((\alpha_1 \cap \alpha_2) \to (\delta \times \delta)) \\
&\qquad \to ((\gamma_1 \to \alpha_1) \cap (\gamma_2 \to \alpha_2)) \\
&\qquad \to (\gamma_1 \cap \gamma_2) \to (\delta \times \delta))
\end{aligned}
$$

The example may be clearer in a system with expansion variables [Kfoury and Wells, 1999]:

$$
\begin{aligned}
M' &= \lambda xyz.x(G(y(Fz))) \\
t_1' &= (\emptyset \vdash ((G\alpha) \to \beta) \to (G((F\gamma) \to \alpha)) \to (G(F\gamma)) \to \beta) \\
S' &= \{F_1 \mapsto \Box, F_2 \mapsto \Box, G \mapsto \Box \cap \Box, \beta \mapsto \delta \times \delta\}
\end{aligned}
$$

Then $t_2 = S'(t_1')$. Expansion and substitution are integrated.

Confusion in the Community

There is enormous confusion in the research community about principality. Examples include:

- Claims that the Hindley/Milner (HM) system [Damas and Milner, 1982] does not have principal typings using the syntactic notion of *instance* via substitution. These are erroneous because the various systems of intersection types have principal typings, but not via just substitution.
- A recent workshop paper titled “ML has Principal Typings”. This talk will later prove the opposite.
- A paper with “principal” in the title currently submitted by a good researcher to a major conference with false claims:
  - Claim: All typings for a term M in STLC are obtained from its principal typing using *only substitution*.
  - Claim: In HM when $M : (\Gamma \vdash \tau)$, the type τ is principal for M iff τ is a *subtype* of any type τ' such that $M : (\Gamma \vdash \tau')$.

Technical Definitions

Let the metavariable $\mathbf{S}$ range over type systems.

Let $\mathbf{S} \rhd M : t$ mean $M : t$ is derivable by the rules of $\mathbf{S}$, in which case the system $\mathbf{S}$ *assigns* the typing t to the term M.

$\mathtt{Terms}_{\mathbf{S}}(t) = \{\, M \mid \mathbf{S} \rhd M : t \,\}$.

$\mathtt{Typings}_{\mathbf{S}}(M) = \{\, t \mid \mathbf{S} \rhd M : t \,\}$.

Now a new ordering on typings is defined. Let $t_1 \leq_{\mathbf{S}} t_2$ mean $\mathtt{Terms}_{\mathbf{S}}(t_1) \subseteq \mathtt{Terms}_{\mathbf{S}}(t_2)$, in which case the typing t_1 is *at least as strong as* t_2.

Observations on Systems with PTs

The following holds for each system $\mathbf{S}$ with principal typings that I know.

In $\mathbf{S}$, each typable term M can be assigned a typing t which is *principal for M* in the sense that for every other typing t' assignable to M, there exist operations $O_1, \ldots, O_n$ such that

- $t' = O_n(\cdots(O_1(t))\cdots)$, and
- for any term N, if $\mathbf{S} \rhd N : t$, then $\mathbf{S} \rhd N : t_i$ where $t_i = O_i(\cdots(O_1(t))\cdots)$ for $1 \leq i \leq n$.

For some (but not all) systems a stronger and simpler statement about the operations holds.

Observation: If t is principal for M, then $t \leq_{\mathbf{S}} t'$ for every $t' \in \mathtt{Typings}_{\mathbf{S}}(M)$.

A New System-Independent Definition

A typing t is *principal for M* in system $\mathbf{S}$ iff

- $\mathbf{S} \rhd M : t$ and
- $\mathbf{S} \rhd M : t'$ implies $t \leq_{\mathbf{S}} t'$.

Is this the right definition?

- All typings called principal by earlier definitions are principal by this definition.
- $t \leq_{\mathbf{S}} t'$ exactly when an analysis algorithm can correctly replace a typing t inferred for a term M by t' *without inspecting M again*.
- If t is principal for M, then t is minimal in $\mathtt{Typings}_{\mathbf{S}}(M)$, so an analysis algorithm possessing t has the mathematical potential to infer any typing for M in a larger context.

Remarks on Principal Typings for Analysis

- If a type analysis algorithm possesses a principal typing in system $\mathbf{S}$ for a term M, then it can never gain any more information by reanalyzing M in system $\mathbf{S}$.
- Principal typings allow *compositional* type analysis, where analyzing a fragment uses only the *results* for its subfragments, which can be analyzed independently in any order.
- Compositional analysis *helps* with separate compilation and modularity and with making a complete/terminating analysis algorithm.

Principality in the Hindley/Milner System?

There is much confusion about principality in HM. Here is the property that HM actually has.

- A typing t is a *Γ-typing* iff
  - $t = (\Gamma' \cup S(\Gamma) \vdash \tau)$, and
  - *Γ', S, and τ do not mention "$\forall$"*.
- A term M is *Γ-typable* iff $\mathtt{HM} \rhd M : t$ for some Γ-typing t.
- A typing t is *Γ-principal for M* iff
  - $\mathtt{HM} \rhd M : t$,
  - t is a Γ-typing, and
  - if t' is a Γ-typing for M, then $t' = W(S(t))$ for some weakening W and substitution S (W and S can not mention "$\forall$" by the condition above).

Principality for HM: If M is Γ-typable, there is a Γ-principal t for M.

HM Does Not Have Principal Typings (1)

A counterexample is the term (xx).

Suppose $\mathtt{HM} \rhd (xx) : t$ where $t = (\Gamma \vdash \tau)$. It will be proven that t is not principal for (xx). The strategy is to find t' and M such that:

$$\mathtt{HM} \rhd (xx) : t'$$
$$\mathtt{HM} \rhd M : t$$
$$\mathtt{HM} \not\rhd M : t'$$

First the typing t' of the counterexample is defined. Let $(\forall \vec{\alpha}.\sigma) = \Gamma(x)$ where σ does not mention "$\forall$". Let $\sigma' = \forall.(\sigma \rightarrow \beta)$ for fresh β. Let $\Gamma' = \{x{:}\sigma'\}$. Choose τ' arbitrarily. Let $t' = (\Gamma' \vdash \tau')$.

Easy: $\mathtt{HM} \rhd (xx) : t'$.

So t' is another typing for (xx), but it will be seen that $t \not\leq_{\mathtt{HM}} t'$.

HM Does Not Have Principal Typings (2)

Some useful auxiliary machinery:

- Let $\mathtt{K} = (\lambda xy.x)$.
- To force identical derived result types for two subterms: Let $\mathtt{Unify}(N, N') = (\lambda xy.y(xN)(xN'))$ for fresh x, y.
- To measure the length of the leftmost path in a type viewed as a tree: Let $\mathtt{LLen}(\alpha) = 0$, $\mathtt{LLen}(\rho \to \rho') = 1 + \mathtt{LLen}(\rho)$, and $\mathtt{LLen}(\forall \alpha.\rho) = \mathtt{LLen}(\rho)$.
- To have a term whose derived result type must have a fixed $\mathtt{LLen}$ value: Let y_1, y_2, ... be fresh. Let $\mathtt{FixLLen}(0) = \mathtt{c}$ for base type constant $\mathtt{c}$. Let $\mathtt{FixLLen}(i+1) = (\lambda x.\mathtt{K}\, y_{i+1}\, \mathtt{Unify}(x, \mathtt{FixLLen}(i)))$ for fresh x.

  Easy: If $\mathtt{HM} \rhd \mathtt{FixLLen}(i) : (\hat{\Gamma} \vdash \hat{\tau})$, then $\mathtt{LLen}(\hat{\tau}) = i$.

HM Does Not Have Principal Typings (3)

(Memory from previous overheads)

$$\begin{array}{lll} \mathtt{HM} \rhd (xx) : t & t = (\Gamma \vdash \tau) & \Gamma(x) = \forall \vec{\alpha}.\sigma \\ \mathtt{HM} \rhd (xx) : t' & t' = (\Gamma' \vdash \tau') & \Gamma' = \{x{:}\sigma'\} \\ & & \sigma' = \forall.(\sigma \to \beta) \end{array}$$

Now the term M of the counterexample is defined. Let $k = \mathtt{LLen}(\Gamma(x))$. Let $M = \mathtt{K}(xx)(\lambda y_1 \ldots y_k.x\,\mathtt{FixLLen}(k-1))$.

Not hard: $\mathtt{HM} \rhd M : t$.

The finale: $\mathtt{HM} \not\rhd M : t'$, because $\mathtt{LLen}(\Gamma'(x)) = k+1$ and can not shrink by instantiation to the needed k to allow typing the subterm $(x\,\mathtt{FixLLen}(k-1))$.

Therefore, $t \not\leq_{\mathtt{HM}} t'$ and t is not a principal typing in $\mathtt{HM}$ for (xx). Because t is arbitrary, (xx) has no $\mathtt{HM}$-principal typing.

Implications of HM's Lack of PTs

It is not wrong that HM does not have principal typings. It just means an HM analysis algorithm must do one of these:

- Be incomplete (not finding typings for some typable terms).
- Be noncompositional (not strictly bottom-up). For example, the $\mathcal{W}$ algorithm for HM [Damas and Milner, 1982] is noncompositional because for $(\texttt{let}\ x = M\ \texttt{in}\ N)$ it first analyzes M and then uses the result in analyzing N.
- Not use HM typings for intermediate results. E.g., the principal typing of (xx) in the Chap. 1 system of Damas [1985]:

  $$(\{x{:}\alpha, x{:}\alpha \to \beta\} \vdash \beta)$$

  This is essentially intersection types, i.e.:

  $$(\{x{:}\alpha \cap (\alpha \to \beta)\} \vdash \beta)$$

  Essentially the same was done by Shao and Appel [1993] and Bernstein and Stark [1995].

System F Does Not Have Principal Typings

- System $\mathtt{F}$ [Girard, 1972; Reynolds, 1974] can be presented as a Curry-style system which assigns typings to pure λ-terms [Leivant, 1983].
- As for $\mathtt{HM}$, I have proven that (xx) has no principal typing.
- As for $\mathtt{HM}$, the proof works by showing for each $t \in \mathtt{Typings_F}(xx)$ there is another $t' \in \mathtt{Typings_F}(xx)$ and a term $M \in \mathtt{Terms_F}(t) \setminus \mathtt{Terms_F}(t')$, thus proving t is not principal.
- Unlike for $\mathtt{HM}$, the term M makes one dimension of a type too small for use with t' without relying on a ground-type constant but instead uses methods of Wells [1999].

Conclusions

- A principal typing for M in $\mathbf{S}$ is a typing t which represents all other typings for M in $\mathbf{S}$.

  This is now characterized precisely by requiring that $t \leq_{\mathbf{S}} t'$ for every other typing t' assignable to M.

- The existence of principal typings for a system $\mathbf{S}$ allows the possibility of an analysis algorithm which is complete, is compositional, and uses $\mathbf{S}$-typings for intermediate results.

- Unfortunately, neither `HM` nor `F` have principal typings. Because these systems are commonly used as the basis of other type systems, this has wide implications.

- If principal typings are needed, one can often obtain them by adding intersection types.

References

Karen L. Bernstein and Eugene W. Stark. Debugging type errors (full version). Technical report, State University of New York, Stony Brook, November 1995.

L. Damas and Robin Milner. Principal type schemes for functional programs. In *Conf. Rec. 9th Ann. ACM Symp. Princ. of Prog. Langs.*, pages 207–212, 1982.

Luis Manuel Martins Damas. *Type assignment in Programming Languages*. PhD thesis, University of Edinburgh, Edinburgh, Scotland, April 1985.

J[ean]-Y[ves] Girard. *Interprétation Fonctionnelle et Elimination des Coupures de l'Arithmétique d'Ordre Supérieur*. Thèse d'Etat, Université de Paris VII, 1972.

J. Roger Hindley. *Basic Simple Type Theory*, volume 42 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1997.

Assaf J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In *Conf. Rec. POPL '99: 26th ACM Symp. Princ. of Prog. Langs.*, pages 161–174, 1999. ISBN 1-58113-095-3.

Daniel Leivant. Polymorphic type inference. In *Conf. Rec. 10th*

Ann. ACM Symp. Princ. of Prog. Langs., pages 88–98, 1983. ISBN 0-89791-090-7.

J. C. Reynolds. Towards a theory of type structure. In *Colloque sur la Programmation*, volume 19 of *LNCS*, pages 408–425, Paris, France, 1974. Springer-Verlag.

Zhong Shao and Andrew Appel. Smartest recompilation. In *Conf. Rec. 20th Ann. ACM Symp. Princ. of Prog. Langs.*, 1993.

Steffen J. van Bakel. Intersection type assignment systems. *Theoret. Comp. Sci.*, 151(2):385–435, 27 November 1995.

J. B. Wells. Typability and type checking in the second-order λ-calculus are equivalent and undecidable. In *Proc. 9th Ann. IEEE Symp. Logic in Comp. Sci.*, pages 176–185, 1994. ISBN 0-8186-6310-3. Superseded by Wells [1999].

J. B. Wells. Typability and type checking in System F are equivalent and undecidable. *Ann. Pure Appl. Logic*, 98(1–3):111–156, 1999. Supersedes Wells [1994].