Branching Types

squashing typing derivations in systems with intersection types

Joe Wells and Christian Haack

Heriot-Watt University

Computing & Elec. Eng. Dept.

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

http://www.cee.hw.ac.uk/~haack/

Overview

- Context of research (The Church Project)
- Intersection types
- The problem: disjoint subderivations for a subterm
- Our solution: branching types
- Related and future work

The Church Project

- Investigates foundations, design principles, and implementation techniques for programming languages. Named in memory of Alonzo Church.
- Members most relevant to this research:

Boston Coll.	Bob Muller
Boston Univ.	Assaf Kfoury, Gang Chen, Geoff Washburn, Ian Westmacott
Harvard Univ.	Allyn Dimock, Glenn Holloway
Heriot-Watt Univ.	Joe Wells, Torben Amtoft, Christian Haack
Kansas State Univ.	Anindya Banerjee
Wellesley Coll.	Lyn Turbak

- Web address: `http://www.church-project.org/`

The Church Project: Related Work

- λ^{CIL}: explicitly typed calculus, intersection/union types [Wells et al., 1997, 2002].
- A polyvariant/polymorphic type/flow analysis algorithm using rank-2 intersection types [Banerjee, 1997].
- Strongly typed representation transformation in an SML compiler supporting standard, selective, and lightweight (limited forms) closure conversion and flow-directed inlining [Dimock, Muller, Turbak, and Wells, 1997; Dimock, Westmacott, Muller, Turbak, Wells, and Considine, 2001b; Dimock, Westmacott, Muller, Turbak, and Wells, 2001a].
- Principal typing algorithm for intersection types [Kfoury and Wells, 1999].
- Exact complexity of rank-k ACI intersection type inference [Kfoury, Mairson, Turbak, and Wells, 1999].

Overview

- Context of research (The Church Project)
- Intersection types
- The problem: disjoint subderivations for a subterm
- Our solution: branching types
- Related and future work

Intersection Types

- Type polymorphism by *listing* usage types [Coppo et al., 1980; Barbanera et al., 1995].
- Why "intersection"? If $[\![\sigma]\!]$ and $[\![\tau]\!]$ are program fragment sets, then $[\![\sigma \cap \tau]\!] = [\![\sigma]\!] \cap [\![\tau]\!]$.
- Example comparing intersection and $\forall$-quantified types:

  intersection types: $(\texttt{fn x} \Rightarrow \texttt{x})^{(\texttt{int}\to\texttt{int})\cap(\texttt{bool}\to\texttt{bool})}$

  $\forall$-quantified types: $(\texttt{fn x} \Rightarrow \texttt{x})^{\forall\alpha.(\alpha\to\alpha)}$

  Example is semantically like $\forall\alpha \in \{\texttt{int}, \texttt{bool}\}.\alpha \to \alpha$, but has significant practical differences.

- Benefits of intersection types:
  - Type information closer to actual behavior.
  - More programs are typable.

Polymorphism with Intersection Types

$$\texttt{val swap}^{\left(\begin{smallmatrix}(\texttt{int}\times\texttt{bool})\to(\texttt{bool}\times\texttt{int})\\ \cap(\texttt{real}\times\texttt{real})\to(\texttt{real}\times\texttt{real})\end{smallmatrix}\right)} = (\texttt{fn}\ (\texttt{x}^{\texttt{int}}_{\texttt{real}}, \texttt{y}^{\texttt{bool}}_{\texttt{real}}) \Rightarrow (\texttt{y}^{\texttt{bool}}_{\texttt{real}}, \texttt{x}^{\texttt{int}}_{\texttt{real}}));$$

$$\texttt{val pair1}^{\texttt{int}\times\texttt{bool}} = (1, \texttt{true});$$

$$\texttt{val pair2}^{\texttt{real}\times\texttt{real}} = (2.7, 9.9);$$

$$(\texttt{swap}^{(\texttt{int}\times\texttt{bool})\to(\texttt{bool}\times\texttt{int})}\ \texttt{pair1},$$
$$\texttt{swap}^{(\texttt{real}\times\texttt{real})\to(\texttt{real}\times\texttt{real})}\ \texttt{pair2});$$

- All types discovered automatically [van Bakel, 1993; Jim, 1996; Kfoury and Wells, 1999; Ronchi Della Rocca, 1988].
- Exposes usage types throughout.
- Structuring of independent type analyses is the problem.

Typability for Various Systems

F: System F.

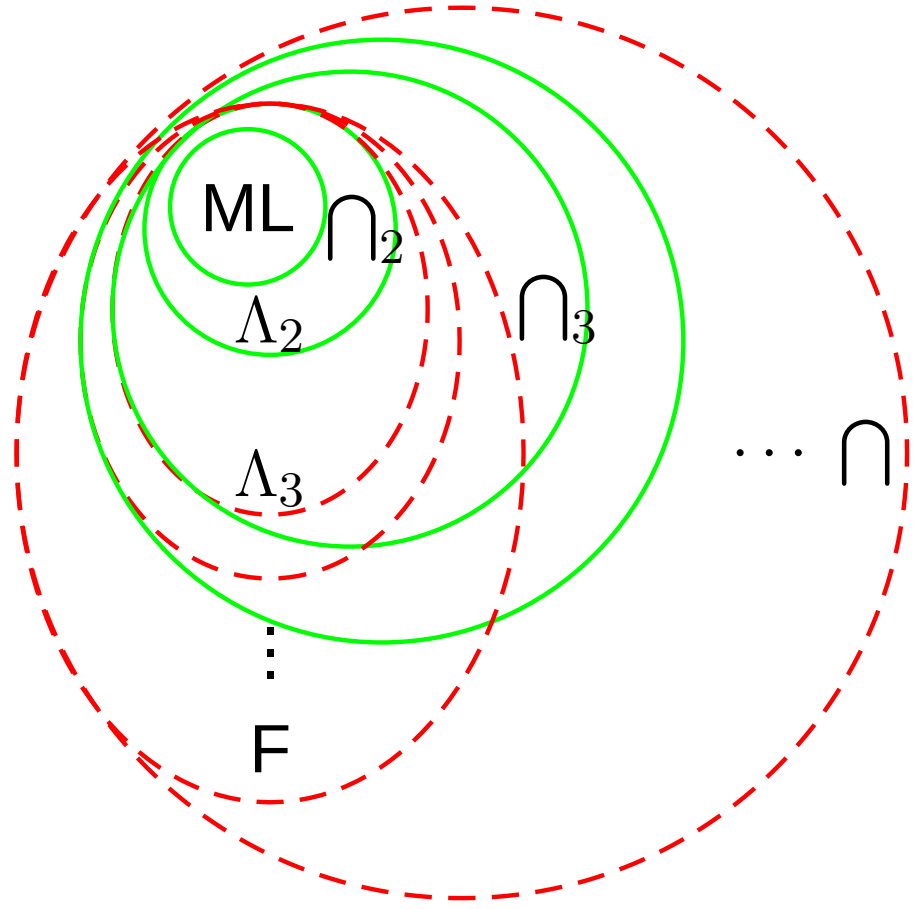
Λ_k: rank-k System F.

$\bigcap$: intersection types.

$\bigcap_k$: rank-k of $\bigcap$.

Decidable.

Undecidable.



(Asymptotic complexity now known [Kfoury, Mairson, Turbak, and Wells, 1999].)

Flexibility of Intersection Types

```
fun self_apply2 z ⇒ (z z) z;
fun apply f x ⇒ f x;
fun reverse_apply y g ⇒ g y;
fun id w ⇒ w;
(self_apply2 apply not true,
 self_apply2 reverse_apply id false not);
```

- The example *safely* computes $(\texttt{false}, \texttt{true})$.
- Urzyczyn [1997] proved this example is not typable in F_ω, considered the most powerful type system with universal quantifiers.
- The example is typable in the rank-3 restriction of intersection types.

Overview

- Context of research (The Church Project)
- Intersection types
- The problem: disjoint subderivations for a subterm
- Our solution: branching types
- Related and future work

Trouble with Intersection Introduction

The intersection-introduction rule:

$$\frac{E \vdash M : \sigma; \quad E \vdash M : \tau}{E \vdash M : \sigma \wedge \tau} \ (\wedge\text{-intro})$$

Notice: same proof term for premises and conclusion, no syntax is introduced. The usual type annotation approach fails immediately, e.g.:

$$\frac{E \vdash (\lambda x{:}\sigma.\, x) : (\sigma \to \sigma); \quad E \vdash (\lambda x{:}\tau.\, x) : (\tau \to \tau)}{E \vdash (\lambda x{:}\boxed{???}.\, x) : (\sigma \to \sigma) \wedge (\tau \to \tau)}$$

Where $\boxed{???}$ appears, what should be written?

Earlier Approaches

- Reynold's Forsythe [Reynolds, 1996]:

$$(\lambda x{:}\sigma_1 | \cdots | \sigma_n.\, M) : (\sigma_1 \to \tau) \wedge \cdots \wedge (\sigma_n \to \tau)$$

However, it can not make typed version of K have type τ_K:

$$\begin{aligned} K &= (\lambda x.\lambda y.x) \\ \tau_K &= (\sigma \to (\sigma \to \sigma)) \wedge (\tau \to (\tau \to \tau)) \end{aligned}$$

- Pierce [1991] gives a typed version of K the right type:

$$(\mathbf{for}\ \alpha \in \{\sigma, \tau\}.\lambda x{:}\alpha.\, \lambda y{:}\alpha.\, x) : \tau_K$$

However, it can not give the term M_f the type τ_f:

$$\begin{aligned} M_f &= \lambda x.\lambda y.\lambda z.(xy, xz) \\ \tau_f &= \begin{pmatrix} (((\alpha \to \delta) \wedge (\beta \to \epsilon)) \to \alpha \to \beta \to (\delta \times \epsilon)) \\ \wedge\, ((\gamma \to \gamma) \to \gamma \to \gamma \to (\gamma \times \gamma)) \end{pmatrix} \end{aligned}$$

Extending An Earlier Approach

- We could go beyond Pierce's approach to make a typed version of M_f with the right type:

$$\begin{array}{l}\textbf{for}\ \{[\theta \mapsto \alpha, \kappa \mapsto \beta, \eta \mapsto \delta, \nu \mapsto \epsilon], \\ \qquad [\theta \mapsto \gamma, \kappa \mapsto \gamma, \eta \mapsto \gamma, \nu \mapsto \gamma]\ \}. \\ \lambda x : (\theta \to \eta) \wedge (\kappa \to \nu)\,.\,\lambda y : \theta\,.\,\lambda z : \kappa\,.\,(xy, xz)\end{array}$$

- This is still unsatisfactory:
  - Type information for a subterm is not stored at that location.
  - In general, well-typedness of a subterm can not be determined independently of the larger term it is in.
  - No Curry/Howard correspondence (where terms are proofs).

Our Earlier λ^{CIL} Approach

- A typed version of M_f with type τ_f in λ^{CIL} [Wells et al., 1997]:

$$\bigwedge(\lambda x : (\alpha \to \delta) \wedge (\beta \to \epsilon) \,.\, \lambda y : \alpha \,.\, \lambda z : \beta \,.\, ((\pi_1^{\wedge} x)y, (\pi_2^{\wedge} x)z)$$
$$\lambda x : \gamma \to \gamma \,.\, \lambda y : \gamma \,.\, \lambda z : \gamma \,.\, (xy, xz))$$

- The tree structures of the λ^{CIL} term and the usual typing derivation with intersection types are the same!
- We have worked out the rules for correctly carrying out the usual λ-calculus manipulations, but these rules are very tedious to implement.

Overview

- Context of research (The Church Project)
- Intersection types
- The problem: disjoint subderivations for a subterm
- Our solution: branching types
- Related and future work

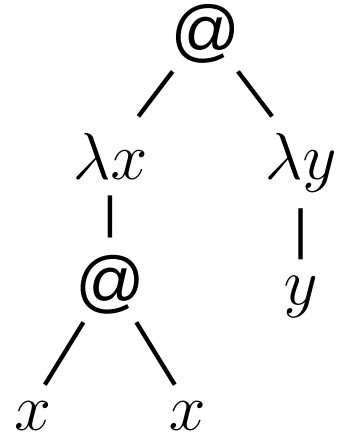
The System λ^{B}

Our system λ^{B} will be presented mainly by examples.

- First, an example typing will be worked in both λ^{CIL} and λ^{B} style. Don't worry if you don't understand parts.
- Then, examples will cover the topics of *kinds*, *type selection* and its *parameters* and *arguments*, *branching types*, and *type equivalence*.
- Then, the syntax and typing rules will be given, mainly to point out how most of the rules are the usual ones.
- Unfortunately, I will skip the concepts of *expansion*, *inner* and *outer* kinds of type selection parameters, and the precise definitions of *type erasure* and *trivial* type selection parameters and arguments.
- Finally, some theorems will be stated.

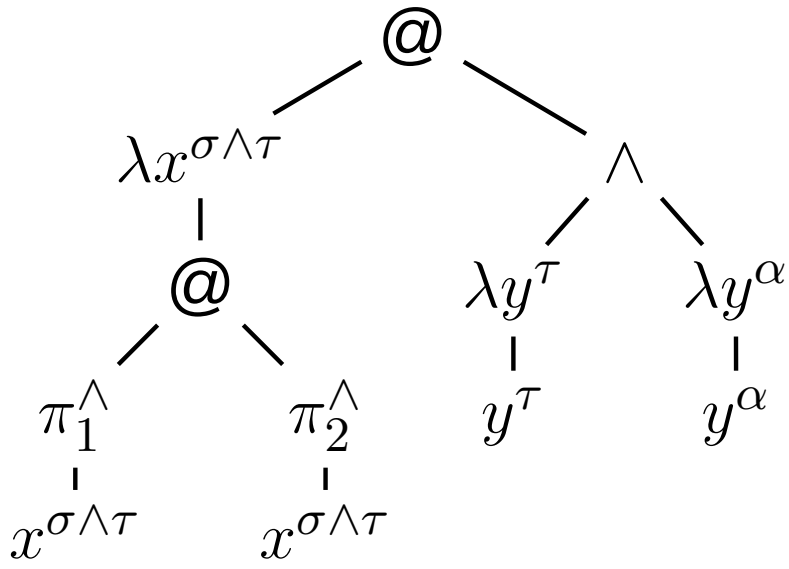
Example: An Untyped Term

$$(\lambda x.x\ x)\ (\lambda y.y)$$



Example: A Corresponding λ^{CIL}-term

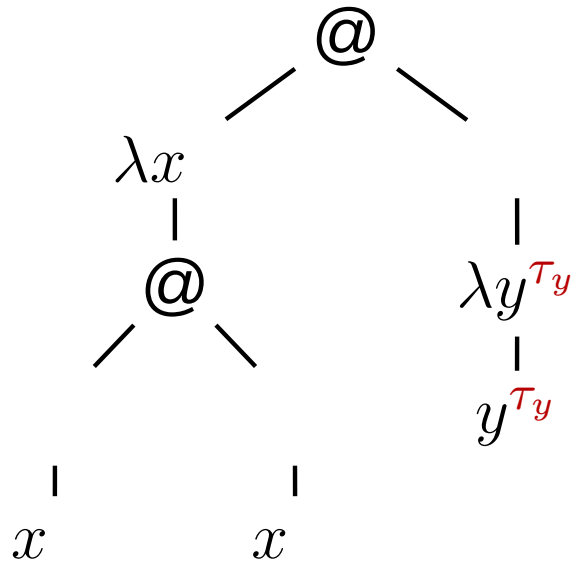
$$(\lambda x.x\ x)\ (\lambda y.y)$$



where $\tau = \alpha \to \alpha$ and $\sigma = \tau \to \tau$

Example: A Corresponding λ^{B}-term

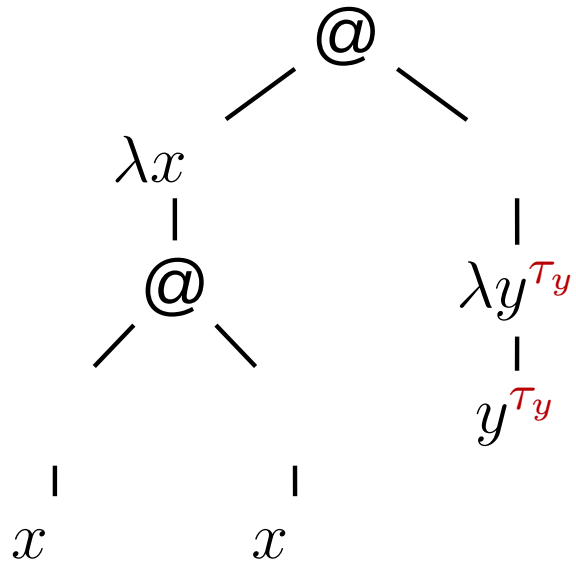
$$(\lambda x.x\ x)\ (\lambda y.y)$$



$$\begin{aligned} \tau &= \alpha \to \alpha \\ \sigma &= \tau \to \tau \\ \tau_y &= \{i = \tau,\ j = \alpha\} \end{aligned}$$

Example: A Corresponding λ^{B}-term

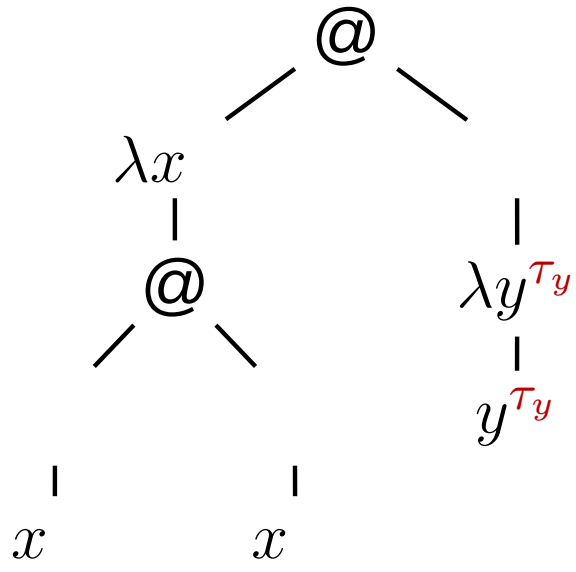
$$(\lambda x.x\ x)\ (\lambda y.y)$$



$$\begin{array}{rcl} \tau & = & \alpha \to \alpha \\ \sigma & = & \tau \to \tau \\ \tau_y & = & \{i = \tau,\ j = \alpha\} \\ \lambda y^{\tau_y}.y^{\tau_y} & : & \tau_y \to \tau_y \end{array}$$

Example: A Corresponding λ^{B}-term

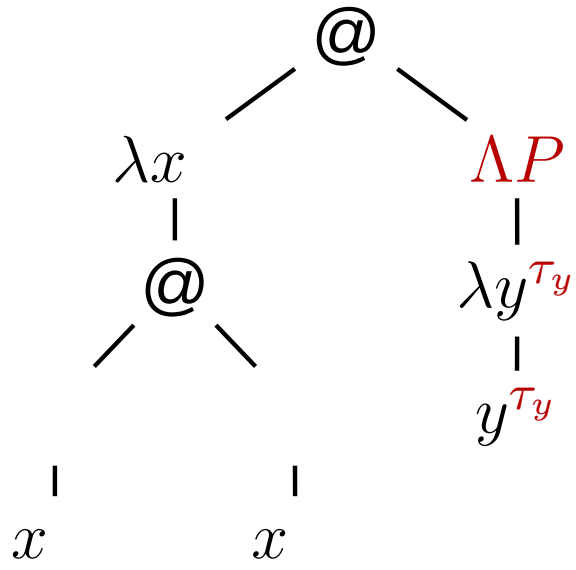
$$(\lambda x.x\ x)\ (\lambda y.y)$$



$$\begin{array}{rcl} \tau & = & \alpha \to \alpha \\ \sigma & = & \tau \to \tau \\ \tau_y & = & \{i = \tau,\ j = \alpha\} \\ \lambda y^{\tau_y}.y^{\tau_y} & : & \tau_y \to \tau_y \\ & \simeq & \{i = \tau \to \tau,\ j = \alpha \to \alpha\} \end{array}$$

Example: A Corresponding λ^{B}-term

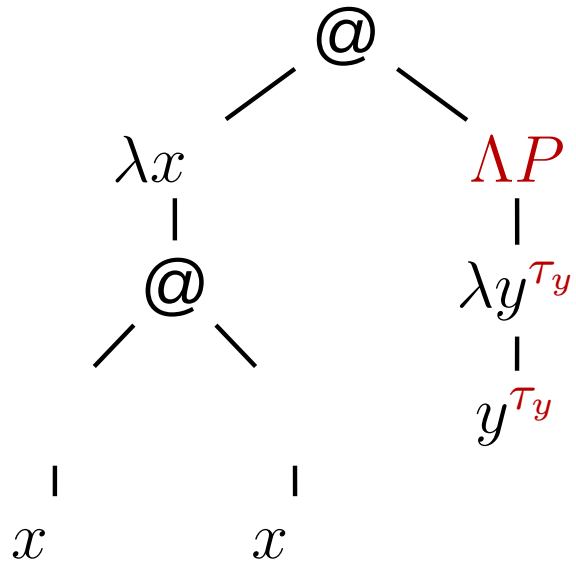
$$(\lambda x.x\ x)\ (\lambda y.y)$$



$$\begin{array}{rcl}
\tau & = & \alpha \to \alpha \\
\sigma & = & \tau \to \tau \\
\tau_y & = & \{i = \tau,\ j = \alpha\} \\
\lambda y^{\tau_y}.y^{\tau_y} & : & \tau_y \to \tau_y \\
 & \simeq & \{i = \tau \to \tau,\ j = \alpha \to \alpha\} \\
P & = & \mathsf{join}\{i = *, j = *\}
\end{array}$$

Example: A Corresponding λ^{B}-term

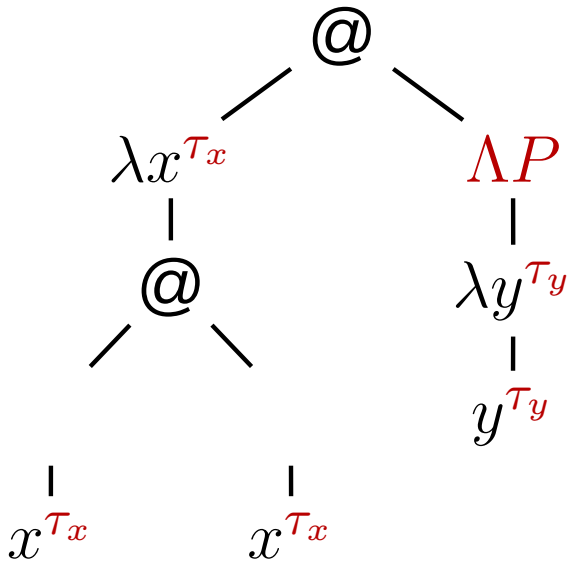
$$(\lambda x.x\ x)\ (\lambda y.y)$$



$$\begin{array}{rcl}
\tau & = & \alpha \to \alpha \\
\sigma & = & \tau \to \tau \\
\tau_y & = & \{i = \tau,\ j = \alpha\} \\
\lambda y^{\tau_y}.y^{\tau_y} & : & \tau_y \to \tau_y \\
 & \simeq & \{i = \tau \to \tau,\ j = \alpha \to \alpha\} \\
P & = & \mathsf{join}\{i = *, j = *\} \\
\Lambda P.\lambda y^{\tau_y}.\tau_y & : & \forall P.(\tau_y \to \tau_y)
\end{array}$$

Example: A Corresponding λ^{B}-term

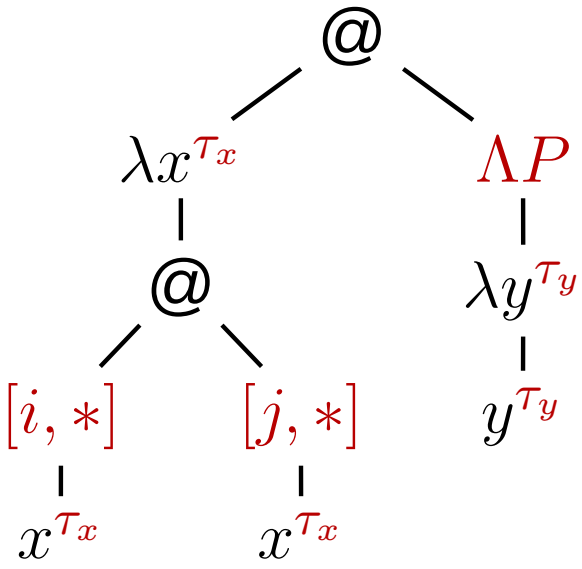
$$(\lambda x.x\ x)\ (\lambda y.y)$$



$$\begin{array}{rcl}
\tau & = & \alpha \to \alpha \\
\sigma & = & \tau \to \tau \\
\tau_y & = & \{i = \tau,\ j = \alpha\} \\
\lambda y^{\tau_y}.y^{\tau_y} & : & \tau_y \to \tau_y \\
 & \simeq & \{i = \tau \to \tau,\ j = \alpha \to \alpha\} \\
P & = & \mathsf{join}\{i = *, j = *\} \\
\Lambda P.\lambda y^{\tau_y}.\tau_y & : & \forall P.(\tau_y \to \tau_y) \\
\tau_x & = & \forall P.(\tau_y \to \tau_y)
\end{array}$$

Example: A Corresponding λ^{B}-term

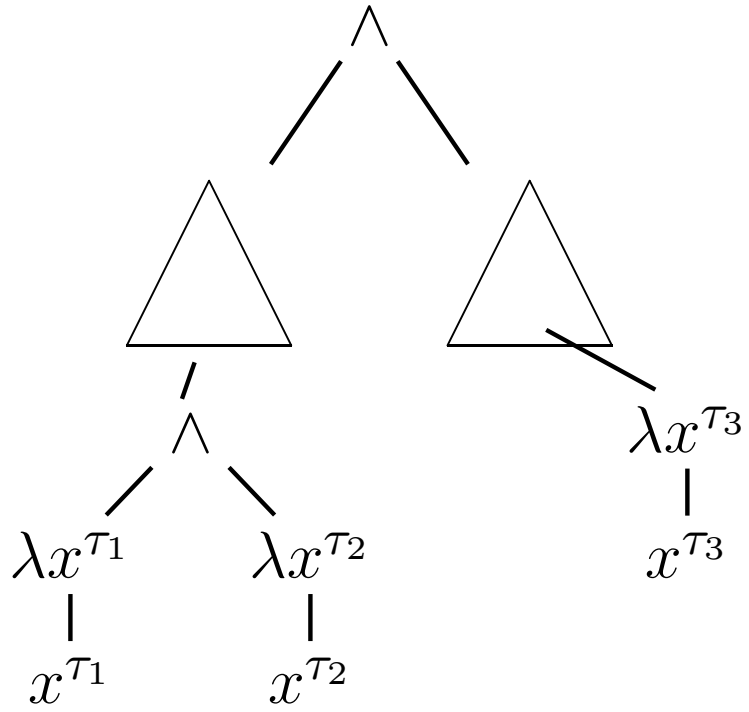
$$(\lambda x.x\ x)\ (\lambda y.y)$$



$$\begin{array}{rcl}
\tau & = & \alpha \to \alpha \\
\sigma & = & \tau \to \tau \\
\tau_y & = & \{i = \tau,\ j = \alpha\} \\
\lambda y^{\tau_y}.y^{\tau_y} & : & \tau_y \to \tau_y \\
 & \simeq & \{i = \tau \to \tau,\ j = \alpha \to \alpha\} \\
P & = & \mathsf{join}\{i = *, j = *\} \\
\Lambda P.\lambda y^{\tau_y}.\tau_y & : & \forall P.(\tau_y \to \tau_y) \\
\tau_x & = & \forall P.(\tau_y \to \tau_y)
\end{array}$$

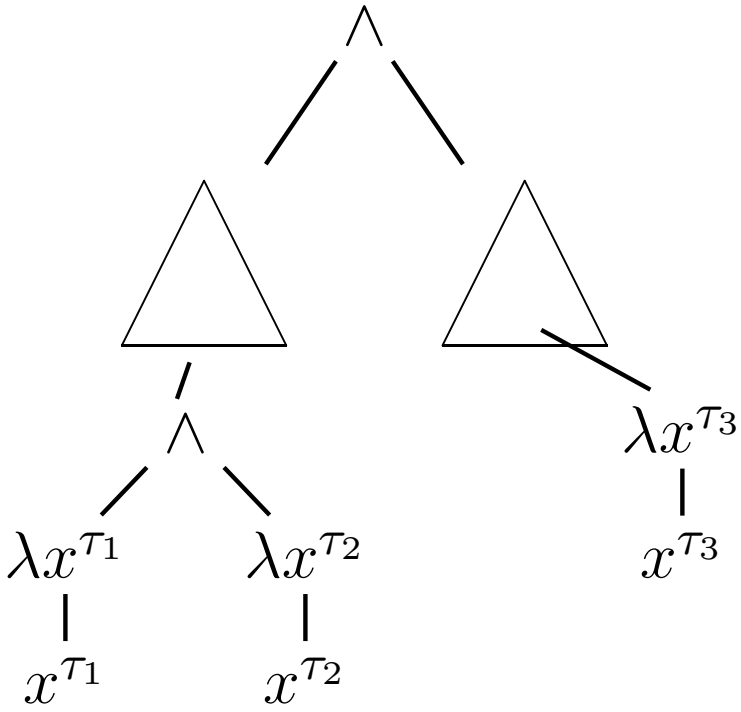
Nested Branching Types

Consider λ^{CIL}-term of this shape:



Nested Branching Types

Consider λ^{CIL}-term of this shape:

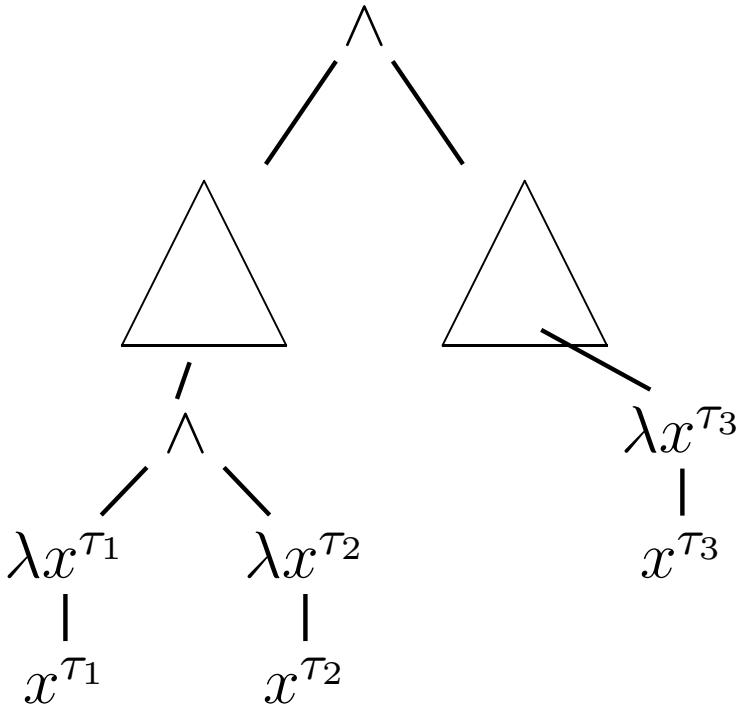


In λ^{B}, the variable x is annotated by this branching type:

$$\{i = \{k = \tau_1,\, l = \tau_2\},\, j = \tau_3\}$$

Nested Branching Types

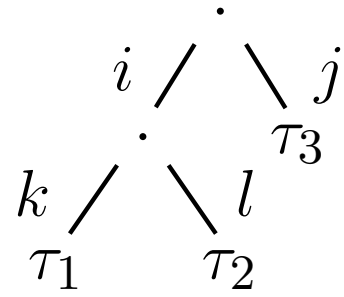
Consider λ^{CIL}-term of this shape:



In λ^{B}, the variable x is annotated by this branching type:

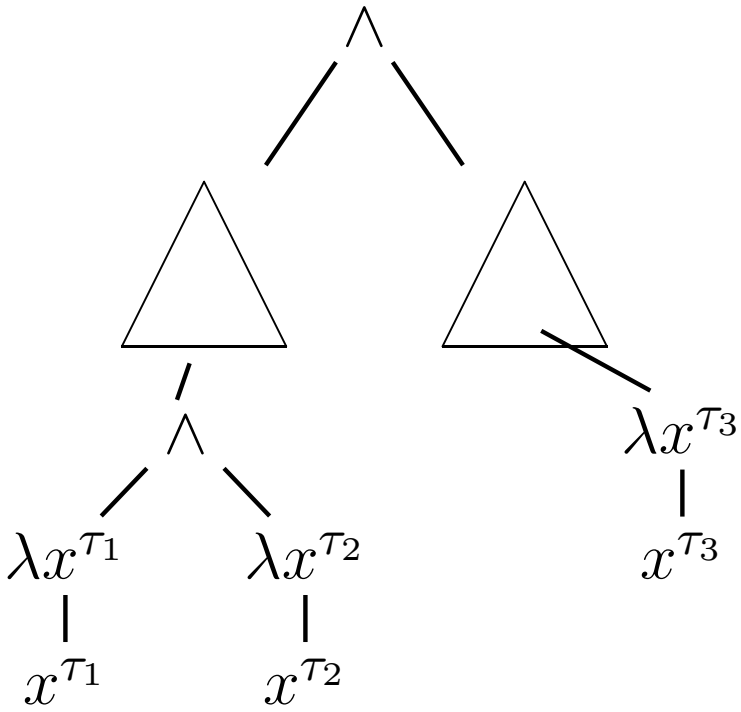
$$\{i = \{k = \tau_1,\, l = \tau_2\},\, j = \tau_3\}$$

This type may be viewed as a tree:



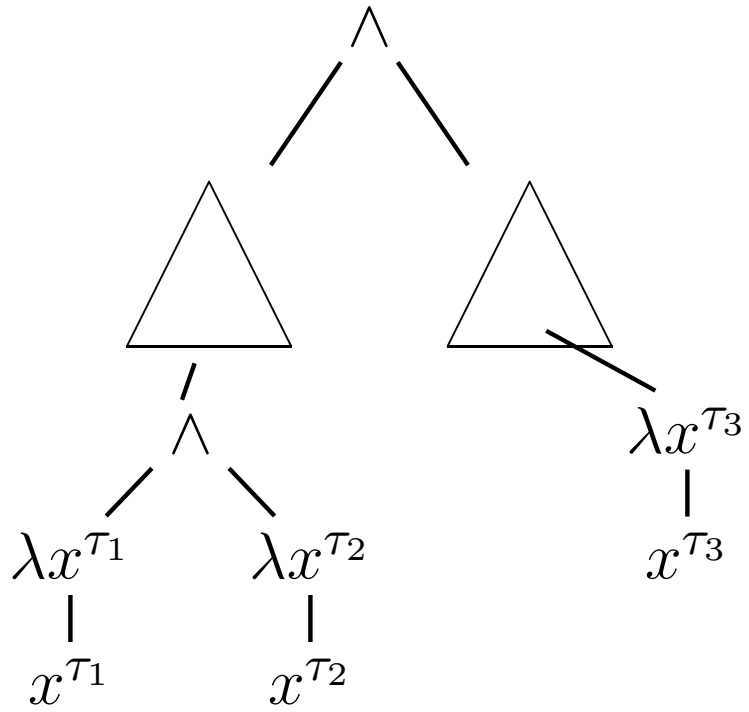
Kinds

Consider λ^{CIL}-term of this shape:



Kinds

Consider λ^{CIL}-term of this shape:



Typing judgments and kinds (branching shapes):

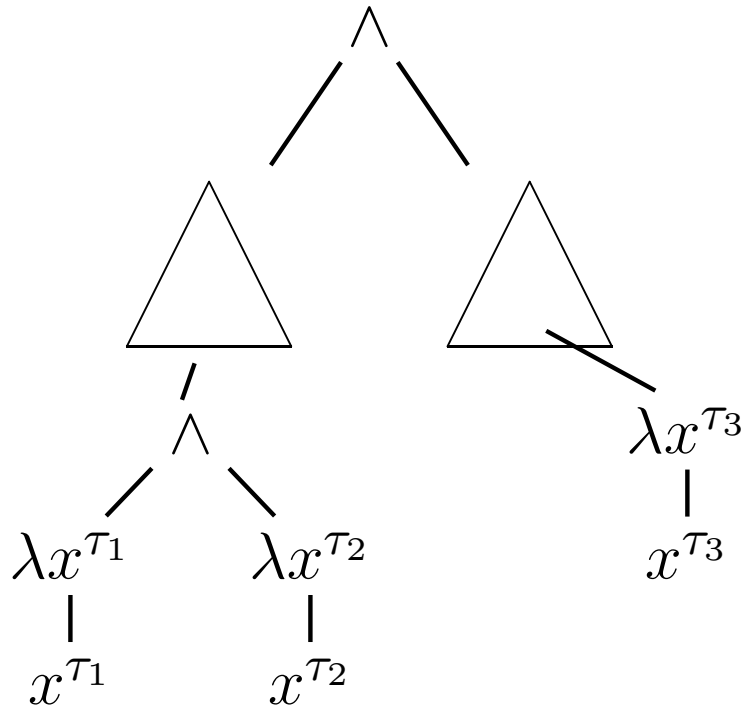
$$E \vdash M : \tau \text{ at } \kappa$$

$$\kappa \in \mathsf{Kind} \ ::= \ * \ \mid \ \{i = \kappa_i\}^I$$

Kind κ structures independent typings of term M.

Kinds

Consider λ^{CIL}-term of this shape:



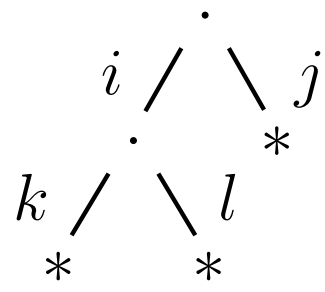
Typing judgments and kinds (branching shapes):

$$E \vdash M : \tau \text{ at } \kappa$$

$$\kappa \in \mathsf{Kind} \ ::= \ * \ \mid \ \{i = \kappa_i\}^I$$

Kind κ structures independent typings of term M.

The kind used for typing $\lambda x.x$:



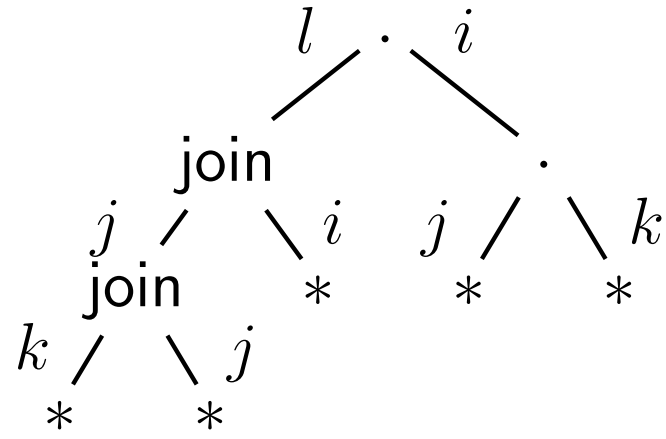
Type Selection Parameters

$$\bar{P} \in \mathsf{IndParameter} \quad ::= \quad * \mid \mathsf{join}\{i = \bar{P}_i\}^I$$
$$P \in \mathsf{Parameter} \quad ::= \quad \bar{P} \mid \{i = P_i\}^I$$

Example:

$$\{l = \mathsf{join}\{j = \mathsf{join}\{k = *,\ j = *\},\ i = *\},\ i = \{j = *,\ k = *\}\}$$

Its tree representation:



Branching Types

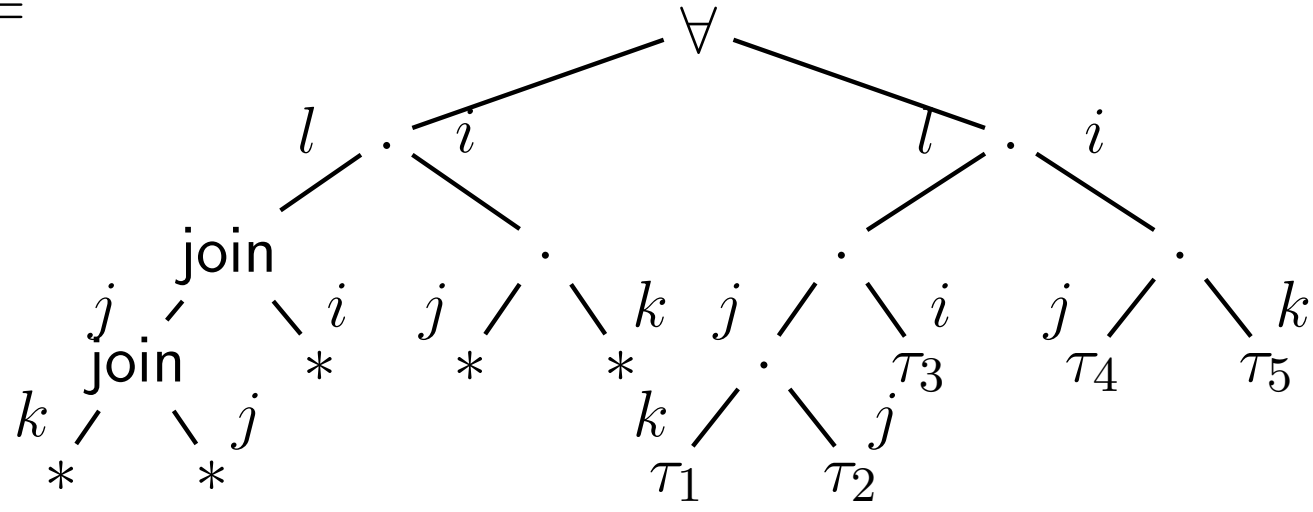
$$\sigma, \tau \in \mathsf{Ty} \quad ::= \quad \alpha \mid \sigma \to \tau \mid \{i = \tau_i\}^I \mid \forall P.\tau$$

Example:

$P = \{l = \mathsf{join}\{j = \mathsf{join}\{k = *,\ j = *\},\ i = *\},\ i = \{j = *,\ k = *\}\}$

$\tau = \{l = \{j = \{k = \tau_1,\ j = \tau_2\},\ i = \tau_3\},\ i = \{j = \tau_4,\ k = \tau_5\}\}$

$\sigma = \forall P.\tau =$



Type Equivalence

The type reduction rules:

$$\begin{array}{rcl} \{i = \sigma_i\}^I \to \{i = \tau_i\}^I & \succ & \{i = \sigma_i \to \tau_i\}^I \\ \forall \{i = P_i\}^I . \{i = \tau_i\}^I & \succ & \{i = \forall P_i . \tau_i\}^I \\ \forall *. \tau & \succ & \tau \end{array}$$

$\succeq$ is the reflexive and transitive closure of $\succ$.
$\simeq$ is the reflexive, transitive, and symmetric closure of $\succ$.

Type Equivalence

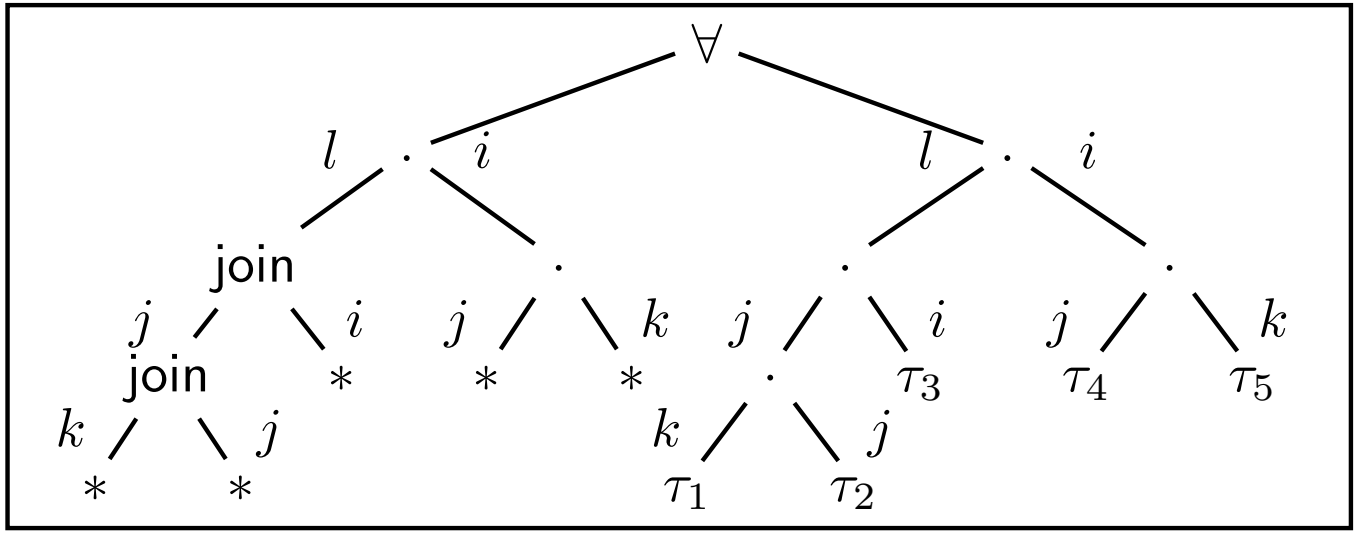
The type reduction rules:

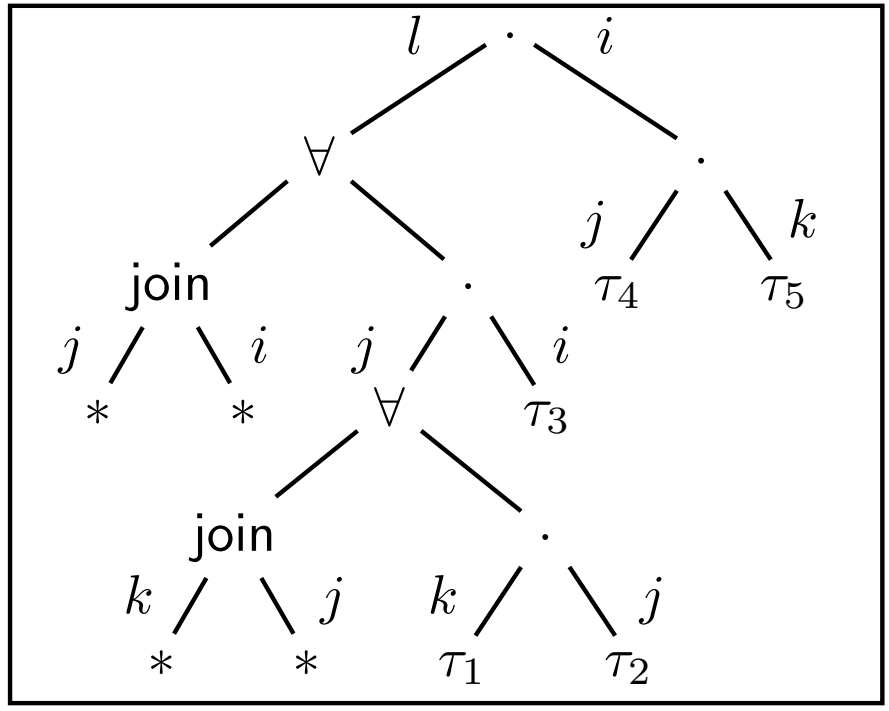
$$\begin{array}{rcl} \{i = \sigma_i\}^I \to \{i = \tau_i\}^I & \succ & \{i = \sigma_i \to \tau_i\}^I \\ \forall \{i = P_i\}^I . \{i = \tau_i\}^I & \succ & \{i = \forall P_i . \tau_i\}^I \\ \forall *. \tau & \succ & \tau \end{array}$$

$\succeq$ is the reflexive and transitive closure of $\succ$.
$\simeq$ is the reflexive, transitive, and symmetric closure of $\succ$.

The type reduction rules "bring a type's branching shape to the surface".

Example of Equivalent Types

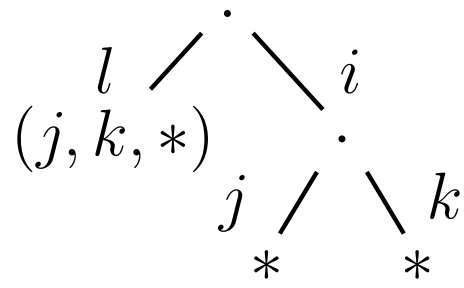




Type Selection Arguments

$$\begin{array}{rcl} \bar{A} \in \mathsf{IndArgument} & ::= & * \mid i, \bar{A} \\ A \in \mathsf{Argument} & ::= & \bar{A} \mid \{i = A_i\}^I \end{array}$$

Example:

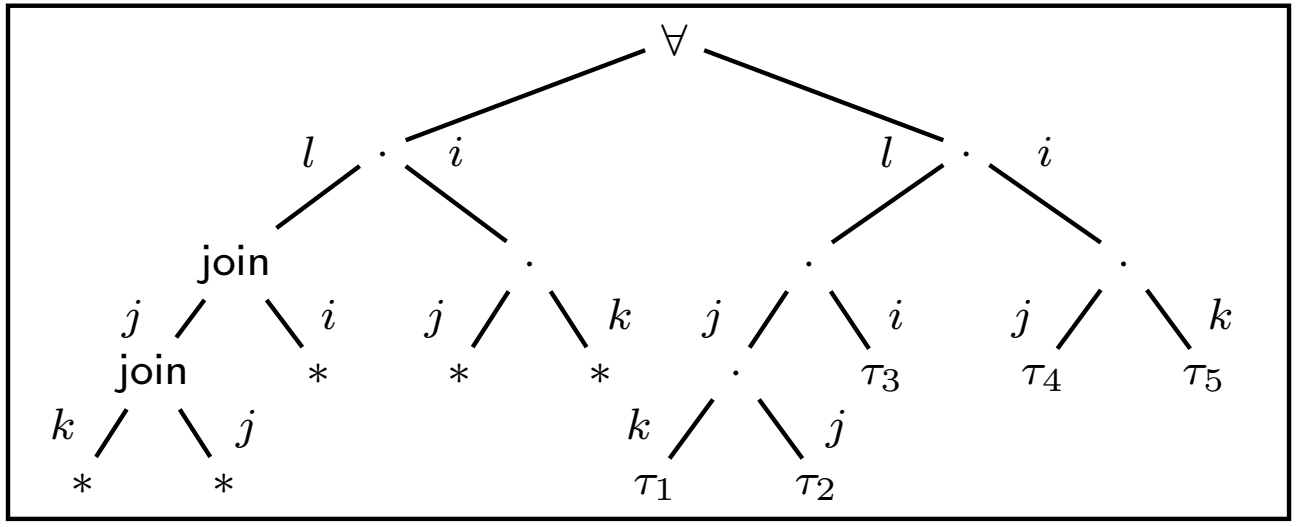


Type Selection

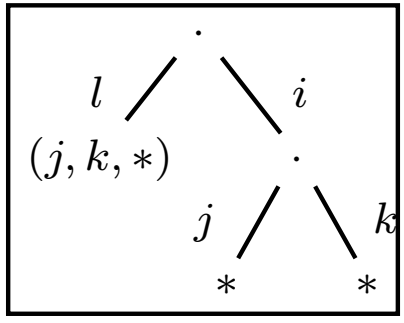
$\mathsf{select}^{\mathsf{i}} : \mathsf{Ty} \times \mathsf{Argument} \longrightarrow \mathsf{Ty}$ (partial function)

Example:

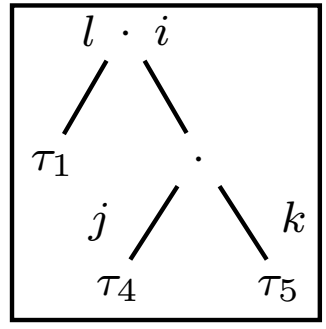
$\sigma =$



$A =$



$\mathsf{select}^{\mathsf{i}}(\tau, A) =$



Terms and Typing Rules

$$M, N \in \mathsf{Term} \quad ::= \quad \Lambda P.M \;\mid\; M[A] \;\mid\; \lambda x^{\tau}.M \;\mid\; M\ N \;\mid\; x^{\tau}$$

Terms and Typing Rules

$$M, N \in \mathsf{Term} \quad ::= \quad \Lambda P.M \;\mid\; M[A] \;\mid\; \lambda x^{\tau}.M \;\mid\; M\ N \;\mid\; x^{\tau}$$

Conversion rule:

$$(\simeq) \qquad \frac{E \vdash M : \tau \text{ at } \kappa}{E \vdash M : \tau' \text{ at } \kappa} \quad \text{if } (\tau \simeq \tau')$$

Terms and Typing Rules

$$M, N \in \mathsf{Term} \quad ::= \quad \Lambda P.M \;\mid\; M[A] \;\mid\; \lambda x^{\tau}.M \;\mid\; M\ N \;\mid\; x^{\tau}$$

Conversion rule:

$$(\simeq) \qquad \frac{E \vdash M : \tau \text{ at } \kappa}{E \vdash M : \tau' \text{ at } \kappa} \quad \text{if } (\tau \simeq \tau')$$

Standard rules:

$$(\mathsf{ax}) \qquad \frac{}{E \vdash x^{\tau} : \tau \text{ at } \kappa} \quad \text{if } (E : \kappa) \text{ and } (\tau \simeq E(x))$$

Terms and Typing Rules

$$M, N \in \mathsf{Term} \quad ::= \quad \Lambda P.M \;\mid\; M[A] \;\mid\; \lambda x^{\tau}.M \;\mid\; M\ N \;\mid\; x^{\tau}$$

Conversion rule:

$$(\simeq) \qquad \frac{E \vdash M : \tau \text{ at } \kappa}{E \vdash M : \tau' \text{ at } \kappa} \quad \text{if } (\tau \simeq \tau')$$

Standard rules:

$$(\mathsf{ax}) \qquad \frac{}{E \vdash x^{\tau} : \tau \text{ at } \kappa} \quad \text{if } (E : \kappa) \text{ and } (\tau \simeq E(x))$$

$$(\rightarrow_{\mathsf{i}}) \qquad \frac{E[x \mapsto \sigma] \vdash M : \tau \text{ at } \kappa}{E \vdash \lambda x^{\sigma}.M : \sigma \rightarrow \tau \text{ at } \kappa}$$

$$(\rightarrow_{\mathsf{e}}) \qquad \frac{E \vdash M : \sigma \rightarrow \tau \text{ at } \kappa; \quad E \vdash N : \sigma \text{ at } \kappa}{E \vdash M\ N : \tau \text{ at } \kappa}$$

Terms and Typing Rules (cont.)

$\forall$-rules:

$$(\forall_{\mathsf{e}}) \qquad \frac{E \vdash M : \tau \text{ at } \kappa}{E \vdash M[A] : \tau' \text{ at } \kappa} \quad \text{if } (\mathsf{select}^{\mathsf{i}}(\tau, A) = \tau')$$

$$(\forall_{\mathsf{i}}) \qquad \frac{\mathsf{expand}(E, P) \vdash M : \tau \text{ at } \lceil P \rceil}{E \vdash \Lambda P.M : \forall P.\tau \text{ at } \lfloor P \rfloor}$$

Term Reduction

Substitution:

$$\begin{array}{rcl} & \ldots & \\ (\Lambda P.M)[x := N] & = & \Lambda P.\ (M[x := \mathsf{expand}(N, P)]) \\ & \ldots & \end{array}$$

Reduction rules:

$$(\beta_\lambda) \qquad ((\lambda x^\tau.M)N) \ \rightarrow \ (M[x := N])$$

Term Reduction

Substitution:

$$\begin{array}{rcl} & \ldots & \\ (\Lambda P.M)[x := N] & = & \Lambda P.\ (M[x := \mathsf{expand}(N, P)]) \\ & \ldots & \end{array}$$

Reduction rules:

$$\begin{array}{lrcl}
(\beta_\lambda) & ((\lambda x^\tau.M)N) & \to & (M[x := N]) \\
(\beta_\Lambda) & (\Lambda P.M)[A] & \to & \Lambda P'.((\mathsf{select}^{\mathsf{b}}(M, A^{\mathsf{s}}))[A^{\mathsf{a}}]), \\
 & \multicolumn{3}{l}{\text{if } \mathsf{match}(P, A) = (P', A^{\mathsf{s}}, A^{\mathsf{a}}) \text{ and } P, A \text{ not trivial}} \\
(*_\Lambda) & (\Lambda P.M) & \to & M, \quad \text{if } P \text{ is trivial} \\
(*_A) & (M[A]) & \to & M, \quad \text{if } A \text{ is trivial}
\end{array}$$

Theorems

Theorem (Subject Reduction).
If $(M \to N)$ and $(E \vdash M : \tau \text{ at } \kappa)$, then $(E \vdash N : \tau \text{ at } \kappa)$.

Type erasure: $|\cdot| : \mathsf{Term} \to \mathsf{UntypedTerm}$

Theorem (Soundness of Reduction).
If $M \to N$, then $|M| \to^* |N|$.

Theorem (Completeness of Reduction).
If M is well-typed and $|M| \to |N|$, then $(M \to^* N)$.

More Theorems

Theorem (Correspondence with Intersection Types).
There exist straightforward translations from typing derivations in λ^{B} to typing derivations in a standard system of intersection types and *vice versa*.

Corollary (Strong Normalization).
If M is a well typed λ^{B}-term, then M and $|M|$ are strongly normalizable.
If pure λ-term M is strongly normalizable, then there exists a well typed λ^{B}-term M' such that $|M'| = M$.

Overview

- Context of research (The Church Project)
- Intersection types
- The problem: disjoint subderivations for a subterm
- Our solution: branching types
- Related and future work

Other Related Work

- Venneri succeeded in completely removing the intersection introduction rule, but this was for combinatory logic [Venneri, 1994; Dezani-Ciancaglini et al., 1997], and the approach does not seem transferable to the λ-calculus.
- Ronchi Della Rocca and Roversi [2001] have a system called Intersection Logic (IL) which is similar to λ^{B}, but has nothing corresponding to our explicitly typed terms or our type equivalences.
- Capitani et al. [2001] have designed a system called HL (Hyperformulae Logic) similar to IL, although it seems to be less complicated. HL also has nothing corresponding to our explicitly typed terms or our type equivalences.

Future Work

- Extend to handle subtyping.
- Extend with support for an equivalent of the ω type constant of some systems with intersection types.
- Extend with union types.
- Integrate with features like the expansion variables of System I [Kfoury and Wells, 1999].
- Extend with many other real language features.
- Implement in a compiler.

Conclusions

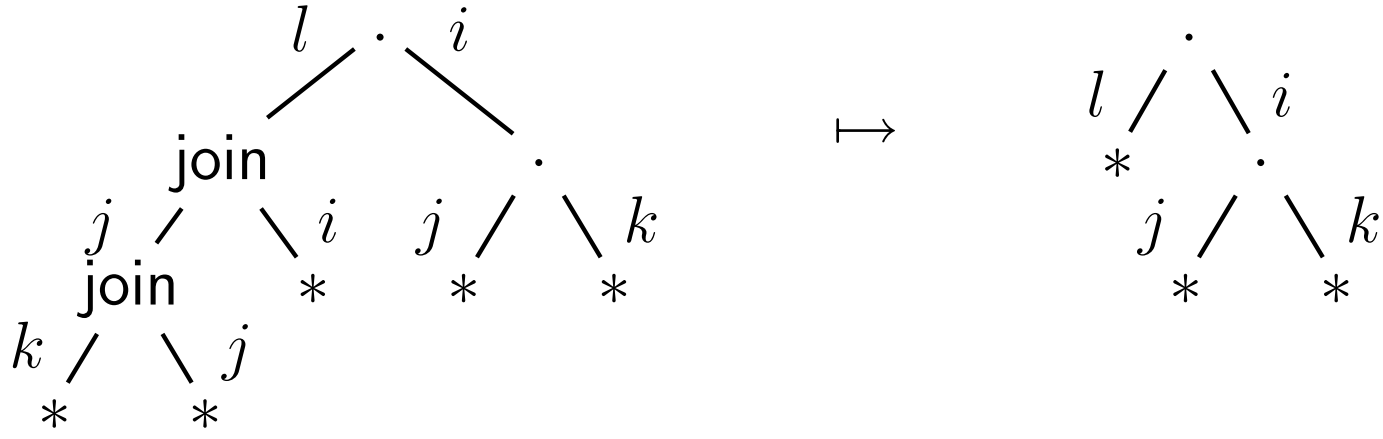
- λ^{B} is the first explicitly typed calculus with the power of intersection types which has the Curry/Howard correspondence (the structure of typed terms corresponds to the structure of the proofs they annotate) and does not duplicate subterms.
- Lots of nice theoretical properties have been verified for λ^{B}: subject reduction, soundness and completeness of typed reduction w.r.t. β-reduction on the corresponding untyped λ-terms, correspondence to traditional intersection types (in unpublished long version).
- In logic, λ^{B} terms may be useful as typed realizers of the so-called *strong conjunction*, but we do not plan to investigate this ourselves.

Outer Kinds of Parameters

$\lfloor \cdot \rfloor$: Parameter $\longrightarrow$ Kind

$$\lfloor * \rfloor = *, \qquad \lfloor \mathsf{join}\{i = \bar{P}_i\}^I \rfloor \; = \; *, \qquad \lfloor \{i = P_i\}^I \rfloor = \{i = \lfloor P_i \rfloor\}^I$$

Example:

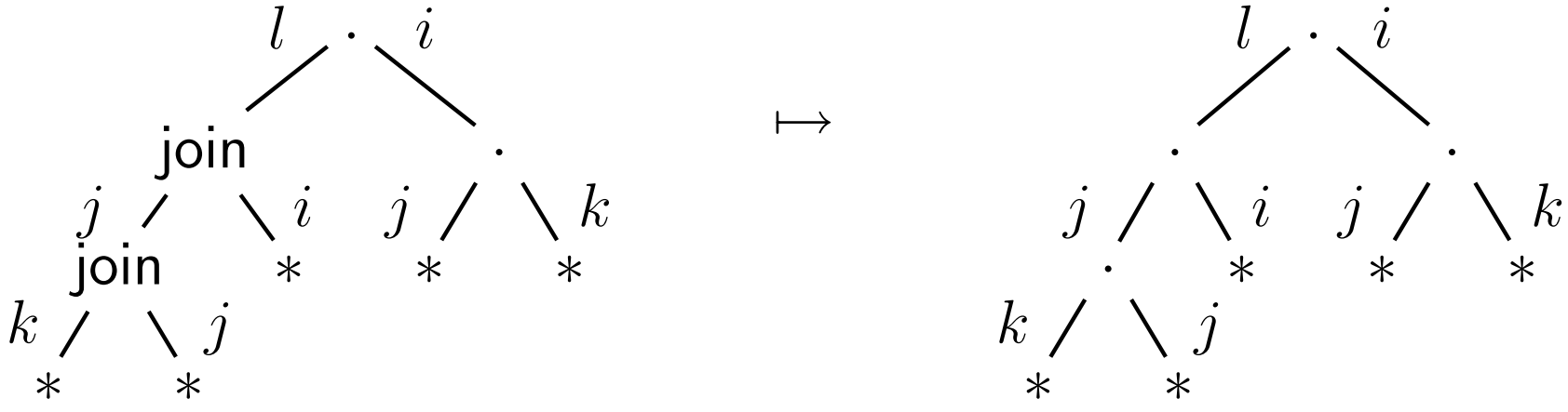


Inner Kinds of Parameters

$\lceil \cdot \rceil$: Parameter $\longrightarrow$ Kind

$\lceil * \rceil = *, \qquad \lceil \mathsf{join}\{i = \bar{P}_i\}^I \rceil = \{i = \lceil \bar{P}_i \rceil^I\}, \qquad \lceil \{i = P_i\}^I \rceil = \{i =$

Example:



References

Anindya Banerjee. A modular, polyvariant, and type-based closure analysis. In ICFP '97 ICFP '97. ISBN 0-89791-918-1.

Franco Barbanera, Mariangiola Dezani-Ciancaglini, and Ugo de'Liguoro. Intersection and union types: Syntax and semantics. *Inform. & Comput.*, 119:202–230, 1995.

Beatrice Capitani, Michele Loreti, and Betti Venneri. Hyperformulae, parallel deductions and intersection types. *Electronic Notes in Theoretical Computer Science*, 50, 2001. URL `http://www.elsevier.nl/locate/entcs/volum` Proceedings of ICALP 2001 workshop: Bohm's Theorem: Applications to Computer Science Theory (BOTH 2001), Crete, Greece, 2001-07-13.

Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Principal type schemes and λ-calculus semantics. In J. R[oger] Hindley and J[onathan] P. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 535–560. Academic Press, 1980. ISBN 0-12-349050-2.

Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, and Betti Venneri. The "relevance" of intersection and union types. *Notre Dame J. Formal Logic*, 38(2):246–269, Spring 1997.

Allyn Dimock, Robert Muller, Franklyn Turbak, and J. B. Wells. Strongly typed flow-directed representation transformations. In ICFP '97 ICFP '97, pages 11–24. ISBN 0-89791-918-1.

Allyn Dimock, Ian Westmacott, Robert Muller, Franklyn Turbak, and J. B. Wells. Functioning without closure: Type-safe customized function representations for Standard ML. In *Proc. 6th Int'l Conf. Functional Programming*, pages 14–25. ACM Press, 2001a. ISBN 1-58113-415-0.

Allyn Dimock, Ian Westmacott, Robert Muller, Franklyn Turbak, J. B. Wells, and Jeffrey Considine. Program representation size in an intermediate language with intersection and union types. In *Types in Compilation, Third Int'l Workshop, TIC 2000*, volume 2071 of *LNCS*, pages 27–52. Springer-Verlag, 2001b. ISBN 3-540-42196-3.

ICFP '97. *Proc. 1997 Int'l Conf. Functional Programming*, 1997. ACM Press. ISBN 0-89791-918-1.

Trevor Jim. What are principal typings and what are they good for? In *Conf. Rec. POPL '96: 23rd ACM Symp. Princ. of Prog. Langs.*, 1996.

Assaf J. Kfoury, Harry G. Mairson, Franklyn A. Turbak, and J. B. Wells. Relating typability and expressibility in finite-rank intersection type systems. In *Proc. 1999 Int'l Conf. Functional Programming*, pages 90–101. ACM Press, 1999. ISBN 1-58113-111-9.

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.

Benjamin C. Pierce. Programming with intersection types, union types, and polymorphism. Technical Report CMU-CS-91-106, Carnegie Mellon University, February 1991.

John C. Reynolds. Design of the programming language Forsythe. In P. O'Hearn and R. D. Tennent, editors, *Algol-like Languages*. Birkhauser, 1996.

Simona Ronchi Della Rocca. Principal type schemes and unification for intersection type discipline. *Theoret. Comput. Sci.*, 59(1–2):181–209, March 1988.

Simona Ronchi Della Rocca and Luca Roversi. Intersection logic. In *Computer Science Logic, CSL '01*. Springer-Verlag, 2001.

Paweł Urzyczyn. Type reconstruction in $\mathbf{F}_\omega$. *Math. Structures Comput. Sci.*, 7(4):329–358, 1997.

Steffen J. van Bakel. *Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems*. PhD thesis, Catholic University of Nijmegen, 1993.

Betti Venneri. Intersection types as logical formulae. *J. Logic Comput.*, 4(2):109–124, April 1994.

J. B. Wells, Allyn Dimock, Robert Muller, and Franklyn Turbak. A typed intermediate language for flow-directed compilation. In *Proc. 7th Int'l Joint Conf. Theory & Practice of Software Development*, pages 757–771, 1997. ISBN 3-540-62781-2. Superseded by Wells et al. [2002].

J. B. Wells, Allyn Dimock, Robert Muller, and Franklyn Turbak. A calculus with polymorphic and polyvariant flow types. *J. Funct. Programming*, 12(3):183–227, May 2002. Supersedes Wells et al. [1997].