Type Inference for Intersection Types with Expansion Variables

Assaf Kfoury
Boston University
Computer Science Dept.
http://www.cs.bu.edu/~kfoury

Joe Wells
Heriot-Watt University
Computing & Elec. Eng. Dept.
http://www.cee.hw.ac.uk/~jbw

June 26, 1999

Lyn Turbak helped with the transparencies.

Overview

- Context of research (Church Project).
- Definitions and relevance.
- Our contributions.
- Technical material.
- Future work.

The Church Project

- Investigates intersection types in compilation of higher-order strongly typed languages. Named in memory of Alonzo Church.

- Members:

Boston Coll.	Bob Muller
Boston Univ.	Assaf Kfoury, Torben Amtoft, Elena Machkasova, Santiago Pericas, Gabriel Seher, Ian Westmacott
Harvard Univ.	Allyn Dimock, Glenn Holloway
Heriot-Watt Univ.	Joe Wells
Stevens Inst. of Tech.	Anindya Banerjee
Wellesley Coll.	Lyn Turbak

  Graduated students: Trevor Jim, Alex Sorg, Dan Bachiochi, Gary McDonald.

- Web address: `http://www.cs.bu.edu/groups/church`

The Church Project: Progress

- λ^{CIL}: explicitly typed calculus, intersection/union types [Wells et al., 1997].
- A polyvariant/polymorphic type/flow analysis algorithm using rank-2 intersection types [Banerjee, 1997].
- Strongly typed representation transformation framework [Dimock, Muller, Turbak, and Wells, 1997] supporting:
  - Standard, selective, and lightweight closure conversion.
  - Flow-directed inlining.
- In progress: An ML compiler using the above.
- 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 (Church Project).
- Definitions and relevance.
  - Type inference.
  - Principal typings.
  - Intersection types.
  - Restriction to finite ranks.
- Our contributions.
- Technical material.
- Future work.

Type Inference

- Type systems prove *judgements* $A \vdash M : \tau$, meaning "term (program) M has type τ using type assumptions A".

- A strong type system only allows safe programs:

  "Well typed programs do not go wrong."

- *Type inference*: given program M, find type environment A and type τ such that $A \vdash M : \tau$ is provable.
  - *Complete*: always finds typing if possible.
  - *Total*: answers "**no**" if no typing exists.

- A complete/total type inference algorithm lets programmers write type-free programs; the compiler verifies safety.

Principal Typings

- A *typing* can be represented by a pair (A, τ) such that $A \vdash M : \tau$, or by the entire derivation $\mathcal{D}$ of $A \vdash M : \tau$. We use both, but in this talk use only the latter.

- T is a *principal typing* for M: all other typings for M are obtained from T by some operations.

- Jim [1996]: "The intuition is, a principal typing EXPECTS LESS of its free variables, and PROVIDES MORE than any other typing judgment."

- Principal typings allow *compositional* type inference:
  - Type inference for fragment uses only type inference *results* for subfragments.
  - Helps in making complete/total type inference algorithm.
  - Helps with separate compilation and modularity.

Intersection Types

- Type polymorphism by *listing* type instances.

- Example comparing with universal types:

intersection types:	$(\lambda x.x) : (\mathrm{int} \to \mathrm{int}) \wedge (\mathrm{bool} \to \mathrm{bool})$
universal types:	$(\lambda x.x) : \forall\alpha.(\alpha \to \alpha)$

  The example intersection type is semantically like $\forall\alpha \in \{\mathrm{int}, \mathrm{bool}\}.(\alpha \to \alpha)$, but there are significant practical differences.

- Benefits of intersection types:
  - Type information more closely approximates actual behavior.
  - More programs are typable, e.g., this non-F_ω-typable term:

$$(\lambda z.c(z(\lambda fg.fg))(z(\lambda vu.uv)))(\lambda y.yyy)$$

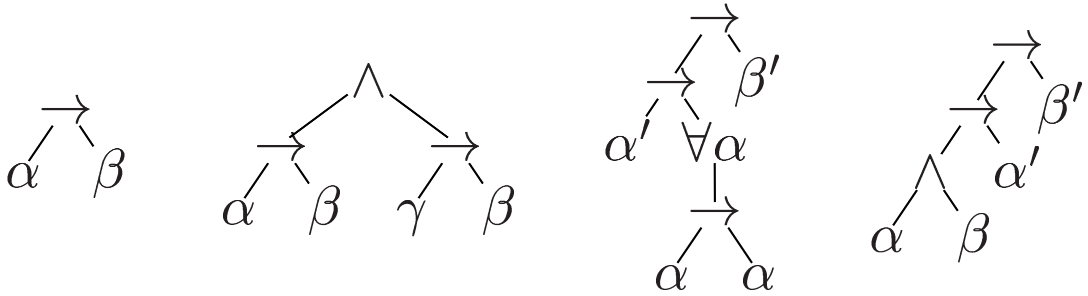
Ranks and the Finite Rank Restriction

- Relative to some advanced type constructor C, e.g., $\wedge$ or $\forall$.

- Counts number of "$\rightarrow$" occurrences an occurrence of C is inside the left argument of [Leivant, 1983].

- Examples:

$\tau =$



$$\mathrm{rank}(\tau) = \qquad 0 \qquad\qquad 1 \qquad\qquad 2 \qquad\qquad 3$$

- Potential benefit: Rank-k restriction of a type system might have decidable typability and a total type inference algorithm.

Overview

- Context of research (Church Project).
- Definitions and relevance.
- Our contributions.
- Technical material.
- Future work.

Our Contributions

- First substitution-based notion of principality for intersection types.

  New technology: *expansion variables*.

- Unification-based type inference algorithm for intersection types. Only returns principal typings.

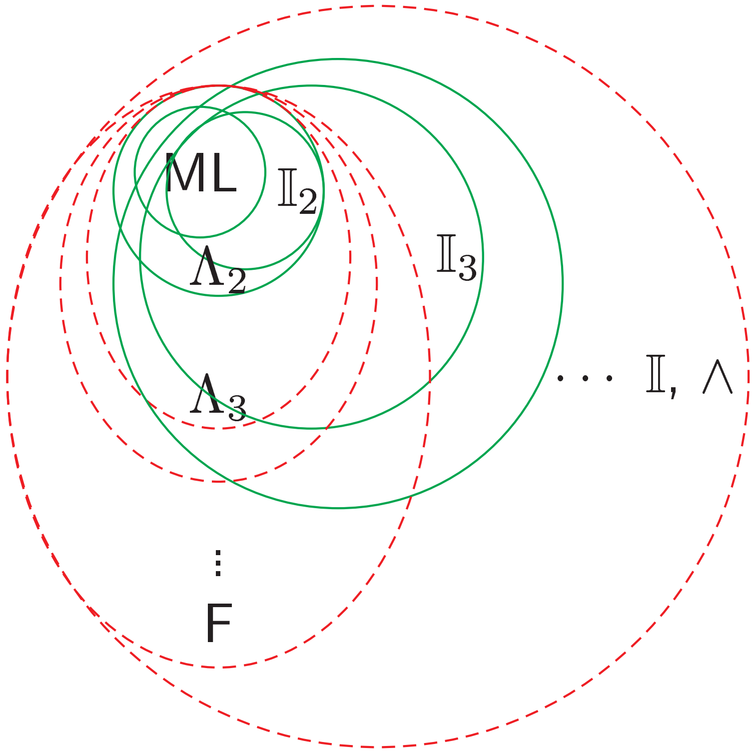
  New technology: *λ-compatible β-unification*.

- Principal typings and complete/total type inference for every finite-rank restriction of intersection types. Proves decidability of finite-rank systems.

Decidability of Typability for Various Systems

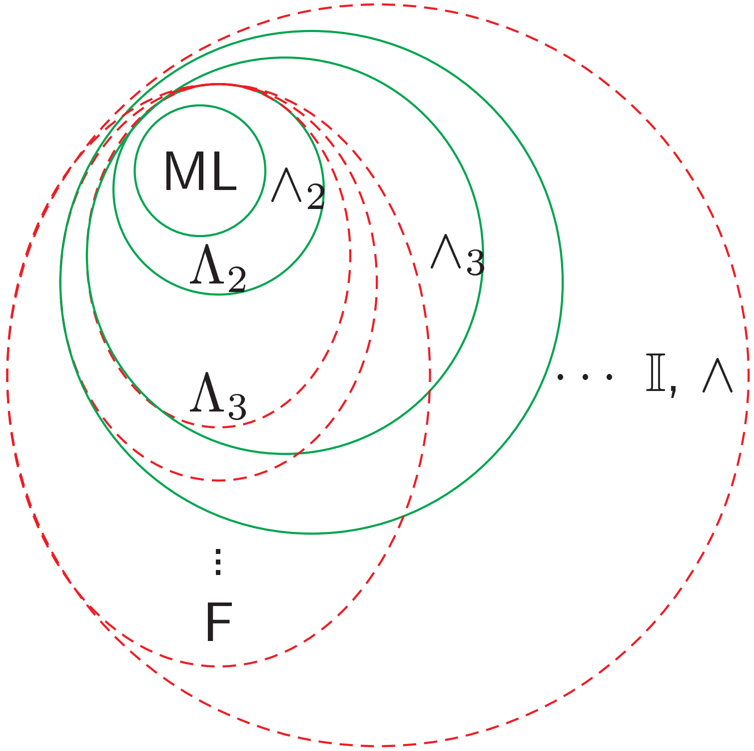
Λ_k: rank-k System F.
$\mathbb{I}$: Our system.
$\mathbb{I}_k$: rank-k System $\mathbb{I}$.
$\wedge$: ACI intersection types.

Decidable.
Undecidable.



Decidability of Typability for Various Systems

Λ_k: rank-k System F.
$\mathbb{I}$: Our system.
$\mathbb{I}_k$: rank-k System $\mathbb{I}$.
$\wedge$: ACI intersection types.
$\wedge_k$: rank-k of $\wedge$.
Decidable.
Undecidable.



Previous Work on Principal Typings

- "Find normal form and compute typing for normal form." By Coppo et al. [1980], Ronchi Della Rocca and Venneri [1984], van Bakel [1993], Sayag and Mauny [1997]. Uses *expansion* (complicated!), *lifting*, *substitution*.

- Unification by Ronchi Della Rocca [1988]. Extremely complicated expansion. Several operations.

- Unification for rank-2 intersection types by van Bakel [1993] and independently by Jim [1996].

Overview

- Context of research (Church Project).
- Definitions and relevance.
- Our contributions.
- Technical material.
  - Types, expansion, substitution, and composition.
  - Typing skeletons/derivations (& more substitution).
  - Constraint sets (& more substitution).
  - Unification problem & algorithm.
  - Type inference.
- Future work.

Types

Variables:

	examples	metavariables
Offsets:	$L, R, LR, LL, RL, RR, \ldots$	s, t
T-variables:	$\mathsf{a}_1^s, \mathsf{a}_2^s, \ldots$	α, β, γ, v
E-variables:	$\mathsf{F}_1^s, \mathsf{F}_2^s, \mathsf{F}_3^s, \ldots$	F, G, H, v

The "offset" part of variable names supports consistent renaming.

Types:

$$\bar{\tau} \in \mathbb{T}^{\rightarrow} ::= \alpha \mid (\tau \rightarrow \bar{\tau})$$

$$\tau \in \mathbb{T} \quad ::= \bar{\tau} \mid (\tau \wedge \tau') \mid (F\, \tau)$$

Note: $\bar{\tau}$ is only "$\rightarrow$" or T-variable. Only $\bar{\tau}$ to right of "$\rightarrow$".

Precedence: $F\, \tau_1 \wedge \tau_2 \rightarrow \tau_3 = ((F\, \tau_1) \wedge \tau_2) \rightarrow \tau_3$

Expansions

Expansions (examples):

$$e_1 = \Box \wedge F\Box, \quad e_2 = (\Box \wedge F\Box) \wedge G\Box, \quad e_3 = (\Box \wedge \Box) \wedge (\Box \wedge \Box)$$

Filling an expansion with types (example):

$$e_2[\alpha, (H\,\beta), (\alpha \rightarrow \beta)] = (\alpha \wedge F(H\,\beta)) \wedge G(\alpha \rightarrow \beta)$$

The paths to the holes in an expansion (example):

$$\mathrm{paths}\,(e_2) = (LL, LR, R)$$

Variable Renaming

Renaming variables by offset extension (examples):

$$\begin{array}{rcl} \alpha = \mathsf{a}_5^{LRL} \text{ and } t = RL & \Rightarrow & \alpha^t = \mathsf{a}_5^{LRLRL} \\ F = \mathsf{F}_i^s & \Rightarrow & F^t = \mathsf{F}_i^{s \cdot t} \end{array}$$

Renaming a type by offset extension of all variables:

$$\langle \tau \rangle^t$$

Restriction. [Well Named Type (Part 1)]
No type variable is an offset extension of another type variable.

The restriction allows renaming sanity.

β-Substitution

β-Substitution: maps T-variables to types ($\mathbb{T}^{\rightarrow}$) and E-variables to expansions.

The *identity β-substitution*:

$$\{\!\![\;]\!\!\}\,(\alpha) = \alpha, \qquad \{\!\![\;]\!\!\}\,(F) = F\,\square$$

Example β-substitution, listing differences from the identity:

$$S = \{\!\![F := \square \wedge G\square,\ \alpha := \beta,\ \alpha' := \beta \rightarrow \beta]\!\!\}$$

Lifting a β-substitution from variables to types:

$$\begin{aligned}
\overline{S}(\alpha) &= S(\alpha),\\
\overline{S}(\tau \rightarrow \bar{\tau}) &= \overline{S}(\tau) \rightarrow \overline{S}(\bar{\tau}),\\
\overline{S}(\tau_1 \wedge \tau_2) &= \overline{S}(\tau_1) \wedge \overline{S}(\tau_2),\\
\overline{S}(F\tau) &= S(F)[\overline{S}(\langle\tau\rangle^{s_1}), \ldots, \overline{S}(\langle\tau\rangle^{s_n})]\\
&\quad \text{where } \mathrm{paths}(S(F)) = (s_1, \ldots, s_n).
\end{aligned}$$

β-Substitution: Example 1

$$S_1 = \left\{ \left[\begin{array}{l} F := H_1\Box \wedge \Box, \\ G := \Box \wedge H_2\Box, \\ \alpha := \alpha_1, \\ \beta := \beta_1 \end{array} \right] \right\}$$

$$\tau = \alpha \wedge FG\beta$$

$$\overline{S_1}(\tau) = \alpha_1 \wedge (H_1 G^L \beta^L \wedge G^R \beta^R)$$

β-Substitution: Example 2

$$
\begin{array}{rcl}
S_2 & = & \left\{ \begin{bmatrix} F := H_1\square \wedge \square, \\ G^L := \square, \\ G^R := \square \wedge H_2\square, \\ \alpha := \alpha_1, \\ \beta^L := \beta_1, \\ \beta^{RL} := \beta_2, \\ \beta^{RR} := \beta_3 \end{bmatrix} \right\} \\
\tau & = & \alpha \wedge FG\beta \\
\overline{S_2}(\tau) & = & \alpha_1 \wedge (H_1\beta_1 \wedge (\beta_2 \wedge H_2\beta_3))
\end{array}
$$

E-paths

E-path set of a variable v in a type τ (examples):

$$\tau = F\,\alpha \wedge G(H\,\beta \wedge \alpha)$$

$$\text{E-path}(\alpha, \tau) = \{F, G\}, \quad \text{E-path}(\beta, \tau) = \{GH\}, \quad \text{E-path}(G, \tau) = \{\varepsilon\}$$

Restriction. [Well Named Type (Part 2)]
$\text{E-path}(v, \tau)$ *must be a singleton set.*

E-path environment of a well named type (example):

$$\tau' = F\,\alpha \wedge G(H\,\beta \wedge \gamma)$$

$$\text{E-env}(\tau') = \{F \mapsto \varepsilon, G \mapsto \varepsilon, H \mapsto G, \alpha \mapsto F, \beta \mapsto GH, \gamma \mapsto G\}$$

Safe Composition

Problem: $\overline{S_2} \circ \overline{S_1}$ is not necessarily $\overline{S_3}$ for any S_3.

The *safe composition* of S_1 and S_2 relative to E-path environment $\mathcal{E}$ is a β-substitution $S_2 \otimes_{\mathcal{E}} S_1$. (Details in paper.)

Lemma. [Safe Composition Works] *If τ and $\overline{S_1}(\tau)$ are well-named and $\mathcal{E} \supseteq$ E-env(τ), then $\overline{S_2 \otimes_{\mathcal{E}} S_1}(\tau) = (\overline{S_2} \circ \overline{S_1})(\tau)$.*

Overview

- Context of research (Church Project).
- Definitions and relevance.
- Our contributions.
- Technical material.
  - Types, expansion, substitution, and composition.
  - Typing skeletons/derivations (& more substitution).
  - Constraint sets (& more substitution).
  - Unification problem & algorithm.
  - Type inference.
- Future work.

Type Environments

Type environment (example): $A = \{x : F\,\alpha,\ y : F\,\alpha \to \beta\}$

Extending/overriding a mapping: $A[x \mapsto \tau]$

Removing a mapping: $A \backslash x$

E-var prefixing: $(F\,A)(x) = F(A(x))$

Joining type environments:

$$(A \wedge B)(x) = \begin{cases} A(x) \wedge B(x) & \text{if both } A(x) \text{ and } B(x) \text{ defined,} \\ A(x) & \text{if only } A(x) \text{ defined,} \\ B(x) & \text{if only } B(x) \text{ defined,} \\ \text{undefined} & \text{if both } A(x) \text{ and } B(x) \text{ undefined.} \end{cases}$$

Commentary on Typing Rules

- *Skeleton*: like a typing derivation, but the function, argument, and result types need not match in the APP rule. (Used in type inference algorithm.)

- Invariant of derivations/skeletons: Type assumption for a variable iff variable actually used.

- A separate ABS-K rule is needed because of the above invariant.

- F-rules are placeholders for trees of $\wedge$-rule uses, which will be introduced by β-substitution later.

Typing Rules of System $\mathbb{I}$

Rules for both Skeletons and Derivations

VAR $\quad x : \bar{\tau} \vdash x : \bar{\tau}$

$\wedge \quad \dfrac{A_1 \vdash_? M : \tau_1; \quad A_2 \vdash_? M : \tau_2}{(A_1 \wedge A_2) \vdash_{\mathrm{e}} M : (\tau_1 \wedge \tau_2)}$

ABS-I $\quad \dfrac{A[x \mapsto \tau] \vdash M : \bar{\tau}}{A \vdash (\lambda x.M) : (\tau \to \bar{\tau})}$

$F \quad \dfrac{A \vdash_? M : \tau}{(F\, A) \vdash_{\mathrm{e}} M : (F\, \tau)}$

ABS-K $\quad \dfrac{A \vdash M : \bar{\tau}; \quad x \notin \mathrm{FV}(M)}{A \vdash (\lambda x.M) : (\bar{\tau}' \to \bar{\tau})}$

Rule for Skeletons Only

$(\mathcal{S})$APP $\quad \dfrac{A_1 \vdash M : \bar{\tau}; \quad A_2 \vdash_? N : \tau}{(A_1 \wedge A_2) \vdash M\, N : \bar{\tau}'}$

Rule for Derivations Only

$(\mathcal{D})$APP $\quad \dfrac{A_1 \vdash M : \tau \to \bar{\tau}; \quad A_2 \vdash_? N : \tau}{(A_1 \wedge A_2) \vdash M\, N : \bar{\tau}}$

Judgements marked $\vdash_{\mathrm{e}}$ can only be used in premise positions marked $\vdash_?$.

Some Properties of System $\mathbb{I}$

M is typable: there is a derivation (*not just a skeleton*) ending in $A \vdash M : \bar{\tau}$.

Theorem. *Strong normalization is equivalent to typability in System $\mathbb{I}$.*

Corollary. *Typability in System $\mathbb{I}$ is undecidable.*

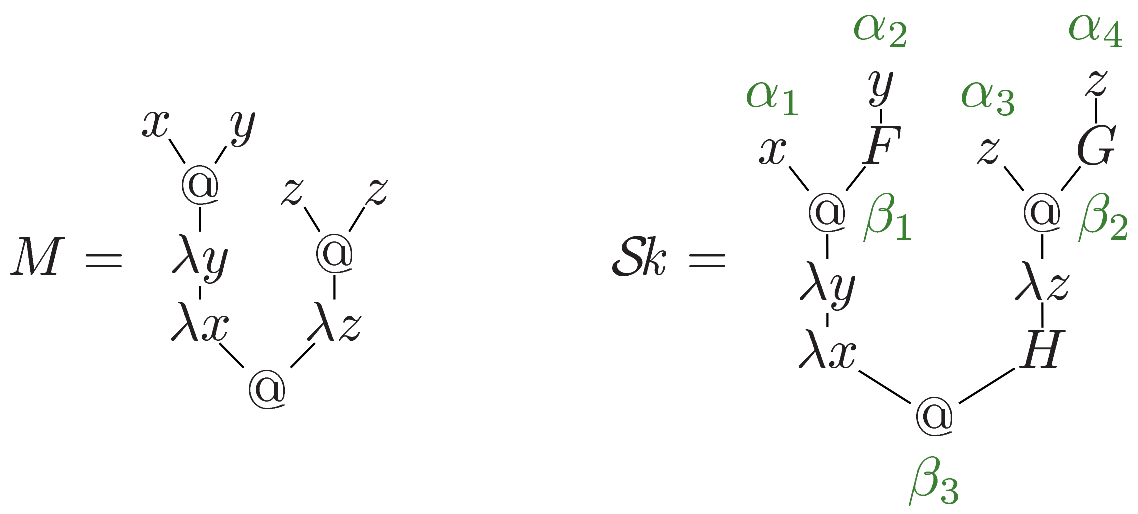
System $\mathbb{I}$ does *not* have subject reduction, but β-reduction preserves typability. Example of no subject reduction:

$$\begin{array}{lcll}
\{z : \alpha_2 \to \alpha_1 \to \alpha_3, w : \alpha_1 \wedge \alpha_2\} & \vdash_{\mathbb{I}} & (\lambda x.(\lambda y.zyx)x)w & : \alpha_3 \\
\{z : \alpha_2 \to \alpha_1 \to \alpha_3, w : \alpha_1 \wedge \alpha_2\} & \not\vdash_{\mathbb{I}} & (\lambda x.zxx)w & : \alpha_3 \\
\{z : \alpha_2 \to \alpha_1 \to \alpha_3, w : \alpha_2 \wedge \alpha_1\} & \vdash_{\mathbb{I}} & (\lambda x.zxx)w & : \alpha_3
\end{array}$$

Skeletons for Terms

Example: Let $M = (\lambda x.\lambda y.xy)(\lambda z.zz)$.

Skeleton $\mathcal{S}k = \mathrm{Skel}(M)$:

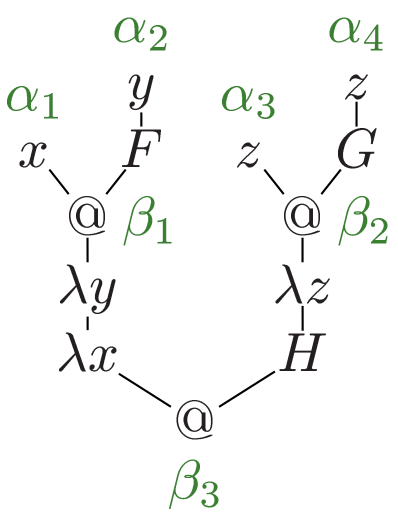


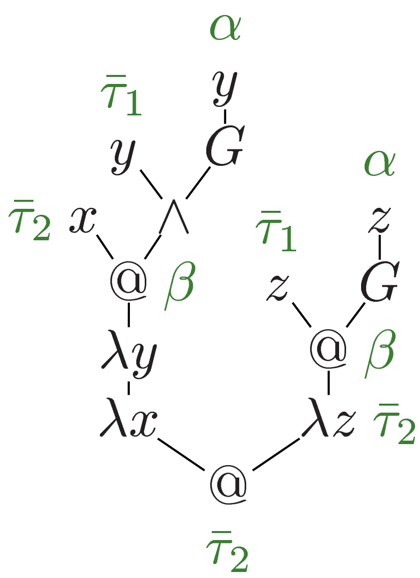
Notation for skeletons: Leaves and applications annotated with derived types. All other type information can be derived from these.

β-Substitution on Skeletons

Example: Let M and $\mathcal{S}k = \mathrm{Skel}(M)$ be as before.

$$S = \left\{\begin{bmatrix} H := \square, & F := \square \wedge G\square, & \alpha_1, \beta_3 := \bar{\tau}_2, \\ \beta_1, \beta_2 := \beta, & \alpha_3, \alpha_2^L := \bar{\tau}_1, & \alpha_2^R := \alpha \end{bmatrix}\right\} \quad \text{where} \quad \begin{aligned} \bar{\tau}_1 &= G\,\alpha \to \beta \\ \bar{\tau}_2 &= (\bar{\tau}_1 \wedge G\,\alpha) \to \beta \end{aligned}$$

$\mathcal{S}k =$ 

$\overline{S}(\mathcal{S}k) =$ 

$\overline{S}(\mathcal{S}k)$ is in fact a derivation and thus M is typable.

β-Substitution on Skeletons: Properties

Lemma. *β-substitution on a skeleton yields a skeleton. β-substitution on a derivation yields a derivation.*

Principal typing for M: a derivation $\mathcal{D}$ that types M such that any other typing $\mathcal{D}' = S(\mathcal{D})$ for some β-substitution S.

Example: $\overline{S}(\mathcal{S}k)$ is a principal typing for M.

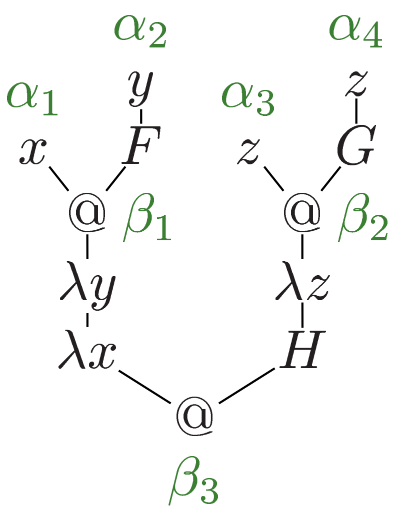
Overview

- Context of research (Church Project).
- Definitions and relevance.
- Our contributions.
- Technical material.
  - Types, expansion, substitution, and composition.
  - Typing skeletons/derivations (& more substitution).
  - Constraint sets (& more substitution).
  - Unification problem & algorithm.
  - Type inference.
- Future work.

Constraint Sets for Terms

Constraint set for a term (example): Let M and $\mathcal{S}k = \mathrm{Skel}(M)$ be as before.

$\mathcal{S}k =$



$$\Gamma\,(\mathcal{S}k) = \left\{ \begin{array}{ll} \alpha_1 & \doteq F\,\alpha_2 \to \beta_1, \\ H\,\alpha_3 & \doteq H(G\,\alpha_4 \to \beta_2), \\ \alpha_1 \to F\,\alpha_2 \to \beta_1 & \doteq H(\alpha_3 \wedge G\,\alpha_4 \to \beta_2) \to \beta_3 \end{array} \right\}$$

Note: One constraint per application.

Constraint satisfaction is difference between $(\mathcal{S})$APP rule and $(\mathcal{D})$APP rule.

Operations on Constraint Sets

- β-substitution and application of E-variables are extended to things containing types (e.g., constraints and constraint sets) by distributing these operations across all of the types.

- *Simplification* on constraints and constraint sets:
  - Breaks apart constraints with same outermost type constructor ($\wedge$ or $\rightarrow$), after looking inside matching outermost E-variables. (Preserves polarity.)
  - Removes solved constraints.

Solutions for Constraint Sets

Example: Let M, $\mathcal{S}\!k = \mathrm{Skel}(M)$, $\Delta = \Gamma(\mathcal{S}\!k)$, and S be as before.

$$\Delta = \Gamma\,(\mathcal{S}\!k) = \left\{ \begin{array}{ll} \alpha_1 & \doteq F\,\alpha_2 \to \beta_1, \\ H\,\alpha_3 & \doteq H(G\,\alpha_4 \to \beta_2), \\ \alpha_1 \to F\,\alpha_2 \to \beta_1 & \doteq H(\alpha_3 \wedge G\,\alpha_4 \to \beta_2) \to \beta_3 \end{array} \right\}$$

$$S = \left\{ \begin{bmatrix} H := \Box, & F := \Box \wedge G\Box, & \alpha_1, \beta_3 := \bar{\tau}_2, \\ \beta_1, \beta_2 := \beta, & \alpha_3, \alpha_2^L := \bar{\tau}_1, & \alpha_2^R := \alpha \end{bmatrix} \right\} \text{ where } \begin{array}{l} \bar{\tau}_1 = G\,\alpha \to \beta \\ \bar{\tau}_2 = (\bar{\tau}_1 \wedge G\,\alpha) \to \beta \end{array}$$

Then S is a solution for Δ.

Principal solution for Δ: a β-substitution S that solves Δ such that if S' also solves Δ, then $S'(\Delta) = S''(S(\Delta))$ for some β-substitution S''.

Example: S is a principal solution for Δ.

Positive/Negative Restrictions

Positive/negative positions: polarity switches between "$\rightarrow$" and left component.

Restricted type:

- $\wedge$ occurs only negatively.
- Each positive position to the left of "$\rightarrow$" has exactly one E-variable.
- No E-variables occur positively to the right of "$\rightarrow$".

	metavariable
positive restricted:	ρ
negative restricted:	σ
positive restricted and "$\rightarrow$" or T-variable:	$\bar{\rho}$
negative restricted and "$\rightarrow$" or T-variable:	$\bar{\sigma}$

Restricted constraint: $(\vec{F}\rho \doteq \vec{F}\sigma)$. (Left side positive, right negative.) The $\vec{F}$ are *outer* occurrences; all other variable occurrences are *inner*.

λ-Compatible Constraint Sets

λ-compatible constraint set:

- Well named.
- Restrictions on polarity and inner/outer occurrences:
  - All constraints in restricted form (detailed earlier).
  - E-variables have at most one inner positive occurrence.
  - T-variables occur at most twice, at most once for each polarity.
  - No variable has inner occurrences on both sides of a constraint.
- Some requirements on connectedness are met. (Details in paper.)

Lemma. $\Gamma(\mathrm{Skel}(M))$ *is λ-compatible for any M.*

Overview

- Context of research (Church Project).
- Definitions and relevance.
- Our contributions.
- Technical material.
  - Types, expansion, substitution, and composition.
  - Typing skeletons/derivations (& more substitution).
  - Constraint sets (& more substitution).
  - Unification problem & algorithm.
  - Type inference.
- Future work.

λ-Compatible β-Unification Algorithms

- *λ-Compatible β-Unification*: Finding a β-substitution solving a λ-compatible constraint set.

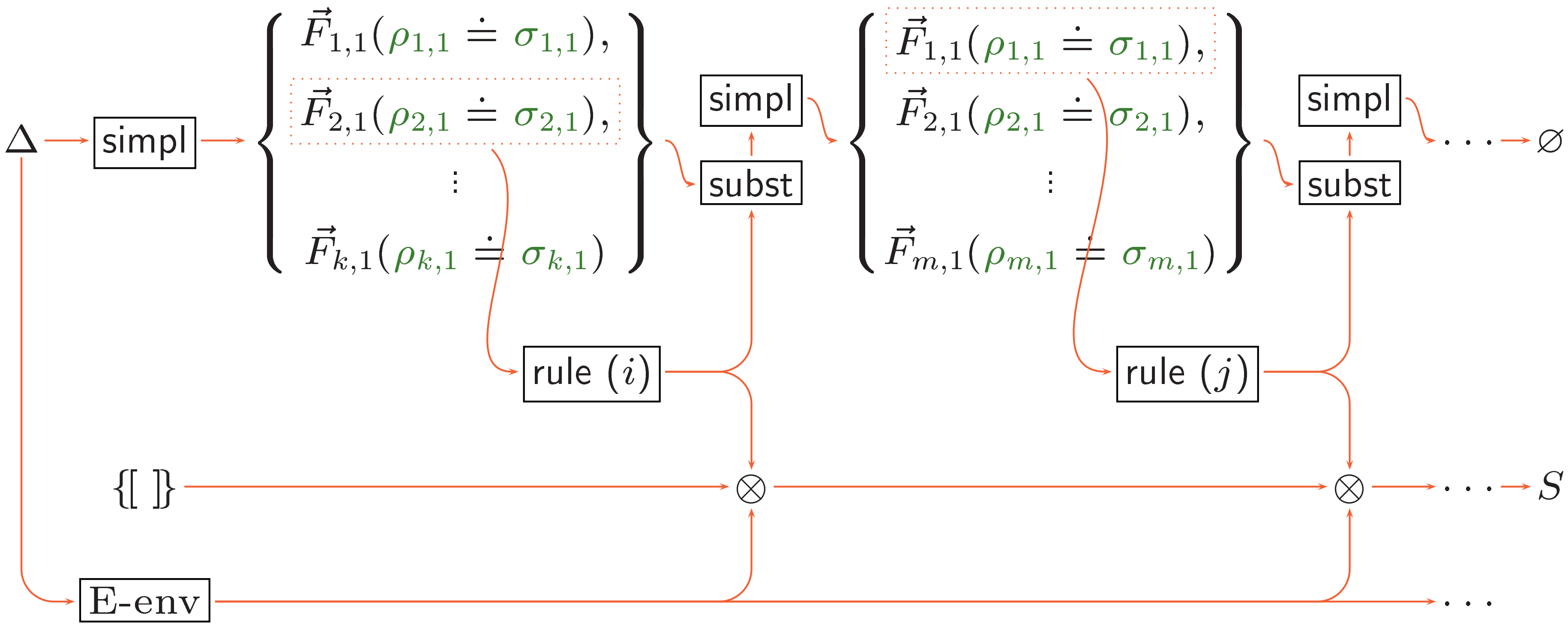
- $\mathrm{Unify}(\Delta)$ halts with a principal solution for Δ iff there is a solution for Δ.

- $\mathrm{UnifyFR}$ is a variant of Unify with additional parameters. (Details in paper).

  When given appropriate values for the extra parameters, $\mathrm{UnifyFR}(\Gamma(\mathrm{Skel}(M)), \ldots)$ yields a principal solution for $\Gamma(\mathrm{Skel}(M))$ when there is a rank-k typing for M and otherwise returns "**no**". (Details in paper.)

  There will not be enough time to present $\mathrm{UnifyFR}$.

Algorithm Unify



Calculation of $\mathrm{Unify}(\Delta) \stackrel{*}{\Rightarrow} S$.

Algorithm Unify

Mode of operation:

- Invocation: $\mathrm{Unify}(\Delta) \Rightarrow \mathrm{Unify}(\mathrm{simplify}(\Delta), \{\![\,]\!\}, \mathrm{E\text{-}env}(\Delta))$.
- Unification step: $\mathrm{Unify}(\Delta_0, S_0, \mathcal{E}) \Rightarrow \mathrm{Unify}(\Delta_1, S_1, \mathcal{E})$, provided:
  - $\vec{F}(\rho \doteq \sigma) \in \Delta_0$.
  - $\rho \doteq \sigma \Rightarrow S$ is an instance of a rule.
  - $\Delta_1 = \mathrm{simplify}(S(\Delta_0))$.
  - $S_1 = S \otimes_{\mathcal{E}} S_0$.
- Successful termination: $\mathrm{Unify}(\varnothing, S, \mathcal{E}) \Rightarrow S$.

Rules:

$$
\begin{array}{lll}
\alpha \doteq \bar{\sigma} & \Rightarrow \{\![\alpha := \bar{\sigma}]\!\} & \textbf{(1)} \\
\bar{\rho} \doteq \alpha & \Rightarrow \{\![\alpha := \bar{\rho}]\!\} & \textbf{(2)} \\
F\,\bar{\rho} \doteq e[\bar{\sigma}_1, \ldots, \bar{\sigma}_n] & \Rightarrow \{\![F := e]\!\} & \textbf{(3)}
\end{array}
$$

Properties of the Algorithm

- Correctness: Each step preserves λ-compatibility and solvability.
- Progress: At least one rule can always be used.
- Termination when solvable: A metric for solvable constraint sets strictly decreases with each step.

Theorem. [Soundness, Completeness, & Principality]
Let Δ be λ-compatible. Then:

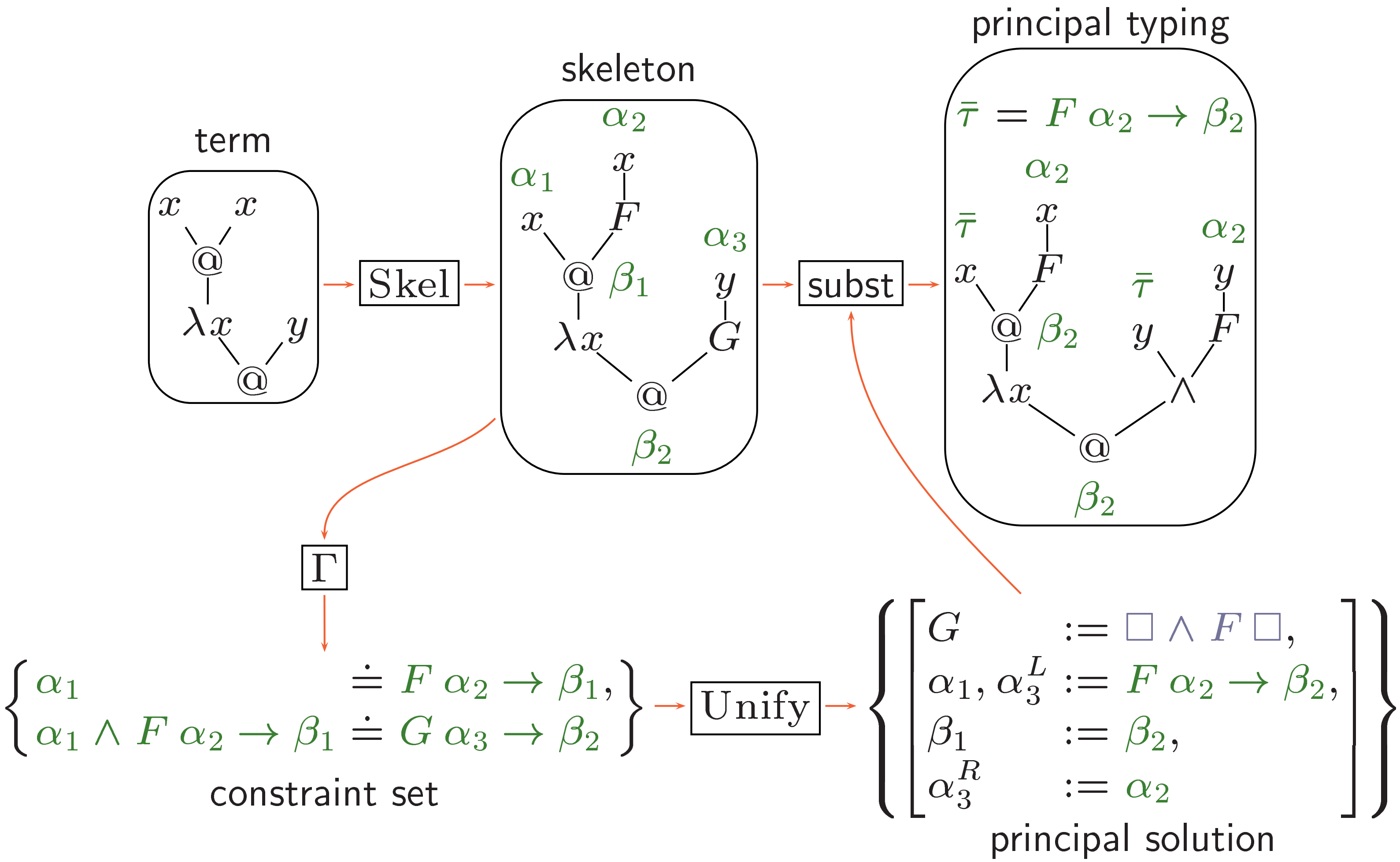
1. *Δ has a solution if and only if $\mathrm{Unify}(\Delta) \stackrel{*}{\Rightarrow} S$.*
2. *If $\mathrm{Unify}(\Delta) \stackrel{*}{\Rightarrow} S$, then S is a principal solution for Δ.*

Example: Using M, S, and $\Delta = \Gamma(\mathrm{Skel}(M))$ from earlier, $\mathrm{Unify}(\Delta) \stackrel{*}{\Rightarrow} S$.

Overview

- Context of research (Church Project).
- Definitions and relevance.
- Our contributions.
- Technical material.
  - Types, expansion, substitution, and composition.
  - Typing skeletons/derivations (& more substitution).
  - Constraint sets (& more substitution).
  - Unification problem & algorithm.
  - Type inference.
- Future work.

Type Inference Algorithm



Overview

- Context of research (Church Project).
- Definitions and relevance.
- Our contributions.
- Technical material.
- Future work.

Future Work: Practical Concerns

- Associative, commutative, and idempotent (ACI) treatment of "$\wedge$". Necessary for obtaining efficient algorithms.
- Support sum types (booleans/conditionals, datatypes/case). Probably requires union types ("$\vee$").
- Support recursive definitions. (Hard!)
- Use skeleton & constraints as "lazy", space-efficient ADT for polyvariant analyses.
- Modularity.

Future Work: Theoretical Concerns

- Clarify relationship between β-unification and other forms of unification such as second-order unification and semi-unification.

- Find more conditions under which β-unification (1) satisfies principality and (2) is decidable.

- Analyze complexity of decidable finite-rank restriction of β-unification.

References

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

Anindya Banerjee. A modular, polyvariant, and type-based closure analysis. In ICFP '97 ICFP [1997].

Mario Coppo, M. Dezani-Ciancaglini, and B. Venneri. Principal type schemes and λ-calculus semantics. In Jonathan P. Seldin and J. Roger Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 535–560. Academic Press, 1980.

Allyn Dimock, Robert Muller, Franklyn Turbak, and J. B. Wells. Strongly typed flow-directed representation transformations. In ICFP '97 ICFP [1997].

Proc. 1997 Int'l Conf. Functional Programming, 1997.

Trevor Jim. What are principal typings and what are they good for? In *Conf. Rec. POPL '96: 23rd ACM Symp. Principles of Prog. Languages*, 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*, 1999.

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. Principles of Prog. Languages*, 1999.

Daniel Leivant. Polymorphic type inference. In *Conf. Rec. 10th Ann. ACM Symp. Principles of Programming Languages*, pages 88–98, 1983.

Simona Ronchi Della Rocca. Principal type schemes and unification for intersection type discipline. *Theor. Comp. Sc.*, 59:181–209, 1988.

Simona Ronchi Della Rocca and B. Venneri. Principal type schemes for an extended type theory. *Theor. Comp. Sc.*, 28:151–169, 1984.

Émilie Sayag and Michel Mauny. Structural properties of intersection types. In *Proceedings of the 8th International Conference on Logic and Computer Science – Theoretical Foundations of Computing (LIRA)*, pages 167–175, Novi Sad, Yugoslavia, 1997.

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*, 1997. Superseded by Wells et al. [199X].

J. B. Wells, Allyn Dimock, Robert Muller, and Franklyn Turbak. A calculus with polymorphic and polyvariant flow types. *J. Funct. Prog.*, 199X. To appear after revisions. Supersedes Wells et al. [1997].