An Introduction to Non-idempotent Intersection Types

Robert Atkey
University of Strathclyde
robert.atkey@strath.ac.uk

MSP101 – 24th February 2022

What are types for?

What are types for?

Types to tell us what programs to write

What are types for?

Types to tell us what programs to write

or

What are types for?

Types to tell us what programs to write

or

Types to tell us about the programs we've written

Types to tell us what programs to write

A problem to be solved:

$$\Gamma \vdash\ ?\ : A$$

We use the structure of A to guide us.

“Type Driven Development”, “Correct-by-Construction”, “Hole-driven development”, ...

Terms are meaningless without their types.

Terms (generally) have a unique type.

Types to tell us about the programs we've written

$$\Gamma \vdash t : ?$$

- t has meaning whether or not it has a type
- Every t may have multiple types; often related by subtyping
- Currently a "hot topic": TypeScript; Typed Python; Typed Ruby; …

Intersection types

If a terms has multiple types, then…

$$\frac{\Gamma \vdash t : \tau_1 \qquad \Gamma \vdash t : \tau_2}{\Gamma \vdash t : \tau_1 \wedge \tau_2}$$

t can act as described by τ_1 and τ_2.

Intersection types

If a terms has multiple types, then...

$$\frac{\Gamma \vdash t : \tau_1 \qquad \Gamma \vdash t : \tau_2}{\Gamma \vdash t : \tau_1 \wedge \tau_2}$$

t can act as described by τ_1 and τ_2.

A typical CBN function type:

$$(\tau_1 \wedge \cdots \wedge \tau_n) \to \tau$$

because a function may make multiple demands on its input.

Intersection types

If a terms has multiple types, then...

$$\frac{\Gamma \vdash t : \tau_1 \qquad \Gamma \vdash t : \tau_2}{\Gamma \vdash t : \tau_1 \wedge \tau_2}$$

t can act as described by τ_1 and τ_2.

A typical CBN function type:

$$(\tau_1 \wedge \cdots \wedge \tau_n) \to \tau$$

because a function may make multiple demands on its input.

Intersection types have remarkable properties:

- Can be used to characterise terms that have normal forms
- Essentially because they completely describe all possible behaviours

Idempotency

Typical subtyping rules:

- $\tau_1 \wedge \tau_2 \sqsubseteq \tau_1$
- $\tau_1 \wedge \tau_2 \sqsubseteq \tau_2$
- $\tau \sqsubset \tau_1$ and $\tau \sqsubset \tau_2$ implies $\tau \sqsubset \tau_1 \wedge \tau_2$

Implies idempotency: $\tau = \tau \wedge \tau$.

Idempotency

Typical subtyping rules:

- $\tau_1 \wedge \tau_2 \sqsubseteq \tau_1$
- $\tau_1 \wedge \tau_2 \sqsubseteq \tau_2$
- $\tau \sqsubset \tau_1$ and $\tau \sqsubset \tau_2$ implies $\tau \sqsubset \tau_1 \wedge \tau_2$

Implies idempotency: $\tau = \tau \wedge \tau$.

If we know that a term has a behaviour, it doesn't matter how much we know it.

Idempotency

Typical subtyping rules:

- $\tau_1 \wedge \tau_2 \sqsubseteq \tau_1$
- $\tau_1 \wedge \tau_2 \sqsubseteq \tau_2$
- $\tau \sqsubset \tau_1$ and $\tau \sqsubset \tau_2$ implies $\tau \sqsubset \tau_1 \wedge \tau_2$

Implies idempotency: $\tau = \tau \wedge \tau$.

If we know that a term has a behaviour, it doesn't matter how much we know it.

This talk: non-idempotent intersection types:

$$(\tau_1 \sqcap \cdots \sqcap \tau_n) \to \tau$$

Functions make multiple demands on their input, and we count how many times.

Untyped λ-calculus

$$s, t \quad ::= \quad x \mid \lambda x.t \mid s\,t$$

Krivine Abstract Machine

An abstract machine for executing programs via a Call-by-Name strategy.

Environments and Closures

- An environment η is a finite mapping from variables to closures;
- A closure $c = (t, \eta)$ is a pair of a term t and an environment η for all its free vars

Stacks and Configurations

- Stacks π are lists of closures
- Configuration is a triple $\langle t, \eta, \pi \rangle$ of a term, an environment for it and a stack.

Execution rules

$$
\begin{array}{llcll}
\textsc{Var} & \langle x, \eta, \pi \rangle & \longrightarrow & \langle t, \eta', \pi \rangle & \eta(x) = (t, \eta') \\
\textsc{Pop} & \langle \lambda x.t, \eta, s \cdot \pi \rangle & \longrightarrow & \langle t, \eta[x \mapsto s], \pi \rangle & \\
\textsc{Push} & \langle s\, t, \eta, \pi \rangle & \longrightarrow & \langle s, \eta, (t, \eta) \cdot \pi \rangle &
\end{array}
$$

Execution rules

$$
\begin{array}{llcll}
\textsc{Var} & \langle x, \eta, \pi \rangle & \longrightarrow & \langle t, \eta', \pi \rangle & \eta(x) = (t, \eta') \\
\textsc{Pop} & \langle \lambda x.t, \eta, s \cdot \pi \rangle & \longrightarrow & \langle t, \eta[x \mapsto s], \pi \rangle & \\
\textsc{Push} & \langle s\, t, \eta, \pi \rangle & \longrightarrow & \langle s, \eta, (t, \eta) \cdot \pi \rangle &
\end{array}
$$

Complete execution of a closed term t:

$$\langle t, \emptyset, [] \rangle \longrightarrow *\langle \lambda x.t', \emptyset, [] \rangle$$

Execution rules

$$\begin{array}{llcll}
\textsc{Var} & \langle x, \eta, \pi \rangle & \longrightarrow & \langle t, \eta', \pi \rangle & \eta(x) = (t, \eta') \\
\textsc{Pop} & \langle \lambda x.t, \eta, s \cdot \pi \rangle & \longrightarrow & \langle t, \eta[x \mapsto s], \pi \rangle & \\
\textsc{Push} & \langle s\, t, \eta, \pi \rangle & \longrightarrow & \langle s, \eta, (t, \eta) \cdot \pi \rangle &
\end{array}$$

Complete execution of a closed term t:

$$\langle t, \emptyset, [] \rangle \longrightarrow *\langle \lambda x.t', \emptyset, [] \rangle$$

Simulates CBN execution, keeping substitution and stack explicit.

The plan

A non-idempotent intersection type system where:

1. A closed term is typable if and only if it halts;
2. The size of the typing derivation is equal to the number of steps.

The types and judgements

The types and judgements

Types

$$
\begin{array}{rcl}
\tau & ::= & * \mid \sigma \mapsto \tau \\
\sigma & ::= & [\tau_1 \sqcap \cdots \sqcap \tau_n]
\end{array}
$$

In intersection types: n can be zero, and order doesn't matter (i.e. finite multisets).

The types and judgements

Types

$$\begin{array}{rcl} \tau & ::= & * \mid \sigma \mapsto \tau \\ \sigma & ::= & [\tau_1 \sqcap \cdots \sqcap \tau_n] \end{array}$$

In intersection types: n can be zero, and order doesn't matter (i.e. finite multisets).

Judgements

$$x_1 : \sigma_1, \cdots, x_n : \sigma_n \vdash t : \tau$$

where $x_1, \cdots, x_n$ are include the free variables of t.

Typing Rules

$$\frac{}{x_1 : [], \cdots, x_i : \tau, \cdots, x_n : [] \vdash x_i : \tau}\ \textsc{Var} \qquad \frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x.t : \sigma \mapsto \tau}\ \textsc{Lam}$$

$$\frac{}{x_1 : [], \cdots, x_n : [] \vdash \lambda x.t : *}\ \textsc{Obs} \qquad \frac{\Gamma \vdash s : [\tau_1 \sqcap \cdots \sqcap \tau_n] \mapsto \tau \qquad \langle \Gamma_i \vdash t : \tau_i \rangle_i}{\Gamma + \Sigma_i \Gamma_i \vdash s\, t : \tau}\ \textsc{App}$$

where addition of contexts is pointwise multiset union.

Examples

A typing of the identity function:

$$\vdash \lambda x.x : [*] \mapsto *$$

Examples

A typing of the identity function:

$$\vdash \lambda x.x : [*] \mapsto *$$

Describing the behaviour of a term when it is applied to the identity function:

$$\vdash \lambda f.\, \lambda x.\, f x : [[*] \mapsto *] \mapsto [*] \mapsto *$$

Examples

A typing of the identity function:

$$\vdash \lambda x.x : [*] \mapsto *$$

Describing the behaviour of a term when it is applied to the identity function:

$$\vdash \lambda f.\, \lambda x.\, f x : [[*] \mapsto *] \mapsto [*] \mapsto *$$

A term that does the same thing, but uses its first argument twice:

$$\vdash \lambda f.\, \lambda x.\, f(f x) : [[*] \mapsto * \sqcap [*] \mapsto *] \mapsto [*] \mapsto *$$

Examples

A typing of the identity function:

$$\vdash \lambda x.x : [*] \mapsto *$$

Describing the behaviour of a term when it is applied to the identity function:

$$\vdash \lambda f.\, \lambda x.\, f x : [[*] \mapsto *] \mapsto [*] \mapsto *$$

A term that does the same thing, but uses its first argument twice:

$$\vdash \lambda f.\, \lambda x.\, f(f x) : [[*] \mapsto * \sqcap [*] \mapsto *] \mapsto [*] \mapsto *$$

Notes:

- If we had idempotency, then these two types would be the same
- There are infinitely many types / behaviours, these ones are describing what behaviour we need from the inputs to get the desired final output.

Putting it together

$$\vdash (\lambda f.\lambda x.\, f(f\,x))\;(\lambda x.\, x) : [*] \mapsto *$$

and

$$\vdash (\lambda f.\lambda x.\, f(f\,x))\;(\lambda x.\, x)\;(\lambda x. t) : *$$

where t is *any* term.

Putting it together

$$\vdash (\lambda f.\lambda x. f(fx))\ (\lambda x.\ x) : [*] \mapsto *$$

and

$$\vdash (\lambda f.\lambda x. f(fx))\ (\lambda x.\ x)\ (\lambda x.t) : *$$

where t is *any* term.

The term $\lambda f.\lambda x. fx$ would have the same typing.

But the size of the derivation is different! The argument $(\lambda x.x)$ gets typed twice!

Typing KAM configurations

Environments and Closures

$$\frac{\forall x. \forall i.\ \vdash_c \eta(x) : \Gamma(x)_i}{\vdash_e \eta : \Gamma} \qquad \frac{\Gamma \vdash t : \tau \qquad \vdash_e \eta : \Gamma}{\vdash_c (t, \eta) : \tau}$$

Stacks and Configurations

$$\frac{}{\vdash_s [\,] : \tau \multimap \tau} \qquad \frac{\langle \vdash_c c : \sigma_i \rangle_i \qquad \vdash \pi : \tau_1 \multimap \tau_2}{\vdash_s c \cdot \pi : (\sigma \mapsto \tau_1) \multimap \tau_2}$$

$$\frac{\vdash_c (t, \eta) : \tau_1 \qquad \vdash_s \pi : \tau_1 \multimap \tau_2}{\vdash_{cfg} \langle t, \eta, \pi \rangle : \tau_2}$$

Properties

Properties

Subject reduction

- $\vdash_{cfg} p : \tau$ and $p \longrightarrow q$ implies $\vdash_{cfg} q : \tau$

Properties

Subject reduction

- $\vdash_{cfg} p : \tau$ and $p \longrightarrow q$ implies $\vdash_{cfg} q : \tau$

Subject expansion

- $\vdash_{cfg} q : \tau$ and $p \longrightarrow q$ implies $\vdash_{cfg} p : \tau$

Properties

Subject reduction

- $\vdash_{cfg} p : \tau$ and $p \longrightarrow q$ implies $\vdash_{cfg} q : \tau$

Subject expansion

- $\vdash_{cfg} q : \tau$ and $p \longrightarrow q$ implies $\vdash_{cfg} p : \tau$

Progress

- $\vdash_{cfg} p : *$ and $| \vdash_{cfg} p : *| > 0$, then exists q, $p \longrightarrow q$.

Properties

Subject reduction

- $\vdash_{cfg} p : \tau$ and $p \longrightarrow q$ implies $\vdash_{cfg} q : \tau$

Subject expansion

- $\vdash_{cfg} q : \tau$ and $p \longrightarrow q$ implies $\vdash_{cfg} p : \tau$

Progress

- $\vdash_{cfg} p : *$ and $|\vdash_{cfg} p : *| > 0$, then exists q, $p \longrightarrow q$.

Subject reduction

- $\vdash_{cfg} p : \tau$ and $p \longrightarrow q$ implies $\vdash_{cfg} q : \tau$
  and $|\vdash_{cfg} p : \tau| = |\vdash_{cfg} q : \tau| + 1$

Properties

Soundness & Completeness

- $\vdash_{cfg} p : *$ iff p terminates in $|\vdash_{cfg} p : *|$ steps.

What is really going on?

What is really going on?

Relational model of Linear Logic:

- Formulae interpreted as sets:

$$
\begin{aligned}
&[\![A \otimes B]\!] = [\![A \parr B]\!] = [\![A \multimap B]\!] = [\![A]\!] \times [\![B]\!] \\
&[\![!A]\!] = \mathcal{M}_f([\![A]\!])
\end{aligned}
$$

- Proofs interpreted as relations
- Reflexive domain: $D \cong (\mathcal{M}_f(D) \times D) + 1$
  Satisfies $D \subseteq (!D \multimap D)$