Hinting
A nice presentation of algorithmic typing

Wojciech Kołowski

Goals of this research and presentation

- Research the difference between extrinsic and intrinsic lambda calculi and decide which style is better for Axi. In practice most languages are hybrid, but it's still nice to make decisions in a principled way.
- Establish a simple, readable and easy to follow framework for implementing type checking and type inference in Axi, so that we can concentrate on proposing new features or language extensions.
- Disseminate some knowledge on these topics to the team.

Actionable insights for Axi

- For now I will lean towards making my Axi proposals more extrinsic.
- I will (try to) follow hint-based typing when making future Axi proposals.
- I recommend Axi prototypes to implement type checking and inference based on hinting as presented here.

Extrinsic vs Intrinsic

There are two kinds of lambda calculi, intrinsically typed also called (Church-style) and extrinsically typed (also called Curry-style).

Intrinsic calculi – examples

- $\lambda x : \texttt{Int}.\, x$ – the identity function at type `Int`.
- $\lambda x : \texttt{String}.\, x$ – the identity function at type `String`. This is a different term from the above.
- $\lambda x.\, x$ – not a term, because it cannot be assigned a unique type.
- $(\lambda x : \texttt{String}.\, x)\ 5$ – not a term, as it cannot be assigned a type because of a type error.

Intrinsic calculi

In intrinsic calculi, terms are inseparable from their types. From the user's point of view, all terms are well-typed and have a unique type. Ill-typed "terms" are not terms, but abominations forbidden by the type checker. Types are disjoint, so terms that could possibly be assigned two different types must be banished from existence. If a term is needed at two different types, there are two distinct copies of it.

Of course, from the language designer's point of view, ill-typed "terms" still exist (they are usually called preterms) and need to be dealt with, it's just that the users don't care.

Extrinsic calculi – examples

- $\lambda x.\, x$ – the identity function. It can be assigned many types, like $\texttt{Int} \to \texttt{Int}$ or $\texttt{String} \to \texttt{String}$, so there's no need to have many identity functions for different types.
- $(\lambda x : \texttt{String}.\, x)\ 5$ – this is a term, but it cannot be assigned a type because of a type-error.

Extrinsic calculi

In extrinsic calculi, terms and types are separate beings and the relationship between them is established by the typing judgement. Ill-typed terms are terms, they just happen to be dangerous, so they're forbidden by the type checker. Terms that can be assigned many types are ok and are not forbidden, but cherished, so in some sense types are not disjoint.

In practice

In practice, purely intrinsic languages become hard to use as soon as they have polymorphism, because the amount of type annotations becomes unwieldy.

In practice, purely extrinsic languages do not have decidable type inference (and thus also type checking) as soon as they have polymorphism, because there are not enough type annotations.

In practice, most languages are a hybrid of both approaches. Proof assistants are more intrinsic, with features like implicit arguments which mimic extrinsic language constructs. Programming languages are more extrinsic, with more or less optional type annotations in various places, which make type checking and inference tractable.

Plan

Since practical languages are hybrids driven by practical needs, we will devote the rest of this talk to the study of (some) type checking and type inference algorithms, culminating in the description of hint-based typing, or hinting, as I call it. To keep things simple, we will only look at the Simply Typed Lambda Calculus (STLC). But before we see the final boss, STLC with Hints, we will take a look at four more variants of STLC to learn where our presentation is coming from:

- Extrinsic STLC – the most basic variant of STLC.
- Bidirectional STLC – a nice variant of STLC with intuitive type checking and type inference algorithms.
- Intrinsic STLC – an old an venerable variant, in which complete type inference is particularly easy.
- Dual Intrinsic STLC – a crazy and obscure cousin of the above, in which type checking is particualrly easy.

Types, contexts and judgements

All variants of STLC presented in these slides will have the same types, contexts and declarative typing judgement. The only differences will be terms and possibly additional judgements.

Types:
$A, B ::= A \to B \mid A \times B \mid A + B \mid \mathbf{1} \mid \mathbf{0}$

Typing contexts:
$\Gamma ::= \cdot \mid \Gamma, x : A$

Judgements:
$\Gamma \vdash e : A$

Types, contexts and judgements – explanations

As for types, we have function types $A \to B$, (binary) product types $A \times B$, (binary) sum types $A + B$, the unit type $\mathbf{1}$ and the empty type $\mathbf{0}$.

Contexts are either empty (denote by the symbol $\cdot$) or they are another context Γ extended with a typing declaration $x : A$ (denoted by $\Gamma, x : A$).

The judgement $\Gamma \vdash e : A$ means that in context Γ, the term e has type A.

Extrinsic STLC

Extrinsic STLC is the simplest version of typed lambda calculus. The terms are the same as in untyped lambda calculus (with the addition of terms for products, sums, unit and empty). The typing relation takes the form of a type assignment system – we describe which terms have which types, without worrying about issues such as implementation.

Terms

Terms:

$e ::=$

$x \mid$

$\lambda x.\, e \mid e_1\ e_2 \mid$

$(e_1, e_2) \mid \texttt{outl}\ e \mid \texttt{outr}\ e \mid$

$\texttt{inl}\ e \mid \texttt{inr}\ e \mid \texttt{case}\ e\ \texttt{of}\ (e_1, e_2) \mid$

$\texttt{unit} \mid \mathbf{0}\texttt{-elim}\ e$

Note: the terms are, in order of appearance, variables, functions, applications, pairs, left and right projections, left and right sum constructors, pattern matching for sums, a value of the unit type, and the eliminator for the empty type.

Example (big-step) semantics – values

$$\frac{}{\lambda x.\, e \texttt{ value}}$$

$$\frac{v_1 \texttt{ value} \quad v_2 \texttt{ value}}{(v_1, v_2) \texttt{ value}}$$

$$\frac{v \texttt{ value}}{\texttt{inl } v \texttt{ value}} \qquad \frac{v \texttt{ value}}{\texttt{inr } v \texttt{ value}}$$

$$\frac{}{\texttt{unit value}}$$

Values are the final results of computation. Note that a function is a value whether or not its body is. Other values are pairs of values, values injected into a sum on the left or right, and unit.

Example (big-step) semantics – evaluation

$$\frac{}{\lambda x.\, e \Downarrow \lambda x.\, e} \qquad \frac{e_1 \Downarrow \lambda x.\, e \quad e_2 \Downarrow v \quad e\,[x := v] \Downarrow v'}{e_1\ e_2 \Downarrow v'}$$

$$\frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{(e_1, e_2) \Downarrow (v_1, v_2)} \qquad \frac{e \Downarrow (v_1, v_2)}{\texttt{outl}\ e \Downarrow v_1} \qquad \frac{e \Downarrow (v_1, v_2)}{\texttt{outr}\ e \Downarrow v_2}$$

$$\frac{e \Downarrow v}{\texttt{inl}\ e \Downarrow \texttt{inl}\ v} \qquad \frac{e \Downarrow v}{\texttt{inr}\ e \Downarrow \texttt{inr}\ v}$$

$$\frac{e \Downarrow \texttt{inl}\ v \quad f\ v \Downarrow v'}{\texttt{case}\ e\ \texttt{of}\ (f, g) \Downarrow v'} \qquad \frac{e \Downarrow \texttt{inr}\ v \quad g\ v \Downarrow v'}{\texttt{case}\ e\ \texttt{of}\ (f, g) \Downarrow v'}$$

$$\frac{}{\texttt{unit} \Downarrow \texttt{unit}}$$

Example (small-step) semantics – basic rules

$$\frac{v\ \texttt{value}}{(\lambda x.\, e)\ v \longrightarrow e\,[x := v]}$$

$$\frac{v_1\ \texttt{value} \quad v_2\ \texttt{value}}{\texttt{outl}\ (v_1, v_2) \longrightarrow v_1} \qquad \frac{v_1\ \texttt{value} \quad v_2\ \texttt{value}}{\texttt{outr}\ (v_1, v_2) \longrightarrow v_2}$$

$$\frac{v\ \texttt{value}}{\texttt{case}\ (\texttt{inl}\ v)\ \texttt{of}\ (f, g) \longrightarrow f\ v}$$

$$\frac{v\ \texttt{value}}{\texttt{case}\ (\texttt{inr}\ v)\ \texttt{of}\ (f, g) \longrightarrow g\ v}$$

Example (small-step) semantics – boring rules

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2} \qquad \frac{v_1\ \texttt{value} \quad e_2 \longrightarrow e_2'}{v_1\ e_2 \longrightarrow v_1\ e_2'}$$

$$\frac{e_1 \longrightarrow e_1'}{(e_1, e_2) \longrightarrow (e_1', e_2)} \qquad \frac{v_1\ \texttt{value} \quad e_2 \longrightarrow e_2'}{(v_1, e_2) \longrightarrow (v_1, e_2')}$$

$$\frac{e \longrightarrow e'}{\texttt{outl}\ e \longrightarrow \texttt{outl}\ e'} \qquad \frac{e \longrightarrow e'}{\texttt{outr}\ e \longrightarrow \texttt{outr}\ e'}$$

$$\frac{e \longrightarrow e'}{\texttt{inl}\ e \longrightarrow \texttt{inl}\ e'} \qquad \frac{e \longrightarrow e'}{\texttt{inr}\ e \longrightarrow \texttt{inr}\ e'}$$

$$\frac{e \longrightarrow e'}{\texttt{case}\ e\ \texttt{of}\ (f, g) \longrightarrow \texttt{case}\ e'\ \texttt{of}\ (f, g)}$$

Example semantics – explanations

$e \Downarrow v$ should be read "term e evalutes to value v". It describes computation in a coarse-grained manner, telling us what is the result of evaluating each term. If $e \Downarrow v$, then v `value`.

$e \longrightarrow e'$ should be read "term e reduces to term e'". It describes computation in a fine-grained manner, telling us what can happen in a single computation step. To actually describe computation fully in this style, we need to take the transitive closure of this relation, written $e \longrightarrow^* v$.

Both presented semantics are call-by-value, i.e. we evaluate a function's argument before performing a substitution. They are equivalent, i.e. $e \Downarrow v$ if and only if $e \longrightarrow^* v$

Declarative typing – basics

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}\,\textsc{Var}$$

Declarative typing – type-directed rules

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x.\, e : A \to B} \qquad \frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash f\ a : B}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B} \qquad \frac{\Gamma \vdash e : A \times B}{\Gamma \vdash \mathtt{outl}\ e : A} \qquad \frac{\Gamma \vdash e : A \times B}{\Gamma \vdash \mathtt{outr}\ e : B}$$

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash \mathtt{inl}\ e : A + B} \qquad \frac{\Gamma \vdash e : B}{\Gamma \vdash \mathtt{inr}\ e : A + B}$$

$$\frac{\Gamma \vdash e : A + B \quad \Gamma \vdash f : A \to C \quad \Gamma \vdash g : B \to C}{\Gamma \vdash \mathtt{case}\ e\ \mathtt{of}\ (f, g) : C}$$

$$\frac{}{\Gamma \vdash \mathtt{unit} : \mathbf{1}} \qquad \frac{\Gamma \vdash e : \mathbf{0}}{\Gamma \vdash \mathbf{0}\text{-}\mathtt{elim}\ e : A}$$

Metatheory

Extrinsic STLC enjoys strong metatheoretical properties:

- Confluence: if a term can be reduced to two different terms, these two can in turn be reduced to a common result.
- Termination: computation on well-typed terms always terminates.
- Type preservation: if we compute with a well-typed term, the result has the same type.
- Canonicity: in the empty context, normal forms are inductively generated from term constructors.

Intuitively: given a well-typed term, in finite time it computes to another term of the same type which cannot compute anymore. In the empty context, we know the result of this computation must be a constructor.

(Non)uniqueness of typing

Despite these strong metatheoretic properties, Extrinsic STLC does not enjoy another important property: **uniqueness of typing**. This means that there are terms which can be assigned multiple types. For example, $\lambda x.\, x$ can be assigned the type $A \to A$ for any type A. The four culprits of this failure are lambda abstractions, sum constructors and exfalso.

Bidirectional STLC

Bidirectional STLC is a version of simply typed lambda calculus which focuses on a clean implementation of the type checker. The terms are as in Extrinsic STLC, but with the addition of a general type annotation construct that can appear anywhere and is not mandatory. The typing judgement is split into two: type checking and type inference, both of which are algorithmic, i.e. easily implementable.

Terms and judgements

Terms:

$e ::=$

$x \mid (e : A) \mid$
$\lambda x.\, e \mid e_1\ e_2 \mid$
$(e_1, e_2) \mid \texttt{outl}\ e \mid \texttt{outr}\ e \mid$
$\texttt{inl}\ e \mid \texttt{inr}\ e \mid \texttt{case}\ e\ \texttt{of}\ (e_1, e_2) \mid$
$\texttt{unit} \mid \mathbf{0}\texttt{-elim}\ e$

Note: green color marks terms which were not present in Extrinsic STLC.

Judgements:

$\Gamma \vdash e \Leftarrow A$ – in context Γ, term e checks against type A (A is an input)

$\Gamma \vdash e \Rightarrow A$ – in context Γ, term e infers type A (A is an output)

Declarative typing – new rules

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash (e : A) : A}\,\textsc{Annot}$$

Note: the only rules shown are those which were not present in Extrinsic STLC.

Bidirectional typing – basics

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x \Rightarrow A}\textsc{Var}$$

$$\frac{\Gamma \vdash e \Leftarrow A}{\Gamma \vdash (e : A) \Rightarrow A}\textsc{Annot}$$

$$\frac{\Gamma \vdash e \Rightarrow B \quad A = B}{\Gamma \vdash e \Leftarrow A}\textsc{Sub}$$

Bidirectional typing – type-directed rules

$$\frac{\Gamma, x : A \vdash e \Leftarrow B}{\Gamma \vdash \lambda x.\, e \Leftarrow A \to B} \qquad \frac{\Gamma \vdash f \Rightarrow A \to B \quad \Gamma \vdash a \Leftarrow A}{\Gamma \vdash f\ a \Rightarrow B}$$

$$\frac{\Gamma \vdash a \Leftarrow A \quad \Gamma \vdash b \Leftarrow B}{\Gamma \vdash (a, b) \Leftarrow A \times B} \qquad \frac{\Gamma \vdash e \Rightarrow A \times B}{\Gamma \vdash \texttt{outl}\ e \Rightarrow A} \qquad \frac{\Gamma \vdash e \Rightarrow A \times B}{\Gamma \vdash \texttt{outr}\ e \Rightarrow B}$$

$$\frac{\Gamma \vdash e \Leftarrow A}{\Gamma \vdash \texttt{inl}\ e \Leftarrow A + B} \qquad \frac{\Gamma \vdash e \Leftarrow B}{\Gamma \vdash \texttt{inr}\ e \Leftarrow A + B}$$

$$\frac{\Gamma \vdash e \Rightarrow A + B \quad \Gamma \vdash f \Leftarrow A \to C \quad \Gamma \vdash g \Leftarrow B \to C}{\Gamma \vdash \texttt{case}\ e\ \texttt{of}\ (f, g) \Leftarrow C}$$

$$\frac{}{\Gamma \vdash \texttt{unit} \Leftarrow \mathbf{1}} \qquad \frac{\Gamma \vdash e \Leftarrow \mathbf{0}}{\Gamma \vdash \mathbf{0}\texttt{-elim}\ e \Leftarrow A}$$

Bidirectional typing – additional rules

$$\frac{}{\Gamma \vdash \texttt{unit} \Rightarrow \mathbf{1}}$$

$$\frac{\Gamma \vdash e \Rightarrow A + B \quad \Gamma \vdash f \Rightarrow A \to C \quad \Gamma \vdash g \Rightarrow B \to C}{\Gamma \vdash \texttt{case}\ e\ \texttt{of}\ (f, g) \Rightarrow C}$$

$$\frac{\Gamma \vdash a \Rightarrow A \quad \Gamma \vdash b \Rightarrow B}{\Gamma \vdash (a, b) \Rightarrow A \times B}$$

The basic rules are as presented in the previous slide. However, it is possible to add some enhancements. First, we can replace the rule for `unit` with an inference rule and if we need to check, we can use subsumption. Second, the paper argues that sum elimination is a general principle, and so it should allow both checking and inference versions. We could also add an inference rule for pairs. It seems there isn't a trade-off either – if the checking rule fails, we can use subsumption and try to infer.

(Non)uniqueness of typing

Similarly to Extrinsic STLC, typing is not unique in Bidirectional STLC. This is because while we do have annotations in terms, we are not forced to use them. Therefore, we can check terms like $\lambda x.\, x$ with any type of the form $A \to A$. However, inference is unique.

Bidirectional typing – how the sausage is made

We did not describe how to derive the algorithmic rules of Bidirectional STLC from the declarative rules of Extrinsic STLC. This "bidirectional recipe" can be found in the paper Bidirectional Typing, section 4. Sections 1-4 provide a nice introduction to the topic of bidirectional typing.

Bidirectional Typing Rules: A Tutorial is a nice tutorial paper which shows how to implement a bidirectional typechecker for STLC with booleans (but no products or sums).

Intrinsic STLC

Intrinsic STLC is the most widespread version of simply typed lambda calculus. The terms differ somewhat from Extrinsic STLC, as type annotations are mandatory on lambdas, sum constructors and the empty type eliminator, but the typing judgement remains the same.

Terms

Terms:

$e ::=$

$x \mid$

$\lambda x : A.\, e \mid e_1\ e_2 \mid$

$(e_1, e_2) \mid \texttt{outl}\ e \mid \texttt{outr}\ e \mid$

$\texttt{inl}_B\ e \mid \texttt{inr}_A\ e \mid \texttt{case}\ e\ \texttt{of}\ (e_1, e_2) \mid$

$\texttt{unit} \mid \mathbf{0}\texttt{-elim}_A\ e$

Note: red color marks places which differ from Extrinsic STLC.

Declarative typing – differences

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x : A.\, e : A \to B}$$

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash \mathtt{inl}_B\ e : A + B} \qquad \frac{\Gamma \vdash e : B}{\Gamma \vdash \mathtt{inr}_A\ e : A + B}$$

$$\frac{\Gamma \vdash e : \mathbf{0}}{\Gamma \vdash \mathbf{0}\text{-}\mathtt{elim}_A\ e : A}$$

Note: the only rules shown are those that differ from Extrinsic STLC.

Type inference – basics

Thanks to abundant type annotations, we can (re)interpret the typing judgement $\Gamma \vdash e : A$ as a type inference judgement $\Gamma \vdash e \Rightarrow A$.

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x \Rightarrow A}\,\textsc{Var}$$

Type inference – type-directed rules

$$\frac{\Gamma, x : A \vdash e \Rightarrow B}{\Gamma \vdash \lambda x : A.\, e \Rightarrow A \to B} \qquad \frac{\Gamma \vdash f \Rightarrow A \to B \quad \Gamma \vdash a \Rightarrow A}{\Gamma \vdash f\ a \Rightarrow B}$$

$$\frac{\Gamma \vdash a \Rightarrow A \quad \Gamma \vdash b \Rightarrow B}{\Gamma \vdash (a, b) \Rightarrow A \times B} \qquad \frac{\Gamma \vdash e \Rightarrow A \times B}{\Gamma \vdash \mathtt{outl}\ e \Rightarrow A} \qquad \frac{\Gamma \vdash e \Rightarrow A \times B}{\Gamma \vdash \mathtt{outr}\ e \Rightarrow B}$$

$$\frac{\Gamma \vdash e \Rightarrow A}{\Gamma \vdash \mathtt{inl}_B\ e \Rightarrow A + B} \qquad \frac{\Gamma \vdash e \Rightarrow B}{\Gamma \vdash \mathtt{inr}_A\ e \Rightarrow A + B}$$

$$\frac{\Gamma \vdash e \Rightarrow A + B \quad \Gamma \vdash f \Rightarrow A \to C \quad \Gamma \vdash g \Rightarrow B \to C}{\Gamma \vdash \mathtt{case}\ e\ \mathtt{of}\ (f, g) \Rightarrow C}$$

$$\frac{}{\Gamma \vdash \mathtt{unit} \Rightarrow \mathbf{1}} \qquad \frac{\Gamma \vdash e \Rightarrow \mathbf{0}}{\Gamma \vdash \mathbf{0}\text{-}\mathtt{elim}_A\ e \Rightarrow A}$$

Uniqueness of typing

Because of the annotations on lambda, sum constructors and exfalso, Intrinsic STLC does enjoy uniqueness of typing. It is easy to prove this by induction: types of most terms are determined by the induction hypothesis, whereas for the aforementioned four we need to supplement the induction hypothesis with the annotation.

Dual Intrinsic STLC

If the usual Intrinsic STLC turns out to be a system in which we can infer all types, what would a system in which we can check all types look like?

Terms

Terms:

$e ::=$

$x \mid$

$\lambda x.\, e \mid \mathtt{app}_A\ e_1\ e_2 \mid$

$(e_1, e_2) \mid \mathtt{outl}_B\ e \mid \mathtt{outr}_A\ e \mid$

$\mathtt{inl}\ e \mid \mathtt{inr}\ e \mid \mathtt{case}_{A,B}\ e\ \mathtt{of}\ (e_1, e_2) \mid$

$\mathtt{unit} \mid \mathbf{0}\text{-}\mathtt{elim}\ e$

Note: red color marks places which differ from Extrinsic STLC.

Declarative typing – differences

$$\frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash \mathtt{app}_A\ f\ a : B}$$

$$\frac{\Gamma \vdash e : A \times B}{\Gamma \vdash \mathtt{outl}_B\ e : A} \qquad \frac{\Gamma \vdash e : A \times B}{\Gamma \vdash \mathtt{outr}_A\ e : B}$$

$$\frac{\Gamma \vdash e : A + B \quad \Gamma \vdash f : A \to C \quad \Gamma \vdash g : B \to C}{\Gamma \vdash \mathtt{case}_{A,B}\ e\ \mathtt{of}\ (f, g) : C}$$

Note: the only rules shown are those that differ from Extrinsic STLC.

Type checking – basics

Thanks to abundant type annotations, we can (re)interpret the typing judgement $\Gamma \vdash e : A$ as a type checking judgement $\Gamma \vdash e \Leftarrow A$.

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x \Leftarrow A}\,\textsc{Var}$$

Type checking – type-directed rules

$$\frac{\Gamma, x : A \vdash e \Leftarrow B}{\Gamma \vdash \lambda x.\, e \Leftarrow A \to B} \qquad \frac{\Gamma \vdash f \Leftarrow A \to B \quad \Gamma \vdash a \Leftarrow A}{\Gamma \vdash \mathtt{app}_A\ f\ a \Leftarrow B}$$

$$\frac{\Gamma \vdash a \Leftarrow A \quad \Gamma \vdash b \Leftarrow B}{\Gamma \vdash (a, b) \Leftarrow A \times B} \qquad \frac{\Gamma \vdash e \Leftarrow A \times B}{\Gamma \vdash \mathtt{outl}_B\ e \Leftarrow A} \qquad \frac{\Gamma \vdash e \Leftarrow A \times B}{\Gamma \vdash \mathtt{outr}_A\ e \Leftarrow B}$$

$$\frac{\Gamma \vdash e \Leftarrow A}{\Gamma \vdash \mathtt{inl}\ e \Leftarrow A + B} \qquad \frac{\Gamma \vdash e \Leftarrow B}{\Gamma \vdash \mathtt{inr}\ e \Leftarrow A + B}$$

$$\frac{\Gamma \vdash e \Leftarrow A + B \quad \Gamma \vdash f \Leftarrow A \to C \quad \Gamma \vdash g \Leftarrow B \to C}{\Gamma \vdash \mathtt{case}_{A,B}\ e\ \mathtt{of}\ (f, g) \Leftarrow C}$$

$$\frac{}{\Gamma \vdash \mathtt{unit} \Leftarrow \mathbf{1}} \qquad \frac{\Gamma \vdash e \Leftarrow \mathbf{0}}{\Gamma \vdash \mathbf{0}\text{-}\mathtt{elim}\ e \Leftarrow A}$$

(Non)uniqueness of typing

Even though Dual Intrinsic STLC has plenty of mandatory annotations, it does not enjoy uniqueness of typing for the usual reasons: we can type $\lambda x.\, x$ with any type of the form $A \to A$. The role of the annotations is not to force types to be unique, but to make it possible to implement type checking.

STLC with Hints

STLC with Hints is a flavour of STLC inspired by Bidirectional STLC. The main insight behind it is that in Bidirectional STLC, we have a hard time deciding whether a rule should be in checking mode or in inference mode, so why not both? This way, we would have some input type that guides us, but also produce an output type, which is in some sense "better". This is a bit silly if we already have the correct type as input, but we can make this idea work by introducing hints, which are types with holes, and insisting that the input is not a type, but merely a hint.

Hints

$$H ::= \mathbf{?} \mid H_1 \to H_2 \mid H_1 \times H_2 \mid H_1 + H_2 \mid \mathbf{1} \mid \mathbf{0}$$

Intuitively, hints are partial types. They are built like types, except that there's one additional constructor, **?**, which can be read as "hole" or "unknown".

We use the letter H for hints. When we use letters like A, B, C which usually stand in for types, it means that the hint **is** a type, i.e. it doesn't contain any **?**s.

Order on hints

$$\frac{}{\mathbf{?} \sqsubseteq H}$$

$$\frac{H_1 \sqsubseteq H_1' \quad H_2 \sqsubseteq H_2'}{H_1 \to H_2 \sqsubseteq H_1' \to H_2'}$$

$$\frac{H_1 \sqsubseteq H_1' \quad H_2 \sqsubseteq H_2'}{H_1 \times H_2 \sqsubseteq H_1' \times H_2'}$$

$$\frac{H_1 \sqsubseteq H_1' \quad H_2 \sqsubseteq H_2'}{H_1 + H_2 \sqsubseteq H_1' + H_2'}$$

$$\frac{}{\mathbf{1} \sqsubseteq \mathbf{1}} \qquad \frac{}{\mathbf{0} \sqsubseteq \mathbf{0}}$$

Order on hints – intuition

The order can be intuitively interpreted as information increase: $H_1 \sqsubseteq H_2$ means that hint H_2 is more informative than H_1, but in a compatible way. In other words, H_1 and H_2 have the same structure, but some **?**s from H_1 were possibly refined to something more informative in H_2.

Combining hints

$$\mathbf{?} \sqcup H = H$$
$$H \sqcup \mathbf{?} = H$$
$$(H_1 \to H_2) \sqcup (H_1' \to H_2') = (H_1 \sqcup H_1') \to (H_2 \sqcup H_2')$$
$$(H_1 \times H_2) \sqcup (H_1' \times H_2') = (H_1 \sqcup H_1') \times (H_2 \sqcup H_2')$$
$$(H_1 + H_2) \sqcup (H_1' + H_2') = (H_1 \sqcup H_1') + (H_2 \sqcup H_2')$$
$$\mathbf{1} \sqcup \mathbf{1} = \mathbf{1}$$
$$\mathbf{0} \sqcup \mathbf{0} = \mathbf{0}$$

The order on hints induces a partial operation $\sqcup$, which computes the least upper bound of two hints when it exists. Intuitively, $\sqcup$ combines two hints which share the same structure, filling the **?**s in the leaves with something more informative coming from the other argument. For hints with incompatible structure the result is undefined.

Combining hints – properties

If all relevant results are defined, then:

- $(H_1 \sqcup H_2) \sqcup H_3 = H_1 \sqcup (H_2 \sqcup H_3)$
- $H_1 \sqcup H_2 = H_2 \sqcup H_1$
- $\mathbf{?} \sqcup H = H = H \sqcup \mathbf{?}$
- $H \sqcup H = H$

If $\sqcup$ were not partial, $(H, \sqcup, \mathbf{?})$ would be a commutative idempotent monoid. But since it is partial, meh...

Hints for term constructors

$$\begin{aligned}
\texttt{hint}(\lambda x.\, e) &= \texttt{?} \to \texttt{?} \\
\texttt{hint}((e_1, e_2)) &= \texttt{?} \times \texttt{?} \\
\texttt{hint}(\texttt{inl}\ e) &= \texttt{?} + \texttt{?} \\
\texttt{hint}(\texttt{inr}\ e) &= \texttt{?} + \texttt{?} \\
\texttt{hint}(\texttt{unit}) &= \mathbf{1}
\end{aligned}$$

Terms

Terms:

$e ::=$

$x \mid (e : H) \mid$

$\lambda x.\, e \mid e_1\ e_2 \mid$

$(e_1, e_2) \mid \texttt{outl}\ e \mid \texttt{outr}\ e \mid$

$\texttt{inl}\ e \mid \texttt{inr}\ e \mid \texttt{case}\ e\ \texttt{of}\ (e_1, e_2) \mid$

$\texttt{unit} \mid \mathbf{0}\texttt{-elim}\ e$

Note: red color marks differences from Bidirectional STLC.

Judgements:

$\Gamma \vdash e \Leftarrow H \Rightarrow A$ – in context Γ, term e checks with hint H and infers type A

Declarative typing – differences

$$\frac{\Gamma \vdash e : A \quad H \sqsubseteq A}{\Gamma \vdash (e : H) : A}\ \textsc{Annot}$$

Hinting – basic rules

$$\frac{(x : A) \in \Gamma \quad H \sqsubseteq A}{\Gamma \vdash x \Leftarrow H \Rightarrow A}\textsc{Var}$$

$$\frac{\Gamma \vdash e \Leftarrow H_1 \sqcup H_2 \Rightarrow A}{\Gamma \vdash (e : H_1) \Leftarrow H_2 \Rightarrow A}\textsc{Annot}$$

$$\frac{\Gamma \vdash e \Leftarrow \texttt{hint}(e) \Rightarrow A \quad e\ \texttt{constructor}}{\Gamma \vdash e \Leftarrow \texttt{?} \Rightarrow A}\textsc{Hole}$$

Note that the rule Hole can only be applied once, because $\texttt{hint}(e)$ can never be **?**. After applying Hole, the only applicable rules are the type-directed ones.

Hinting – type-directed rules

$$\frac{\Gamma, x : A \vdash e \Leftarrow H \Rightarrow B}{\Gamma \vdash \lambda x.\, e \Leftarrow A \to H \Rightarrow A \to B}$$

$$\frac{\Gamma \vdash f \Leftarrow \mathbf{?} \to H \Rightarrow A \to B \quad \Gamma \vdash a \Leftarrow A \Rightarrow A}{\Gamma \vdash f\ a \Leftarrow H \Rightarrow B}$$

$$\frac{\Gamma \vdash a \Leftarrow H_1 \Rightarrow A \quad \Gamma \vdash b \Leftarrow H_2 \Rightarrow B}{\Gamma \vdash (a, b) \Leftarrow H_1 \times H_2 \Rightarrow A \times B}$$

$$\frac{\Gamma \vdash e \Leftarrow H \times \mathbf{?} \Rightarrow A \times B}{\Gamma \vdash \texttt{outl}\ e \Leftarrow H \Rightarrow A} \qquad \frac{\Gamma \vdash e \Leftarrow \mathbf{?} \times H \Rightarrow A \times B}{\Gamma \vdash \texttt{outr}\ e \Leftarrow H \Rightarrow B}$$

Hinting – type-directed rules

$$\frac{\Gamma \vdash e \Leftarrow H \Rightarrow A}{\Gamma \vdash \texttt{inl}\ e \Leftarrow H + B \Rightarrow A + B}$$

$$\frac{\Gamma \vdash e \Leftarrow H \Rightarrow B}{\Gamma \vdash \texttt{inr}\ e \Leftarrow A + H \Rightarrow A + B}$$

$$\frac{\Gamma \vdash e \Leftarrow \mathbf{?} + \mathbf{?} \Rightarrow A + B \qquad \begin{array}{c}\Gamma \vdash f \Leftarrow A \to H \Rightarrow A \to C \\ \Gamma \vdash g \Leftarrow B \to C \Rightarrow B \to C\end{array}}{\Gamma \vdash \texttt{case}\ e\ \texttt{of}\ (f, g) \Leftarrow H \Rightarrow C}$$

$$\frac{}{\Gamma \vdash \texttt{unit} \Leftarrow \mathbf{1} \Rightarrow \mathbf{1}} \qquad \frac{\Gamma \vdash e \Leftarrow \mathbf{0} \Rightarrow \mathbf{0}}{\Gamma \vdash \mathbf{0}\texttt{-elim}\ e \Leftarrow A \Rightarrow A}$$

Hinting – alternative rules

$$\frac{\Gamma \vdash a \Leftarrow ? \Rightarrow A \quad \Gamma \vdash f \Leftarrow A \to H \Rightarrow A \to B}{\Gamma \vdash f\ a \Leftarrow H \Rightarrow B}\ \textsc{AltApp}$$

$$\frac{\begin{array}{l}\Gamma \vdash f \Leftarrow ? \to H \Rightarrow A \to C \\ \Gamma \vdash g \Leftarrow ? \to C \Rightarrow B \to C\end{array} \qquad \Gamma \vdash e \Leftarrow A + B \Rightarrow A + B}{\Gamma \vdash \mathtt{case}\ e\ \mathtt{of}\ (f, g) \Leftarrow H \Rightarrow C}\ \textsc{AltCase}$$

We could have made some different choices. For application, we could try to infer the argument type first and then feed it to the function as a hint. For case, we could try to infer domains of the branches first, then feed these as hints when checking the discriminee.

Notations and derived terms

We can introduce some handy notations:

- $\Gamma \vdash e \Leftarrow A :\equiv \Gamma \vdash e \Leftarrow A \Rightarrow A$
- $\Gamma \vdash e \Rightarrow A :\equiv \Gamma \vdash e \Leftarrow ? \Rightarrow A$

We can embed Intrinsic STLC terms:

- $\lambda x : A.\, e :\equiv (\lambda x.\, e : A \to ?)$
- $\mathtt{inl}_B\ e :\equiv (\mathtt{inl}\ e : ? + B)$
- $\mathtt{inr}_A\ e :\equiv (\mathtt{inr}\ e : A + ?)$
- $\mathbf{0}\text{-}\mathtt{elim}_A\ e :\equiv (\mathbf{0}\text{-}\mathtt{elim}\ e : A)$

We can also embed Dual Intrinsic STLC terms:

- $\mathtt{app}_A\ f\ a :\equiv (f : A \to ?)\ a$
- $\mathtt{outl}_B\ e :\equiv \mathtt{outl}\ (e : ? \times B)$
- $\mathtt{outr}_A\ e :\equiv \mathtt{outr}\ (e : A \times ?)$
- $\mathtt{case}_{A,B}\ e\ \mathtt{of}\ (f, g) :\equiv \mathtt{case}\ (e : A + B)\ \mathtt{of}\ (f, g)$

Rules for derived terms

$$\frac{\Gamma, x : A \vdash e \Rightarrow B}{\Gamma \vdash \lambda x : A.\, e \Rightarrow A \to B} \qquad \frac{\Gamma \vdash f \Leftarrow A \to B \quad \Gamma \vdash a \Leftarrow A}{\Gamma \vdash \mathtt{app}_A\ f\ a \Leftarrow B}$$

$$\frac{\Gamma \vdash e \Leftarrow A \times B}{\Gamma \vdash \mathtt{outl}_B\ e \Leftarrow A} \qquad \frac{\Gamma \vdash e \Leftarrow A \times B}{\Gamma \vdash \mathtt{outr}_A\ e \Leftarrow B}$$

$$\frac{\Gamma \vdash e \Rightarrow A}{\Gamma \vdash \mathtt{inl}_B\ e \Rightarrow A + B} \qquad \frac{\Gamma \vdash e \Rightarrow B}{\Gamma \vdash \mathtt{inr}_A\ e \Rightarrow A + B}$$

$$\frac{\Gamma \vdash e \Leftarrow A + B \quad \Gamma \vdash f \Leftarrow A \to C \quad \Gamma \vdash g \Leftarrow B \to C}{\Gamma \vdash \mathtt{case}_{A,B}\ e\ \mathtt{of}\ (f, g) \Leftarrow C}$$

$$\frac{\Gamma \vdash e \Rightarrow \mathbf{0}}{\Gamma \vdash \mathbf{0}\text{-}\mathtt{elim}_A\ e \Rightarrow A}$$

(Non)uniqueness of typing

Similarly to Extrinsic STLC, STLC with Hints does not enjoy uniqueness of typing. This is because we still can have terms like $\lambda x.\, x$ with hint **?**, which can be typed with any type of the form $A \to A$. However, if the hint is informative enough, then the type is unique. Moreover, every typable term can be given a hint which makes its type unique.