Frozen inference constraints for type-directed disambiguation

Olivier Martinot, Gabriel Scherer

Partout, Inria Saclay, France

August 21, 2021


Type-directed disambiguation

Many language support type-directed disambiguation of names.
How to combine this with type inference?

Type-directed disambiguation

Many language support type-directed disambiguation of names.
How to combine this with type inference?

Type classes (qualified types):
nice inference through *constraint abstraction*
excellent approach for operator overloading.

Type-directed disambiguation outside qualified types

A feature where type classes are not enough:
data constructor disambiguation.

`f (`K t`)`

`match` t `with` K `x ->` u

Type-directed disambiguation outside qualified types

A feature where type classes are not enough:
data constructor disambiguation.

f (K t)
match t with K x -> u

1. We do not want to *abstract* over K.
2. The type of K may not be expressible as a class argument (existentials, etc.; data constructors are not functions.)
3. Different constructors K may have vastly different typing rules.

Constructor disambiguation and type inference

`f` (K t)
`match` t `with` K `x ->` u

Need program types to disambiguate K.
Need the type of K to infer program types.

HM type inference:
propagation by unification (within generalization boundaries).

Bidirectional type inference (commonly used for disambiguation):
leafward propagation from annotations (robust)
$+$ some lateral propagation (fragile): $t\ u$

This Work In Progress explores unification-based type disambiguation
frozen constraints.

Constraint-based type inference: a primer

$$\text{implicitly-typed } t \overset{\text{generate}}{\Longrightarrow} \text{constraint } C \overset{\text{solve}}{\Longrightarrow} \text{explicitly-typed } t'$$

Constraint for application $t\ u$ with return type variable α:

$$[\![t\ u]\!]_\alpha \overset{\text{def}}{=} \exists \beta_t . \exists \gamma_u . \left((\beta_t = \gamma_u \to \alpha) \ \wedge \ [\![t]\!]_{\beta_t} \ \wedge \ [\![u]\!]_{\gamma_u} \right)$$

Frozen constraints

$$\langle\alpha\rangle f$$

α: type inference variable
f: function from partial types to constraints

waits on a *type unification variable* α:
when α becomes (partly) defined as τ,
the constraint $f(\tau)$ must be solved.

Constructor constraint (non-GADT case):

$$[\![K\ t]\!]_\alpha \stackrel{\text{def}}{=} \exists\beta_t.\ ([\![t]\!]_{\beta_t} \wedge \langle\alpha\rangle(\lambda\tau.\ \beta_t = \mathsf{arg\_type}(\tau, K)))$$

Principled (and principal) inference with type-disambiguation.
(Maybe too restrictive?)

Difficult to combine with generalization!

Practical difficulty: generalization (1/2)

If $\langle \alpha \rangle f$ remains unsolved "at the end", type inference fails.

But when is the end?

Practical difficulty: generalization (1/2)

If $\langle\alpha\rangle f$ remains unsolved "at the end", type inference fails.

But when is the end?

How does $\langle\alpha\rangle f$ interact with `let`-generalization?

Practical difficulty: generalization (2/2)

Generalization: which inference variables α are *local* and and can be generalized into polymorphic variables?

Practical difficulty: generalization (2/2)

Generalization: which inference variables α are *local* and and can be generalized into polymorphic variables?

Frozen generalization of τ:
if a variable β of τ is "blocked" by a frozen constraint,
it must be tracked during instantiation and possibly generalized later.

Partially-frozen schemas:

- On generalization: store β as a blocked schema variable.
- On instantiation: track the instance of the partially-frozen schema.
- When β gets unblocked: continue generalization, update tracked instances.

Delicate to implement. Difficult to implement efficiently.

Theoretical difficulty: semantics (1/3)

Constraints are given meaning by a *solution relation* $V \Vdash C$.

A good constraint generator has correct solutions.

A good constraint solver (big-step function or small-step rewrites) preserves solutions.

$$\frac{\tau[V] =_{\mathsf{ty}} \tau'[V]}{V \Vdash \tau = \tau'} \qquad\qquad \frac{V \Vdash C[T/\alpha]}{(T, V) \Vdash \exists\alpha.\, C}$$

How to specify frozen constraints?

Theoretical difficulty: semantics (2/3)

Natural approach:

$$\frac{V \Vdash f(\alpha[V])}{V \Vdash \langle\alpha\rangle f}$$

This specification allows "out of thin air" behaviors.

$$[\alpha \mapsto \mathsf{int}] \Vdash \langle\alpha\rangle(\lambda\tau.\, \alpha = \mathsf{int})$$

Our solver does not: the specification is not precise enough.

Theoretical difficulty: semantics (3/3)

We want to express that $\alpha[V]$ is determined "without looking inside f".
How can we do this?

Morally:

$$\frac{C[\top] \text{ determines } \alpha \qquad V \Vdash C[f(\alpha[V])]}{V \Vdash C[\langle\alpha\rangle f]}$$

Summary

Frozen constraints: interesting but difficult constraint combinator.

Work in progress.

Thanks! Questions?