Mechanizing an elaboration algorithm for the Hindley-Damas-Milner type system

Roger Bosman

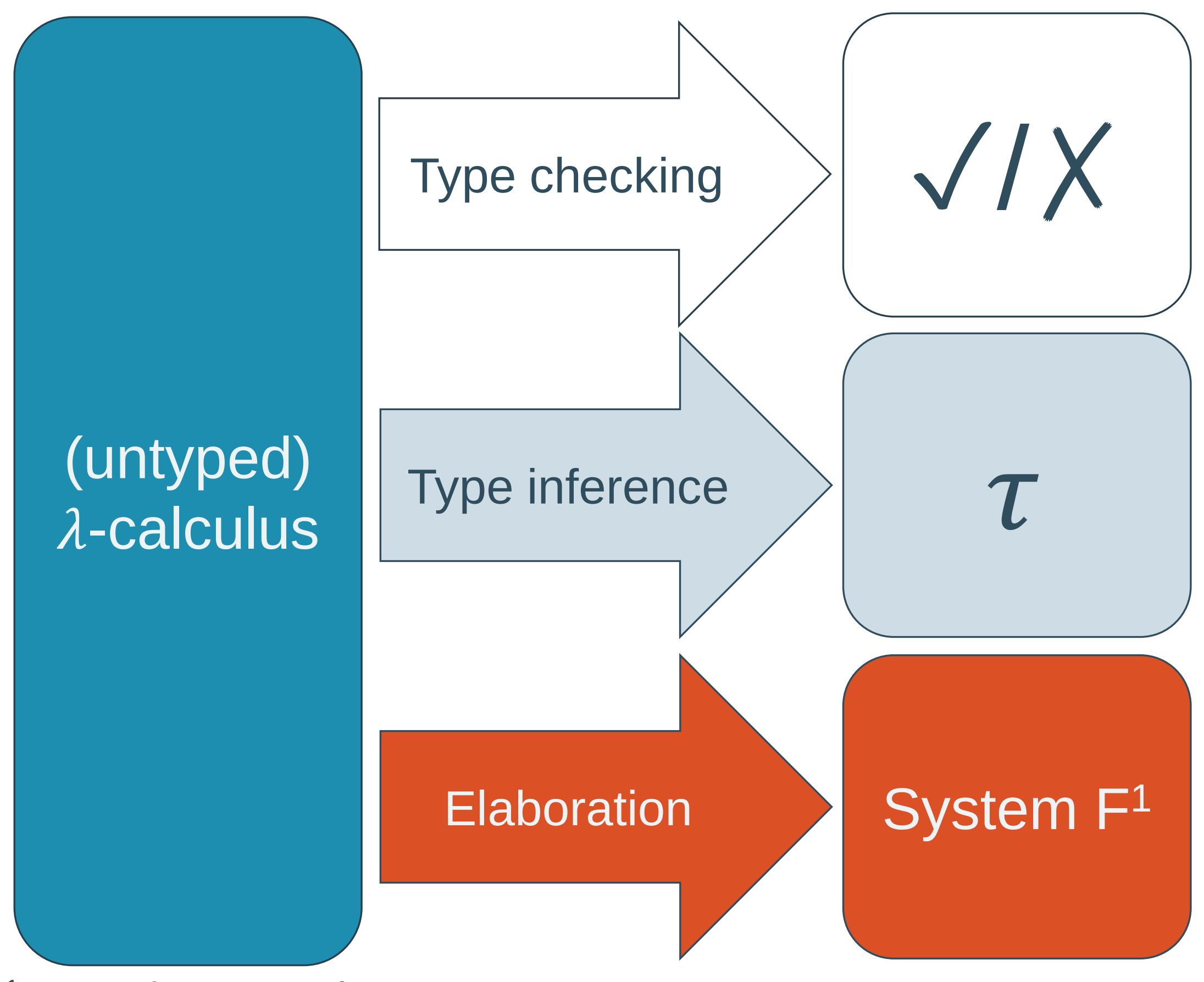
Supervisor: Prof. Tom Schrijvers

Type checking, Inference, & Elaboration

Type checking algorithms studied in detail

➥ Soundness, completeness, decidability, …

Compilers like GHC use *elaboration* algorithms



¹Or another output language

Why has nobody mechanized inference with elaboration before?

Many extensions implemented by elaboration

➥ Type classes, implicits, intersection types, …

Declarative vs. Algorithmic

The Hindley-Damas-Milner (HDM) system [1] cannot be implemented directly because its rules are *declarative* and not *algorithmic.*

$$\frac{\Gamma \vdash \tau_1 \quad \Gamma; x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x. e : \tau_1 \to \tau_2}$$

τ_1 is quantified *existentially*: no clear way of computing

We need an algorithm!

Classic implementation of HDM: algorithm $\mathcal{W}$ [1]

$\hat{\alpha}$ is *freshly generated* base on Γ

$\hat{\alpha}$ either is solved during type checking e or left in the type (to be solved later)

$$\frac{\hat{\alpha} = \text{fresh}(\Gamma) \quad \mathcal{W}(\Gamma; x : \hat{\alpha}, e) = (\sigma, \tau_2)}{\mathcal{W}(\Gamma, \lambda x. e) = (\sigma, \sigma(\hat{\alpha} \to \tau_2))}$$

Our algorithm

Now with explicit scope information!

Type checking	Elaboration

let x = (λy. unit) (λz. z) **in** …

↓ Assign existentials

$x :: \hat{\alpha}$ $y :: \hat{\beta}$ $z :: \hat{\gamma}$	**let** ($x :: \hat{\alpha}$) = ($\lambda(y :: \hat{\beta})$. unit) ($\lambda(z :: \hat{\gamma})$. z) **in** …

↓ Solve constraints

$\hat{\beta} = \hat{\gamma} \to \hat{\gamma}$ $\hat{\alpha} = Unit$	**let** ($x :: Unit$) = ($\lambda(y :: \hat{\gamma} \to \hat{\gamma})$. unit) ($\lambda(z :: \hat{\gamma})$. z) **in** …

↓ Generalize

Well-typed ✓	**let** ($x :: \forall a. Unit$) = Λa. ($\lambda(y :: a \to a)$. unit) ($\lambda(z :: a)$. z) **in** …

Our algorithm

Elaborated term t

$$\boxed{\overbrace{\Psi_{in} \vdash e}^{inputs} : \underbrace{[A]\tau \rightsquigarrow t \dashv \Psi_{out}}_{outputs}}$$

List A of existentials in scope of τ

Inferred type τ

Assign $\hat{\alpha}$ as type of x

Inference of e finds A_2 and refines $\hat{\alpha}$ to A_1

$$\frac{\begin{array}{c}\hat{\alpha} = \text{fresh}(\Psi_{in}) \\ (\Psi_{in}; (\hat{\alpha}); x : \hat{\alpha} \vdash e : [A_2]\tau_2 \rightsquigarrow t \dashv (\Psi_{out}; A_1; x : \tau_1)\end{array}}{\Psi_{in} \vdash \lambda x. e : [A_1 + A_2](\tau_1 \to \tau_2) \rightsquigarrow \lambda(x : \tau_1). t \dashv \Psi_{out}}$$

In scope of abstraction: $A_1 + A_2$

Infer monotype τ depending on A

$$\frac{\Psi_{in} \vdash e : [A]\tau \rightsquigarrow t \dashv \Psi_{out}}{\Psi_{in} \vdash e : \forall A. \tau \rightsquigarrow \Lambda A. \tau \dashv \Psi_{out}}$$

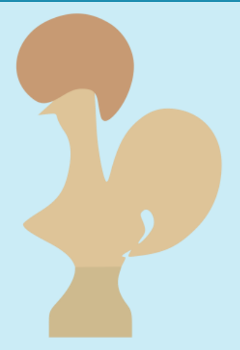
For all $\hat{\alpha} \in A$, add a $\forall$ and a Λ

State of affairs

Mechanization ongoing

- Coq proof assistant[2]
- Generalized rewriting[3]
- Locally nameless[4]
- Ott/Lngen[5,6]

Future work: extend type system with type classes



KU Leuven
Department of Computer Science
Celestijnenlaan 200A box 2402
3001 Leuven, Belgium

[1] Milner, R. (1978). A theory of type polymorphism in programming. *Journal of computer and system sciences*, *17*(3), 348-375.
[2] Bertot, Y., & Castéran, P. (2013). *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media.
[3] Sozeau, M. (n.d.). *Generalized rewriting*. Coq Reference Manual. Retrieved August 18, 2021, from https://coq.inria.fr/refman/addendum/generalized-rewriting.html
[4] Charguéraud, A. (2012). The locally nameless representation. *Journal of automated reasoning*, *49*(3), 363-408.
[5] Sewell, P., Nardelli, F. Z., Owens, S., Peskine, G., Ridge, T., & Sarkar, S. (2010). Ott: Effective tool support for the working semanticist. *Journal of functional programming*, *20*(1), 71-122.
[6] Aydemir, B., & Weirich, S. (2010). LNgen: Tool support for locally nameless representations.