

Rich types, Tractable typing – Type Inference –

Yann Régis-Gianas
yrg@irif.fr

2018-12-07

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Plan

What is type inference?

ML

Constraint based approach

Conclusion

Reading list

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

What is type inference?

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Write down your current answers to the following questions:

1. What is the type of a type inference engine?
2. What are the properties you expect from it?

What is type inference?

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

Informal definition

Type inference is a process that computes a type τ for a term t under some typing environment Γ if such a type exists. In other words, we are reading the judgment:

$$\Gamma \vdash t : \tau$$

where t and Γ are the inputs and τ is the output.

Hence, type inference determines if a term is typable.

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Too vague!

Consider:

```
1 let f = fun x -> x + 1
```

Intuitively, a type inference engine must be able to process the term $x + 1$ under an environment where the type of x is unknown.

Therefore, the typing environments used by a type inference engine differ slightly from the typing environments used by a type checker in the sense that the types bound to identifiers may contain pieces of unknown typing information. As soon as the syntax for types offers a notion of type variables, unknown typing information can be represented by free type variables.

This means that a free type variable occurring in a judgment may have two distinct roles: either it denotes a parameter of the typing derivation¹, or it denotes an unknown type to be instantiated by the inference algorithm. This distinction must be formally made.

¹It is morally universally quantified at the meta-level.

Let us be more formal!

Let us write \mathcal{V} to denote an infinite set of type variable identifiers from which we can generate fresh names. These type variables will denote the pieces of unknown typing information of the inference problem. The goal of type inference is to determine if these type variables can be assigned types to ensure the typability of the input term under the input environment.

Definition

A **type inference engine** is a partial function \mathcal{I} . This function expects \mathcal{V} , as well as a typing environment Γ and a term t . When defined, this function returns ϕ , an idempotent substitution whose domain is a finite subset of \mathcal{V} and an inferred type τ .

Definition

A type inference engine \mathcal{I} is **sound** if whenever $\mathcal{I}(\mathcal{V}, \Gamma, t) = (\mathcal{V}', \phi, \tau)$ then $\phi(\Gamma) \vdash t : \tau$.

The substitution ϕ witnesses the typability of t under Γ .

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Parenthesis: α or $? \alpha$?

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

We could have followed another design choice by introducing a notion of **meta-variables**² to represent unknown types.

Yet, we will see that the strength of Hindley-Milner type system is to play with type variables to promote them from unknown types to parameters through the mechanism of **generalization**.

²As this is done in the Coq system for instance.

Parenthesis: Alternative definition

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

We could have devised an alternative definition in which only the term t is the input and Γ also has to be inferred in addition to the type. Such a definition has interesting benefits but is less standard and less studied.

Soundness is not enough

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

As soon as a term is typable, the type inference algorithm must be able to find a typing for this term.

Definition

A type inference engine is **complete** and **principal** if whenever there exists a substitution ϕ such that $\phi(\Gamma) \vdash t : \phi(\tau)$ holds, then there exists ϕ', ϕ'' and \mathcal{V} disjoint from $\text{FTV}(\Gamma, \tau)$ such that:

$$\begin{aligned}\mathcal{I}(\mathcal{V}, \Gamma, t) &= (\mathcal{V}', \phi'', \tau') \\ \phi''(\tau') &= \tau \\ \phi(\alpha) &= (\phi'' \circ \phi')(\alpha) \quad \forall \alpha \notin \mathcal{V}\end{aligned}$$

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

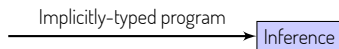
Conclusion

Reading list

Type inference in a compiler

Type inference algorithms can be tricky to implement, in particular when we want them to be efficient. Fortunately, we have a safety net: the typechecker for the corresponding explicitly-typed language!

In practice, the so-called De Bruijn architecture is encouraged:



Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

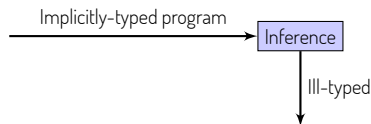
Conclusion

Reading list

Type inference in a compiler

Type inference algorithms can be tricky to implement, in particular when we want them to be efficient. Fortunately, we have a safety net: the typechecker for the corresponding explicitly-typed language!

In practice, the so-called De Bruijn architecture is encouraged:



Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

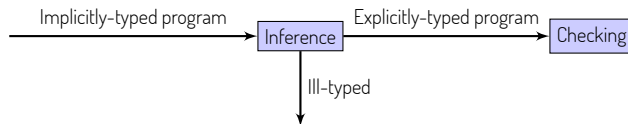
Conclusion

Reading list

Type inference in a compiler

Type inference algorithms can be tricky to implement, in particular when we want them to be efficient. Fortunately, we have a safety net: the typechecker for the corresponding explicitly-typed language!

In practice, the so-called De Bruijn architecture is encouraged:



Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

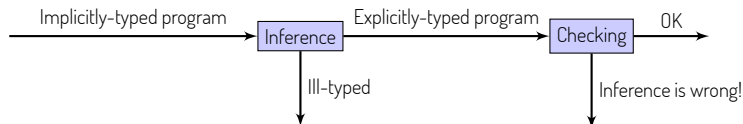
Conclusion

Reading list

Type inference in a compiler

Type inference algorithms can be tricky to implement, in particular when we want them to be efficient. Fortunately, we have a safety net: the typechecker for the corresponding explicitly-typed language!

In practice, the so-called De Bruijn architecture is encouraged:



Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

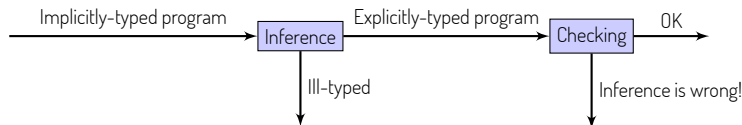
Conclusion

Reading list

Type inference in a compiler

Type inference algorithms can be tricky to implement, in particular when we want them to be efficient. Fortunately, we have a safety net: the typechecker for the corresponding explicitly-typed language!

In practice, the so-called De Bruijn architecture is encouraged:



Elaboration

This extended type inference must **elaborate** an explicitly-typed program as a proof witness for the soundness of its answer. This proof is checked by an hopefully simpler, smaller and trustworthy program, the typechecker. Notice that the principality of the inferred program is not checked here.

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

From type inference to elaboration

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

Definition

A programming language is **implicitly typed** if its syntax allows the introduction of an identifier without declaring its type. The language is **explicitly typed** otherwise.

Let \mathcal{L}_x be an explicitly typed language, and \mathcal{L}_i be an implicitly typed language sharing the same type-erasure semantics and the same type algebra.

Definition

A **sound elaboration engine** is a partial function \mathcal{E} such that:

If $\mathcal{E}(\mathcal{V}, \Gamma, t) = (\mathcal{V}', \phi, \tau, t^*)$ where $t \in \mathcal{L}_i$
then $t^* \in \mathcal{L}_x$ and $\phi(\Gamma) \vdash t^* : \phi(\tau)$

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

This course

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

- ▶ ML, syntax, semantics and type system(s)
- ▶ Algorithm **W** in OCaml
- ▶ Constraint-based approach

Plan

What is type inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based approach

Conclusion

Reading list

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Plan

What is type inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based approach

Conclusion

Reading list

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Syntax for terms (**Syntax**. **ITerm**. **t**³)

$$\begin{array}{lcl} t & ::= & x \\ & | & \lambda x. t \\ & | & t \ t \\ & | & \mathbf{let} \ x = t \ \mathbf{in} \ t \end{array}$$

Notice the absence of type annotations on bindings.

Operational semantics

We assume a call-by-value weak reduction semantics.

What is type inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based approach

Conclusion

Reading list

³See the companion code in the repository.

A stratified type algebra

Syntax for types (**Syntax.Type.t**)

$$\begin{array}{lcl} \tau & ::= & \alpha \\ & | & \varepsilon(\overline{\tau}) \\ \varepsilon & ::= & \rightarrow_2 | \mathbf{int}_0 | \dots \end{array}$$

A type is a first-order term.

Syntax for type schemes (**Syntax.TypeScheme.t**)

$$\sigma ::= \forall \overline{\alpha}. \tau$$

- ▶ The \forall binder quantifies over (mono)types.
- ▶ Quantification is **prenex** : it cannot appear everywhere as in F.
- ▶ This is **predicative** rank-1 parametric polymorphism.

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Definition

The type τ' is an instance of the type scheme $\forall \bar{\alpha}. \tau$, written $\forall \bar{\alpha}. \tau \preceq \tau'$, if there exists $\bar{\tau}$ such that $[\bar{\alpha} \mapsto \bar{\tau}] \tau = \tau'$.

Type system

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

The type system of ML is defined by two typing judgments:

$$\Gamma \vdash t : \tau \quad \text{and} \quad \Gamma \vdash t : \forall \bar{\alpha}. \tau$$

where $\Gamma ::= \bullet \mid \Gamma(x : \sigma)$.

While the first judgment is a standard typing judgment, the second can be seen as a family of standard typing judgments, parameterized by the types $\bar{\alpha}$.

Going from the second judgment to the first is an instantiation. The other way around is a generalization:

$$\frac{\text{Inst} \quad \Gamma \vdash t : \sigma \quad \sigma \preceq \tau}{\Gamma \vdash t : \tau}$$

$$\frac{\text{Gen} \quad \Gamma \vdash t : \tau \quad \bar{\alpha} \# \text{FTV}(\Gamma)}{\Gamma \vdash t : \forall \bar{\alpha}. \tau}$$

What is type inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based approach

Conclusion

Reading list

Exercise

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Exercise

Do you remember why the hypothesis

$$\bar{\alpha} \# \text{FTV}(\Gamma)$$

is important in the Rule **(Gen)**?

Do you know ML?

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Is

```
1 let f x = x in (f 0, f 'a')
```

equivalent to

```
1 (fun f -> (f 0, f 'a')) (fun x -> x)
```

?

Typing rules

In addition to (Gen), the rule (Var) can be used to introduce parameterized typing judgments:

$$\text{Var} \quad \frac{}{\Gamma \vdash x : \Gamma(x)}$$

Binding type schemes to variables is the role of (Let):

$$\text{Let} \quad \frac{\Gamma \vdash t : \sigma \quad \Gamma, (x : \sigma) \vdash u : \tau}{\Gamma \vdash \mathbf{let} \ x = t \ \mathbf{in} \ u : \tau}$$

The rules for applications and abstractions are the same as for STLC:

$$\text{App} \quad \frac{\Gamma \vdash t : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash u : \tau_1}{\Gamma \vdash t u : \tau_2}$$

$$\text{Abs} \quad \frac{\Gamma, (x : \forall \emptyset. \tau_1) \vdash t : \tau_2}{\Gamma \vdash \lambda x. t : \tau_1 \rightarrow \tau_2}$$

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Properties of this type system

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

(We defer the discussion about the soundness of this type system.)

Question 1

Given an environment Γ and a typable term t , is there a unique type τ such that $\Gamma \vdash t : \tau$?

Question 2

Given a derivable judgment $\Gamma \vdash t : \tau$, is there a unique typing derivation that has this conclusion?

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Properties of this type system

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

(We defer the discussion about the soundness of this type system.)

Question 1

Given an environment Γ and a typable term t , is there a unique type τ such that $\Gamma \vdash t : \tau$? **No!**

Question 2

Given a derivable judgment $\Gamma \vdash t : \tau$, is there a unique typing derivation that has this conclusion? **No! The rules are not syntax-directed.**

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Properties of this type system

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

(We defer the discussion about the soundness of this type system.)

Question 1

Given an environment Γ and a typable term t , is there a unique type τ such that $\Gamma \vdash t : \tau$? **No!**

Question 2

Given a derivable judgment $\Gamma \vdash t : \tau$, is there a unique typing derivation that has this conclusion? **No! The rules are not syntax-directed.**

Question 1 will be tackled by the existence of principal type schemes.
Let us deal with Question 2 for now.

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Syntax directed typing rules

We define another type system enjoying uniqueness of typing derivation, similar to the previous one except that the rules (Gen), (Inst), (Var) et (Let) are replaced by ⁴:

Let-Gen

$$\frac{\begin{array}{l} \Gamma \vdash t : \tau_1 \\ \bar{\alpha} = \text{FTV}(\tau_1) \setminus \text{FTV}(\Gamma) \\ \Gamma, (x : \forall \bar{\alpha}. \tau_1) \vdash u : \tau_2 \end{array}}{\Gamma \vdash \mathbf{let} \ x = t \ \mathbf{in} \ u : \tau_2}$$

Var-Inst

$$\frac{\Gamma(x) \preceq \tau}{\Gamma \vdash x : \tau}$$

Since the system is now syntax-directed, does that mean that we have a type inference algorithm? or at least a type checking algorithm?

⁴We defer the proof of equivalence of the two type systems.

Syntax directed typing rules

We define another type system enjoying uniqueness of typing derivation, similar to the previous one except that the rules (Gen), (Inst), (Var) et (Let) are replaced by ⁴:

Let-Gen

$$\frac{\begin{array}{l} \Gamma \vdash t : \tau_1 \\ \bar{\alpha} = \text{FTV}(\tau_1) \setminus \text{FTV}(\Gamma) \\ \Gamma, (x : \forall \bar{\alpha}. \tau_1) \vdash u : \tau_2 \end{array}}{\Gamma \vdash \mathbf{let} \ x = t \ \mathbf{in} \ u : \tau_2}$$

Var-Inst

$$\frac{\Gamma(x) \preceq \tau}{\Gamma \vdash x : \tau}$$

Since the system is now syntax-directed, does that mean that we have a type inference algorithm? or at least a type checking algorithm? Unfortunately, no. There remain too many choices:

- What are the types of λ -bound identifiers?
- How much generalization is needed?

Typability in ML is a non local property.

⁴We defer the proof of equivalence of the two type systems.

ML typing constraints

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

The non locality of the typability problem is better handled by unification constraints. Historically, these constraints are implicitly generated and solved on-the-fly by an algorithm called \mathcal{W} .

More recent approaches split type inference into two phases: a constraint generation and the solving of these constraints.

We will now implement and prove \mathcal{W} , turn it into an elaboration algorithm and present a constraint-based approach to ML type inference.

What is type
inference?

ML

An implicitly typed language

Algorithm \mathcal{W}

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Plan

What is type inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based approach

Conclusion

Reading list

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

$$\begin{aligned}\mathcal{W}(\mathcal{V}, \Gamma, x) &= (\mathcal{V} \setminus \bar{\beta}, id, [\bar{\alpha} \mapsto \bar{\beta}]\tau) \\ \text{where } \Gamma(x) &= \forall \bar{\alpha}. \tau \text{ and } \bar{\beta} \in \mathcal{V}\end{aligned}$$

$$\begin{aligned}\mathcal{W}(\mathcal{V}, \Gamma, \lambda x. t) &= (\mathcal{V}', \phi, \phi(\alpha) \rightarrow \tau) \\ \text{where } \alpha &\in \mathcal{V} \\ \text{and } \mathcal{W}(\mathcal{V} \setminus \alpha, \Gamma; (x : \alpha), t) &= (\mathcal{V}', \phi, \tau)\end{aligned}$$

$$\begin{aligned}\mathcal{W}(\mathcal{V}, \Gamma, t \ u) &= (\mathcal{V}'', \phi \circ \phi_t \circ \phi_u, \phi(\alpha)) \\ \text{where } \alpha &\in \mathcal{V} \\ \text{and } \mathcal{W}(\mathcal{V} \setminus \alpha, \Gamma, u) &= (\mathcal{V}', \phi_u, \tau_u) \\ \text{and } \mathcal{W}(\mathcal{V}', \phi_u(\Gamma), t) &= (\mathcal{V}'', \phi_t, \tau_t) \\ \text{and } \phi &= \text{MGU}(\tau_t \stackrel{?}{=} \tau_u \rightarrow \alpha)\end{aligned}$$

$$\begin{aligned}\mathcal{W}(\mathcal{V}, \Gamma, \text{let } x = t \text{ in } u) &= (\mathcal{V}'', \phi_2 \circ \phi_1, \tau_2) \\ \text{where } \mathcal{W}(\mathcal{V}, \Gamma, t) &= (\mathcal{V}', \phi_1, \tau_1) \\ \text{and } \mathcal{W}(\mathcal{V}', \Gamma, (x : \sigma), u) &= (\mathcal{V}'', \phi_2, \tau_2) \\ \text{and } \sigma &= \text{GEN}(\Gamma, \phi_1(\tau_1))\end{aligned}$$

Auxiliary functions

Let us write U for a first-order unification problem made of a conjunction of qualities between types.

$$\text{MGU}(x \stackrel{?}{=} x \wedge U) = \text{MGU}(U)$$

$$\text{MGU}(\tau \stackrel{?}{=} x \wedge U) = \text{MGU}(x \stackrel{?}{=} \tau \wedge U)$$

when τ is not a variable

$$\text{MGU}(x \stackrel{?}{=} \tau \wedge U) = \text{MGU}(U[x \mapsto \tau]) \circ [x \mapsto \tau]$$

when $x \notin \text{FTV}(\tau)$

$$\text{MGU}(\varepsilon_1(\bar{\tau}_1) \stackrel{?}{=} \varepsilon_2(\bar{\tau}_2) \wedge U) = \text{MGU}(\bar{\tau}_1 \stackrel{?}{=} \bar{\tau}_2 \wedge U)$$

when $\varepsilon_1 = \varepsilon_2$

$$\text{MGU}(\top) = \text{id}$$

MGU is undefined for the other cases.

Besides, the generalization operation over types is defined as:

$$\text{GEN}(\Gamma, \tau) = \forall(\text{FTV}(\tau) \setminus \text{FTV}(\Gamma)).\tau$$

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Theorem (\mathcal{W} is sound)

If $\mathcal{W}(\mathcal{V}, \Gamma, t) = (\mathcal{V}', \phi, \tau)$ then $\phi(\Gamma) \vdash t : \tau$

Proof.

By induction over terms.



Completeness and principality

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

Theorem (\mathcal{W} is complete and computes principal types)

If there exists ϕ and τ such that $\phi(\Gamma) \vdash t : \tau$,
then there exists $\mathcal{V}', \phi', \tau'$ and ρ such that

$$\begin{aligned}\mathcal{W}(\mathcal{V}, \Gamma, t) &= (\mathcal{V}', \phi', \tau') \\ \tau &= \rho(\tau') \\ \phi(\alpha) &= \phi'(\rho(\alpha)) \quad \forall \alpha \notin \mathcal{V}\end{aligned}$$

Proof.

By induction over terms.



What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Completeness and principality

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

Theorem (\mathcal{W} is complete and computes principal types)

If there exists ϕ and τ such that $\phi(\Gamma) \vdash t : \tau$,
then there exists $\mathcal{V}', \phi', \tau'$ and ρ such that

$$\begin{aligned}\mathcal{W}(\mathcal{V}, \Gamma, t) &= (\mathcal{V}', \phi', \tau') \\ \tau &= \rho(\tau') \\ \phi(\alpha) &= \phi'(\rho(\alpha)) \quad \forall \alpha \notin \mathcal{V}\end{aligned}$$

Proof.

By induction over terms.



(In these proofs, the substitution manipulations are “tricky”!)

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Back to Question 1

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Question 1

Given an environment Γ and a typable term t , is there a unique type τ such that $\Gamma \vdash t : \tau$? **No!**

Back to Question 1

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Question 1

Given an environment Γ and a typable term t , is there a unique type τ such that $\Gamma \vdash t : \tau$? **No!** But one can find a type that rules them all!
 \mathcal{W} is a constructive proof of that fact!

Complexity

- ▶ ML typability is NP-hard and DEXPTIME-complete.
- ▶ Here is a typical example that requires an exponential time to type:

```
1  let f0 = fun x -> x in
2  let f1 = (f0, f0) in
3  let f2 = (f1, f1) in
4  ...
5  fN
```

- ▶ **But** under reasonable assumptions⁵, the complexity is quasi-linear.

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

⁵In the wild, the depth of types are bounded!

Plan

What is type inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based approach

Conclusion

Reading list

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Do it yourself!

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Programming exercise

1. Define the syntax of an explicitly typed version of ML.
2. Implement **Elaboration.algorithm_w** targeting the language you just defined.
3. Are you able to locate the binding site of every type variables that occur in the elaborated terms?
(Let us name this question “Question 0”.)

eML, an explicitly typed ML (Syntax)

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

We reuse the syntax of System F for abstractions with respect to types and for type applications:

$$\begin{array}{lcl} M & ::= & x \\ & | & \lambda(x : \tau).M \\ & | & M M \\ & | & \Lambda\alpha.M \\ & | & M\tau \\ & | & \mathbf{let} \ x : \sigma = M \ \mathbf{in} \ M \end{array}$$

Notice that, contrary to System F, λ -bound identifiers are assigned a monomorphic type. As in ML, only **let**-bound identifiers can be polymorphic.

What is type inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based approach

Conclusion

Reading list

eML, an explicitly typed ML (Typing rules)

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

$$\frac{}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma, (x : \tau_1) \vdash M : \tau_2}{\Gamma \vdash \lambda(x : \tau_1).M : \tau_1 \rightarrow \tau_2}$$

$$\frac{\Gamma \vdash M_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash M_2 : \tau_1}{\Gamma \vdash M_1 M_2 : \tau_2}$$

$$\frac{\Gamma \vdash M_1 : \sigma_1 \quad \Gamma, (x : \sigma_1) \vdash M_2 : \sigma_2}{\Gamma \vdash \mathbf{let} \ x : \sigma_1 = M_1 \ \mathbf{in} \ M_2 : \sigma_2} \qquad \frac{\Gamma, \alpha \vdash M : \sigma}{\Gamma \vdash \Lambda \alpha. M : \forall \alpha. \sigma}$$

$$\frac{\Gamma \vdash M : \forall \alpha. \sigma}{\Gamma \vdash M \tau : \sigma[\alpha \mapsto \tau]}$$

Wait a second!

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

But this is not the language we define in the previous exercise,
right?

xML, another explicitly typed ML

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

The language we defined as a target of \mathcal{W} elaboration is:

$$\begin{array}{lcl} N & ::= & \Lambda \bar{\alpha}.Q \\ Q & ::= & x\bar{\tau} \\ & | & \lambda(x : \tau).Q \\ & | & Q\ Q \\ & | & \mathbf{let}\ x : \sigma = N\ \mathbf{in}\ Q \end{array}$$

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

xML, another explicitly typed ML

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

The language we defined as a target of \mathcal{W} elaboration is:

$$\begin{array}{lcl} N & ::= & \Lambda \bar{\alpha}. Q \\ Q & ::= & x \bar{\tau} \\ & | & \lambda(x : \tau). Q \\ & | & Q Q \\ & | & \text{let } x : \sigma = N \text{ in } Q \end{array}$$

How do we relate eML and xML?

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Normalization of eML typing derivations

Let us write

$$\Gamma \vdash M : \sigma \Rightarrow N$$

and

$$\Gamma \vdash M : \tau \Rightarrow Q$$

for two judgments that denote the normalization of the typing derivation of M as a typing derivation in xML.

More formally, we want the following properties to hold:

Lemma (Normalization preserves well-typedness)

If $\Gamma \vdash M : \sigma$ in eML and $\Gamma \vdash M : \sigma \Rightarrow N$ then $\Gamma \vdash N : \sigma$ in xML.
(Idem for the monomorphic case.)

Lemma (Well-formed typing derivations normalize)

If $\Gamma \vdash M : \sigma$ in eML then $\Gamma \vdash M : \sigma \Rightarrow N$.
(Idem for the monomorphic case.)

Do it yourself!

Formalization exercise

Define the rules for the previous two judgments.

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Normalization rules

Let us start with

$$\Gamma \vdash M : \sigma \Rightarrow N$$

The syntax of N forces us to η -expand x :

$$\begin{array}{c} \text{Norm-Var} \\ \Gamma(x) = \forall \bar{\alpha}. \tau \\ \hline \Gamma \vdash x : \forall \bar{\alpha}. \tau \Rightarrow \Lambda \bar{\alpha}. (x \bar{\alpha}) \end{array}$$

The case for type abstraction is obvious:

$$\begin{array}{c} \text{Norm-TAbs} \\ \Gamma, \alpha \vdash M : \sigma \Rightarrow N \\ \hline \Gamma \vdash \Lambda \alpha. M : \forall \alpha. \sigma \Rightarrow \Lambda \alpha. N \end{array}$$

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Normalization rules (continued)

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

Type applications are reduced during the normalization so that the resulting term N is normalized with respect to strong ι -reduction:

Norm-TApp

$$\frac{\Gamma \vdash M : \forall \alpha. \sigma \Rightarrow \Lambda \alpha. N}{\Gamma \vdash M \tau : \sigma[\alpha \mapsto \tau] \Rightarrow N[\alpha \mapsto \tau]}$$

What is type inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based approach

Conclusion

Reading list

To comply with the syntax, the type abstractions coming from the right-hand-side of **let**-bindings must be extruded:

Norm-Let

$$\frac{\Gamma \vdash M_1 : \sigma \Rightarrow N_1 \quad \bar{\alpha} \# \sigma, N_1 \quad \Gamma, (x : \sigma) \vdash M_2 : \forall \bar{\alpha}. \tau \Rightarrow \Lambda \bar{\alpha}. Q}{\Gamma \vdash \mathbf{let} \ x : \sigma = M_1 \ \mathbf{in} \ M_2 : \forall \bar{\alpha}. \tau \Rightarrow \Lambda \bar{\alpha}. \mathbf{let} \ x : \sigma = N_1 \ \mathbf{in} \ Q}$$

Normalization rules (continued)

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

Finally, the two rules for applications and λ -abstraction are straightforward since the two languages coincide on these constructions:

Norm-App

$$\frac{\Gamma \vdash M_1 : \tau_1 \rightarrow \tau_2 \Rightarrow Q_1 \quad \Gamma \vdash M_2 : \tau_1 \Rightarrow Q_2}{\Gamma \vdash M_1 M_2 : \tau_2 \Rightarrow Q_1 Q_2}$$

Norm-Abs

$$\frac{\Gamma(x : \tau) \vdash M : \tau_2 \Rightarrow Q}{\Gamma \vdash \lambda(x : \tau).M : \tau_1 \rightarrow \tau_2 \Rightarrow \lambda(x : \tau).Q}$$

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Back to Lemmas

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

Lemma (Well-formed typing derivations normalize)

If $\Gamma \vdash M : \sigma$ in eML then $\Gamma \vdash M : \sigma \Rightarrow N$.

(Idem for the monomorphic case.)

Proof.

Easy induction over typing derivations of eML. □

Lemma (Normalization preserves well-typedness)

If $\Gamma \vdash M : \sigma$ in eML and $\Gamma \vdash M : \sigma \Rightarrow N$ then $\Gamma \vdash N : \sigma$ in xML.

(Idem for the monomorphic case.)

Proof.

By induction over typing derivations of eML. □

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Two birds with one stone!

Lemma

Type erasure preservation If $\Gamma \vdash M : \sigma \Rightarrow N$ then the type erasures of M and N are equal.

Proof.

Immediate by induction. □

Equivalence of two pairs of type systems for ML

- ▶ For any derivation of eML, there is an equivalent derivations of xML. (The other direction is obvious.) : We have a typechecker for explicitly-typed ML!
- ▶ If we remove the type annotations from the syntax, the proof can be transported to the (implicitly typed) ML type systems we have introduced earlier!
(Question 2 is now solved.)

And what about “Question 0”?

Remember:

Are you able to locate the binding site of every type variables that occur in the elaborated terms?

The unification type variables that have not been promoted to generalized type variables are still floating in the air. This is not really a problem: these variables can be seen as existentially quantified at the toplevel.

Yet, a cleaner treatment of these type variables consists in the introduction of an existential quantification over these (flexible) type variables at the level of terms:

$$t ::= \dots \mid \exists \alpha. t$$

with a companion typing rule of the form:

$$\frac{\Gamma \vdash t : \tau[\alpha \mapsto \tau']}{\Gamma \vdash \exists \alpha. t : \tau}$$

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

An open question : the type soundness of ML

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Many proofs are possible

- ▶ From scratch, by mimicking the proof for System F.
- ▶ By deducing the type soundness of eML from System F's.
(See Didier Remy's course notes, section 4.6.3.)

Plan

What is type inference?

ML

Constraint based approach

Conclusion

Reading list

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Syntax for typing constraints

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

We will now reformulate the type inference problem in ML using **typing constraints** written using the following syntax:

$$\begin{array}{lcl} C & ::= & \text{true} \\ & | & \text{false} \\ & | & C \wedge C \\ & | & \tau = \tau \\ & | & \exists \alpha. C \\ & | & \text{let } x = \lambda \alpha. C \text{ in } C \\ & | & x\tau \end{array}$$

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Meaning of constraints

Consider the standard meaning of the unification constraints U where:

$$U ::= \mathbf{true} \mid \mathbf{false} \mid U \wedge U \mid \tau = \tau \mid \exists \alpha. U$$

then, we can define the meaning of a constraint C of the form:

$$\mathbf{let} \ x = \lambda \alpha. C_1 \ \mathbf{in} \ C_2$$

by defining it as the meaning of the **let**-expanded constraint:

$$(\exists \alpha. C_1) \wedge C_2[x \mapsto \lambda \alpha. C_1]$$

If we call ζ this reduction, we have:

$$\frac{\psi \models U \quad U \text{ is the } (\zeta, \beta)\text{-normal form of } C}{\psi \models C}$$

The key idea

So, what is the point of introducing **let** in constraints?

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

The key idea

So, what is the point of introducing **let** in constraints?

Lemma (Principal Type Scheme)

Every satisfiable constraint abstraction of the form $\lambda\alpha.C$ can be rewritten into an equivalent constraint of the form $\lambda\alpha.\exists\bar{\beta}.\alpha = \tau$.

Lemma (Satisfiable constraint abstractions are type schemes)

The type τ' satisfies the predicate $\lambda\alpha.\exists\bar{\beta}.\alpha = \tau$ iff $\forall\bar{\beta}.\tau \preceq \tau'$.

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

The key idea

So, what is the point of introducing **let** in constraints?

Lemma (Principal Type Scheme)

Every satisfiable constraint abstraction of the form $\lambda\alpha.C$ can be rewritten into an equivalent constraint of the form $\lambda\alpha.\exists\bar{\beta}.\alpha = \tau$.

Lemma (Satisfiable constraint abstractions are type schemes)

The type τ' satisfies the predicate $\lambda\alpha.\exists\bar{\beta}.\alpha = \tau$ iff $\forall\bar{\beta}.\tau \preceq \tau'$.

Typing constraints capture the essence of ML type inference in a declarative way.

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

The key idea

So, what is the point of introducing **let** in constraints?

Lemma (Principal Type Scheme)

Every satisfiable constraint abstraction of the form $\lambda\alpha.C$ can be rewritten into an equivalent constraint of the form $\lambda\alpha.\exists\bar{\beta}.\alpha = \tau$.

Lemma (Satisfiable constraint abstractions are type schemes)

The type τ' satisfies the predicate $\lambda\alpha.\exists\bar{\beta}.\alpha = \tau$ iff $\forall\bar{\beta}.\tau \preceq \tau'$.

Typing constraints capture the essence of ML type inference in a declarative way.

Exercise

Take some time to define the constraint $\llbracket t : \tau \rrbracket$ read as “ t has type τ ”.

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Constraint generation

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

The constraint $\llbracket t : \tau \rrbracket$ is read as “ t has type τ ” and is defined as:

$$\begin{aligned}\llbracket x : \tau \rrbracket &= x \tau \\ \llbracket t u : \tau \rrbracket &= \exists \alpha. \llbracket t : \alpha \rightarrow \tau \rrbracket \wedge \llbracket u : \alpha \rrbracket \\ \llbracket \lambda x. t : \tau \rrbracket &= \exists \beta \gamma. (\tau = \gamma \rightarrow \beta \wedge \\ &\quad \text{let } x = (\lambda \alpha. \alpha = \gamma) \text{ in } \llbracket t : \beta \rrbracket) \\ \llbracket \text{let } x = t \text{ in } u : \tau \rrbracket &= \text{let } x = \lambda \alpha. \llbracket t : \alpha \rrbracket \text{ in } \llbracket u : \tau \rrbracket\end{aligned}$$

Let us define “**def** Γ **in** C ” by induction on Γ :

$$\begin{aligned}\mathbf{def} \bullet \mathbf{in} C &= C \\ \mathbf{def} \Gamma, (x : \forall \bar{\beta}. \tau) \mathbf{in} C &= \mathbf{def} \Gamma \mathbf{in} \mathbf{let} x = \lambda \alpha. \exists \bar{\beta}. \alpha = \tau \mathbf{in} C\end{aligned}$$

Lemma (Soundness)

If the constraint **def** Γ **in** $\llbracket t : \tau \rrbracket$ is satisfiable then $\Gamma \vdash t : \tau$ holds.

Completeness

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

To reason about completeness, we need to extract the typing constraint of a specific typing derivation. This can be done using a (once again!) reformulation of ML type system as a **constrained type system** whose judgment:

$$C, \Gamma \vdash t : \tau$$

is read

Under constraint C and environment Γ , the term t has type τ .

(A similar judgment for type schemes is introduced as well.)

What is type inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based approach

Conclusion

Reading list

Completeness and principality

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Lemma (Completeness and principality)

If $C, \Gamma \vdash t : \tau$ holds then C entails **def** Γ in $\llbracket t : \tau \rrbracket$.

Advantages of the constraint-based approach over \mathcal{W}

By separating constraint generation and constraint solving, the constraint-based approach is more modular than the monolithic algorithm \mathcal{W} . This implies that:

- ▶ The constraint language remains small even if the programming language is extended with new constructions.
- ▶ Being declarative, the constraint language makes it easier to write **correct** generation rules.
- ▶ The optimizations of the type inference engine are clearly localized: they must occur in the constraint solver.

Besides, reasoning about the solving process or about the constraint generation using equivalence of constraints is “higher-level” than reasoning with substitutions.

Last but not least, we will later see that typing constraints lead us to a natural generalization of ML called HM(X) which makes it simpler to add some form of subtyping in ML.

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm \mathcal{W}

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Formalization exercises

- ▶ Extend ML with algebraic data types (that is with data constructor applications and pattern-matching) and compare the amount of work needed to update \mathcal{W} and the amount of work needed to update the constraint generation.

What about elaboration?

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

The constraint-based approach seems seducing as it abstracts the syntax of the source language to only focus on the type inference problem. But what if we want to use such an approach to implement an elaboration engine?

It is not totally true that the input source term is forgotten by the constraint-based inference engine since the syntax of the constraint follows the structure on the input source term. Would it be possible to preserve this structure through the solving process?

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

α -constraints

Constraint can actually be generalized into α -constraints, constraints whose satisfiability is witnessed by a value of type α . In the case of an elaboration, this value will actually be the elaborated term.

The construction of this elaborated term is possible if the inferred types and the inferred type schemes can be “decoded” in the source syntax and if the structure of the elaborated term can be reconstructed bottom-up along the solving process. For the first condition, it suffices to ask the programmer to provide an appropriate decoding function. For the second condition, the syntax for constraints is extended with a new construct:

$$C ::= \dots \mid \text{fmap } f \ C$$

which allows the programmer to incrementally construct the elaborated in a continuation-passing style.

(See François Pottier’s paper in the reading list for more details.)

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Do it yourself!

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Programming exercise

1. Read François Pottier's paper.
2. Complete **ConstraintBasedElaboration** to implement ML type inference with **inferno**.

Plan

What is type inference?

ML

Constraint based approach

Conclusion

Reading list

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Conclusion

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What did you learn today?

- ▶ A type inference engine must be sound, complete and outputs principal types.
- ▶ Prefer elaboration engines over type inference engines.
- ▶ Type inference in ML is neat!

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

Is type inference a closed problem?

- ▶ Type inference in System F is undecidable.
- ▶ Type inference in presence of subtyping is subtle.
- ▶ Type inference in presence of GADTs is ugly.

Plan

What is type inference?

ML

Constraint based approach

Conclusion

Reading list

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

- ▶ The original paper:
Principal type-schemes for functional programs
Luis Damas and Robin Milner
POPL '82
- ▶ For a proof of \mathcal{W} :
 - ▶ The course notes of Xavier Leroy
 - ▶ A simple algorithm and proof for type inference
Mitchell Wand
FI'87
- ▶ About elaboration in ML:
Course notes of Didier Remy (Section 4.6)

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

About constraint-based approaches

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

- ▶ About constraint abstractions:
Constraint abstractions.
Jörgen Gustavsson and Josef Svenningsson.
SPD0'01
- ▶ About α -constraints:
Hindley-Milner Elaboration in Applicative Style
Francois Pottier
ICFP'14
- ▶ About typing constraints for ML:
The essence of ML type inference.
François Pottier and Didier Rémy.
Chapter 10 of ATAPL, 2005.

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list

About efficient implementations of ML inference

Rich types,
Tractable typing
– Type Inference –

Yann Régis-Gianas
yrg@irif.fr

- ▶ About first-order unification using Kaplan–Tarjan data structures:
Résolution d'équations dans des langages d'ordre 1, 2, . . . , ω .
Gérard Huet.
PhD thesis, Université Paris 7, 1976.
- ▶ About using efficient generalization decision:
Extending ML type system with a sorted equational theory.
Didier Rémy.
Technical Report 1766, INRIA, 1992.

What is type
inference?

ML

An implicitly typed language

Algorithm W

An explicitly typed language

Constraint based
approach

Conclusion

Reading list