Type Inference for Pharo

Pablo Tesone

September 30, 2015

Outline

1. Objectives
2. Definitions
3. Types in Pharo
4. Proposed Solution
5. Demo
6. Next Steps

Objectives

- Not type checking.

Objectives

- Not type checking.
- Provide information to the user / refactoring tools.

Objectives

- Not type checking.
- Provide information to the user / refactoring tools.
- Simplify the understanding of the program / Code navigation

Objectives

- Not type checking.
- Provide information to the user / refactoring tools.
- Simplify the understanding of the program / Code navigation
- Realtime tool.

Objectives

- Not type checking.
- Provide information to the user / refactoring tools.
- Simplify the understanding of the program / Code navigation
- Realtime tool.
- Memory / Processor efficiency

Definitions

- What is a type?

Definitions

- What is a type?
- Concrete types. *Bunch of classes*

Definitions

- What is a type?
- Concrete types. *Bunch of classes*
- Abstract types. *Bunch of messages*

Definitions

- What is a type?
- Concrete types. *Bunch of classes*
- Abstract types. *Bunch of messages*
- Typed and Untyped languages.

Definitions

- What is a type?
- Concrete types. *Bunch of classes*
- Abstract types. *Bunch of messages*
- Typed and Untyped languages.
- Type aware / Type unaware languages.

Definitions

- What is a type?
- Concrete types. *Bunch of classes*
- Abstract types. *Bunch of messages*
- Typed and Untyped languages.
- Type aware / Type unaware languages.
- Open world assumption

Definitions

- What is a type?
- Concrete types. *Bunch of classes*
- Abstract types. *Bunch of messages*
- Typed and Untyped languages.
- Type aware / Type unaware languages.
- Open world assumption
- Flow sensitivity / insensitivity.

Definitions

- What is a type?
- Concrete types. *Bunch of classes*
- Abstract types. *Bunch of messages*
- Typed and Untyped languages.
- Type aware / Type unaware languages.
- Open world assumption
- Flow sensitivity / insensitivity.
- Language conventions or idiomatic usages.

Is a Pharo programmer using types?

- We use them a lot, but without (or with limitated) support of the tools.

Is a Pharo programmer using types?

- We use them a lot, but without (or with limitated) support of the tools.
- Normally:
  - I know a bunch of concrete classes, like this message to this object I know it will return SmallInteger or Float.

Is a Pharo programmer using types?

- We use them a lot, but without (or with limitated) support of the tools.
- Normally:
  - I know a bunch of concrete classes, like this message to this object I know it will return SmallInteger or Float. **Concrete types**

Is a Pharo programmer using types?

- We use them a lot, but without (or with limitated) support of the tools.
- Normally:
  - I know a bunch of concrete classes, like this message to this object I know it will return SmallInteger or Float. **Concrete types**
  - I know that to this variable or expression I can send this bunch of messages.

Is a Pharo programmer using types?

- We use them a lot, but without (or with limitated) support of the tools.
- Normally:
  - I know a bunch of concrete classes, like this message to this object I know it will return SmallInteger or Float. **Concrete types**
  - I know that to this variable or expression I can send this bunch of messages. **Abstract types**

Is a Pharo programmer using types?

- We use them a lot, but without (or with limitated) support of the tools.
- Normally:
  - I know a bunch of concrete classes, like this message to this object I know it will return SmallInteger or Float. **Concrete types**
  - I know that to this variable or expression I can send this bunch of messages. **Abstract types**
  - I know that if this variable or expression receives this bunch of messages, it should be these implementors.

Is a Pharo programmer using types?

- We use them a lot, but without (or with limitated) support of the tools.
- Normally:
  - I know a bunch of concrete classes, like this message to this object I know it will return SmallInteger or Float. **Concrete types**
  - I know that to this variable or expression I can send this bunch of messages. **Abstract types**
  - I know that if this variable or expression receives this bunch of messages, it should be these implementors. **Converting abstract types to Concrete types**

Is a Pharo programmer using types?

- We use them a lot, but without (or with limitated) support of the tools.
- Normally:
  - I know a bunch of concrete classes, like this message to this object I know it will return SmallInteger or Float. **Concrete types**
  - I know that to this variable or expression I can send this bunch of messages. **Abstract types**
  - I know that if this variable or expression receives this bunch of messages, it should be these implementors. **Converting abstract types to Concrete types**
- The information is built in the programmers head.

Is a Pharo programmer using types?

- We use them a lot, but without (or with limitated) support of the tools.
- Normally:
  - I know a bunch of concrete classes, like this message to this object I know it will return SmallInteger or Float. **Concrete types**
  - I know that to this variable or expression I can send this bunch of messages. **Abstract types**
  - I know that if this variable or expression receives this bunch of messages, it should be these implementors. **Converting abstract types to Concrete types**
- The information is built in the programmers head.
- The programmer usually starts navigating all or part of the code.

Is a Pharo programmer using types?

- We use them a lot, but without (or with limitated) support of the tools.
- Normally:
  - I know a bunch of concrete classes, like this message to this object I know it will return SmallInteger or Float. **Concrete types**
  - I know that to this variable or expression I can send this bunch of messages. **Abstract types**
  - I know that if this variable or expression receives this bunch of messages, it should be these implementors. **Converting abstract types to Concrete types**
- The information is built in the programmers head.
- The programmer usually starts navigating all or part of the code.
- The programmer uses a lot of past experiences and conventions. Example: if you have a variable receiving *do:*, *add:* and *size*. It's somethins collection-like!

Is a Pharo programmer using types?

- So... we don't need you. We can do it by us!

Is a Pharo programmer using types?

- So... we don't need you. We can do it by us!
- Maybe... but the idea is to improve productivity, help the new comers and ease the learning curve.

Proposed Solution

- Real time / Iterative / Incremental algorithm.

Proposed Solution

- Real time / Iterative / Incremental algorithm. It updates the information as the programmer is writing.

Proposed Solution

- Real time / Iterative / Incremental algorithm. It updates the information as the programmer is writing.
- Open world assumption.

Proposed Solution

- Real time / Iterative / Incremental algorithm. It updates the information as the programmer is writing.
- Open world assumption. It works in a program-to-be, library or framework

Proposed Solution

- Real time / Iterative / Incremental algorithm. It updates the information as the programmer is writing.
- Open world assumption. It works in a program-to-be, library or framework
- Mixing Concrete and Abstract types

Proposed Solution

- Real time / Iterative / Incremental algorithm. It updates the information as the programmer is writing.
- Open world assumption. It works in a program-to-be, library or framework
- Mixing Concrete and Abstract types Giving all the information, you can use whatever you want.

Proposed Solution

- Real time / Iterative / Incremental algorithm. It updates the information as the programmer is writing.
- Open world assumption. It works in a program-to-be, library or framework
- Mixing Concrete and Abstract types Giving all the information, you can use whatever you want.
- Using *Human like* heuristics.

Proposed Solution

- Real time / Iterative / Incremental algorithm. It updates the information as the programmer is writing.
- Open world assumption. It works in a program-to-be, library or framework
- Mixing Concrete and Abstract types Giving all the information, you can use whatever you want.
- Using *Human like* heuristics. I want to add heuristics for idiomatics and normal usages of the environment
- Mixing static analysis and abstract program execution.

How does it works?

- It builds subtyping dependency graph in an static analysis, including sent messages to each expression.
- Then transverse the graph to get the information.
- Applying the subtype information, filtering by messages and using the heuristics.

How does it works? (II)

- This is performed in stages:
  - First the graph is created. This is performed by static analysis.

How does it works? (II)

- This is performed in stages:
  - First the graph is created. This is performed by static analysis. .
  - Once the graph is created. It can be transversed using contextual information, like the class in which I am asking the expression.

How does it works? (II)

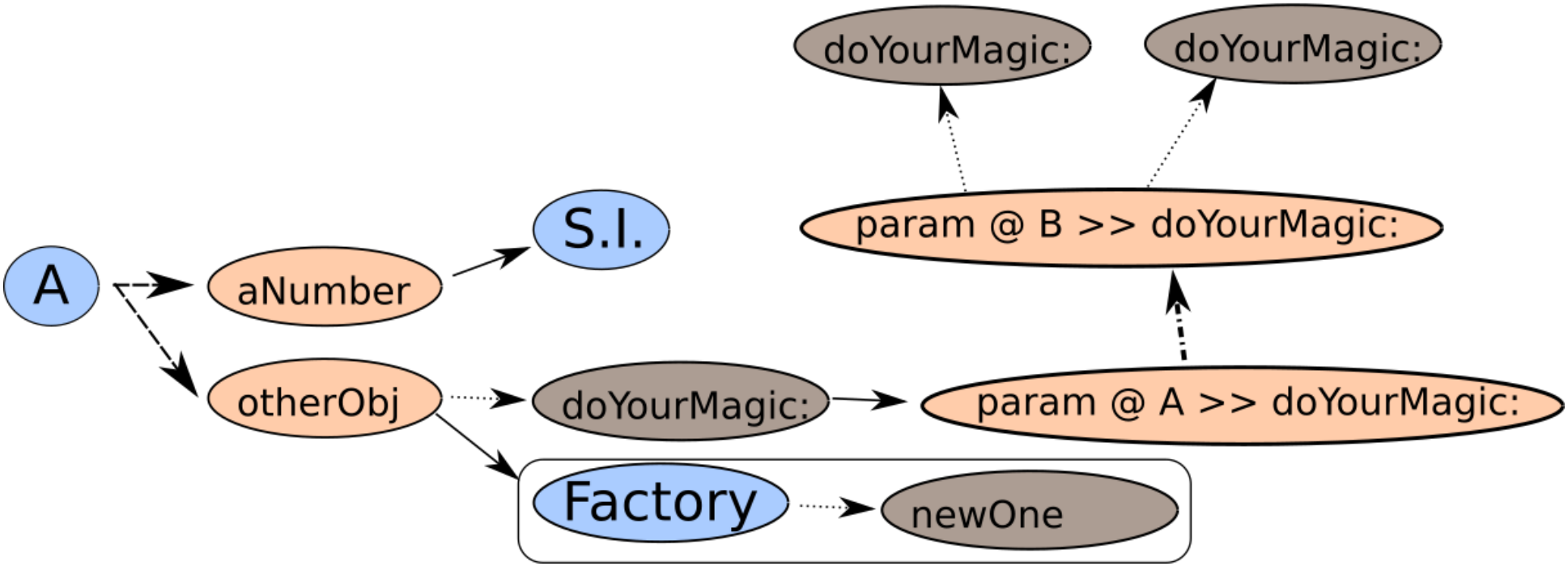
- This is performed in stages:
  - First the graph is created. This is performed by static analysis. .
  - Once the graph is created. It can be transversed using contextual information, like the class in which I am asking the expression.
  - If the code is changed, the graph has to be updated, only the changed parts (If not, it will not be incremental!)

How does it works? (III)

```
Class A (otherObject, aNumber)
A>>initialize
  aNumber := 23
  otherObj := Factory newOne.
A>>doYourMagic: param
  otherObj doYourMagic: param
```

```
Factory class >> newOne
  ^ B new
```

```
B>>doYourMagic: param
  param msg1 ; msg2
```



Demo

Next Steps

- General implementation improvements.

Next Steps

- General implementation improvements.
- Include more handling of collections and generics.

Next Steps

- General implementation improvements.
- Include more handling of collections and generics.
- Improving blocks handling.

Next Steps

- General implementation improvements.
- Include more handling of collections and generics.
- Improving blocks handling.
- Detecting *all* the updated needed after a change in the code

Next Steps

- General implementation improvements.
- Include more handling of collections and generics.
- Improving blocks handling.
- Detecting *all* the updated needed after a change in the code
- Implementing more heuristics, even *unsound* ones.

Next Steps

- General implementation improvements.
- Include more handling of collections and generics.
- Improving blocks handling.
- Detecting *all* the updated needed after a change in the code
- Implementing more heuristics, even *unsound* ones.
- Adding confidence factor.

Next Steps

- General implementation improvements.
- Include more handling of collections and generics.
- Improving blocks handling.
- Detecting *all* the updated needed after a change in the code
- Implementing more heuristics, even *unsound* ones.
- Adding confidence factor.
- Comparing with other solutions.

Next Steps

- General implementation improvements.
- Include more handling of collections and generics.
- Improving blocks handling.
- Detecting *all* the updated needed after a change in the code
- Implementing more heuristics, even *unsound* ones.
- Adding confidence factor.
- Comparing with other solutions.
- Testing with applications.

Next Steps

- General implementation improvements.
- Include more handling of collections and generics.
- Improving blocks handling.
- Detecting *all* the updated needed after a change in the code
- Implementing more heuristics, even *unsound* ones.
- Adding confidence factor.
- Comparing with other solutions.
- Testing with applications.

Comments!

- Continuation of a previous work with Nicolás Passerini.
- Same objectives and restrictions.
- A clean implementation.
- Includes different ideas, not constraint based, but the static analysis is nearly the same.
- The result of learning a lot side by side.

- Questions???

- Questions???
- Thanks a lot!