Live Typing

A Type System for Smalltalk

Hernán A. Wilkinson - @hernanwilkinson

10Pines founder – Professor at UBA

agile software development & services

Hernán Wilkinson

- 10Pines Founder
- FAST Founder
  (Fundación Argentina de Smalltalk)
- Professor at Computer Science Department at UBA
  ….
- mainly, passionate programmer

10 Pines

Type information in Dynamically Typed Languages has always been a topic of **research** due to many reasons ...

There are mainly two ways to have type information in dynamically typed languages:

- Type Inference
- Manual Type Annotation
  (generally combined with type inference)

Type Inference

A General Scheme for the
Automatic Inference of Variable Types†
Extended Abstract

by

Marc A. Kaplan
&
Jeffrey D. Ullman
Princeton University

Summary

We present the best known algorithm for the determination of run-time types in a programming language requiring no type declarations. We demonstrate that it is superior to other published algorithms and that it is the best possible algorithm from among all those that use the same set of primitive operators.

I Introduction

II A Model of Computation in a Programming Language

The basic building block of our programming language is the parallel assignment statement, whose most general form is Q:

$$(X_1, X_2, X_3, \ldots, X_k) \leftarrow (\oplus_{i_1}(Y_{11}, Y_{12}, \ldots, Y_{1d_1}), \oplus_{i_2}(Y_{21}, Y_{22}, \ldots, Y_{2d_2}),$$

1980

Inferring Types in Smalltalk

Norihisa Suzuki
Xerox Palo Alto Research Centers
3333 Coyote Hill Rd., Palo Alto, CA 94304

1. Introduction

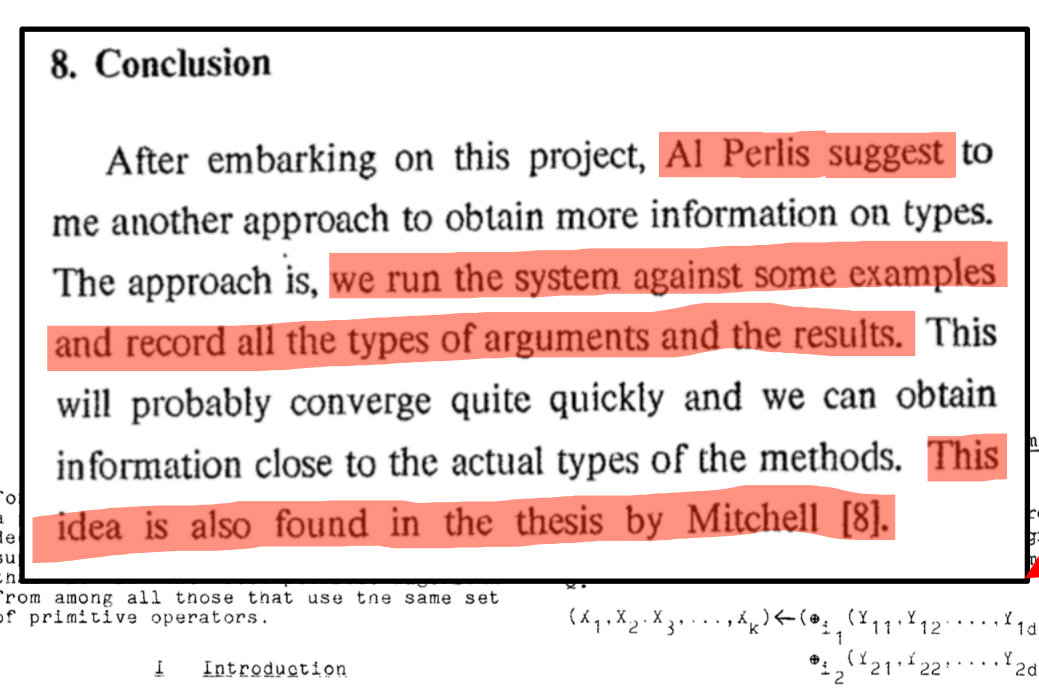
Smalltalk is an object-oriented language designed and implemented by the Learning Research Group of the Xerox Palo Alto Research Center [2, 5, 14]. Some features of this language are: abstract data classes, information inheritance by a superclass-subclass mechanism, message passing semantics, extremely late binding, no type

completed system, and when he discovers a run-time error caused by an unimplemented procedure, he can write the procedure body and proceed the computation from the point where the error was discovered. However, there is no way to guarantee that there will be no run-time errors. We found many "completed" systems which still had such run-time errors.

1981

10 Pines

Type Inference



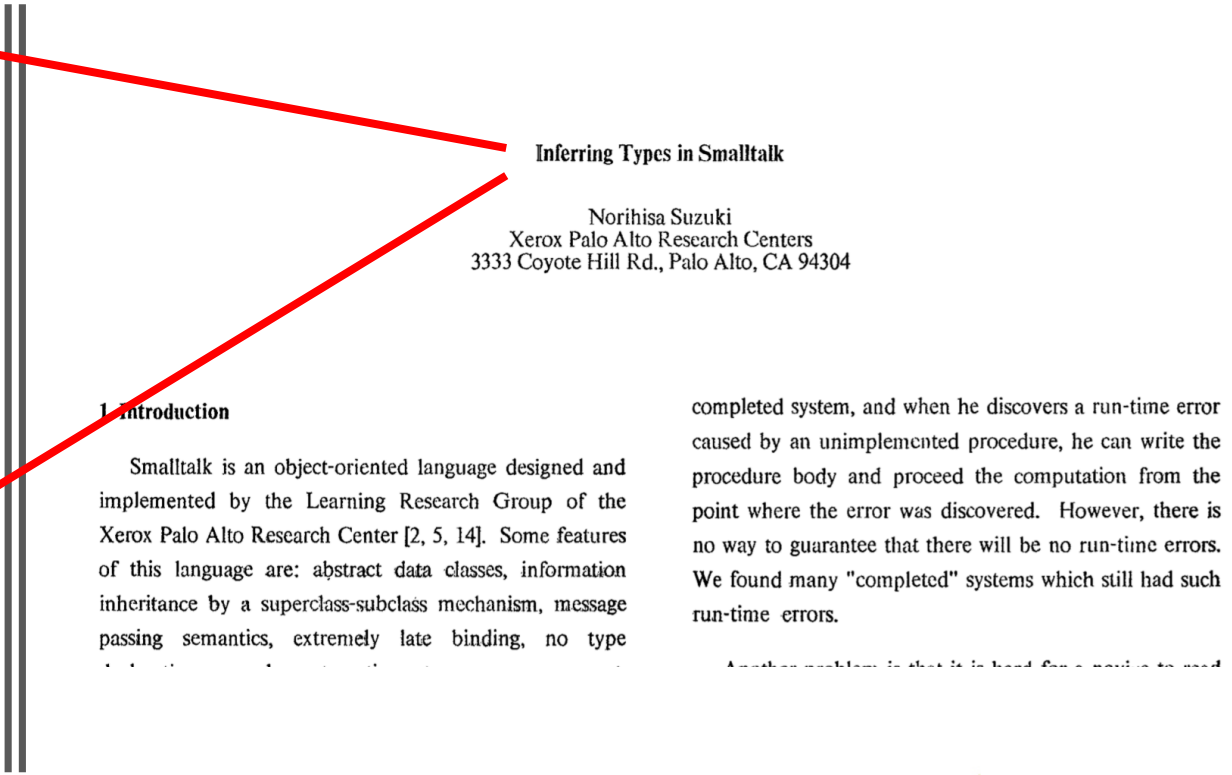
8. Conclusion

After embarking on this project, Al Perlis suggest to me another approach to obtain more information on types. The approach is, we run the system against some examples and record all the types of arguments and the results. This will probably converge quite quickly and we can obtain information close to the actual types of the methods. This idea is also found in the thesis by Mitchell [8].

from among all those that use the same set of primitive operators.

I Introduction

$(x_1, x_2, x_3, \ldots, x_k) \leftarrow (\oplus_{i_1}(y_{11}, y_{12}, \ldots, y_{1d_1}), \oplus_{i_2}(y_{21}, y_{22}, \ldots, y_{2d_2}),$



Inferring Types in Smalltalk

Norihisa Suzuki
Xerox Palo Alto Research Centers
3333 Coyote Hill Rd., Palo Alto, CA 94304

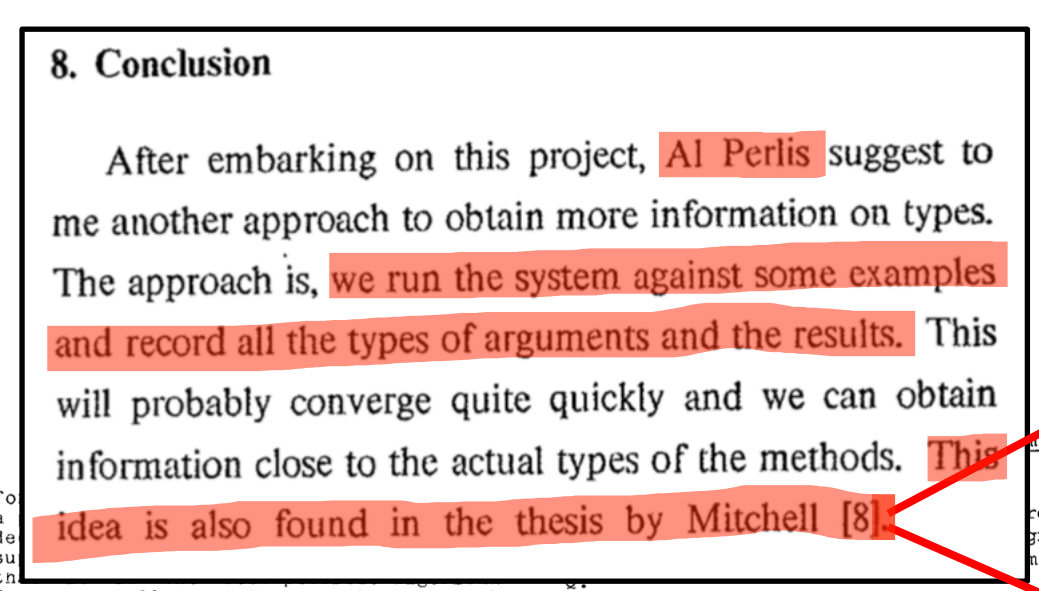
1. Introduction

Smalltalk is an object-oriented language designed and implemented by the Learning Research Group of the Xerox Palo Alto Research Center [2, 5, 14]. Some features of this language are: abstract data classes, information inheritance by a superclass-subclass mechanism, message passing semantics, extremely late binding, no type

completed system, and when he discovers a run-time error caused by an unimplemented procedure, he can write the procedure body and proceed the computation from the point where the error was discovered. However, there is no way to guarantee that there will be no run-time errors. We found many "completed" systems which still had such run-time errors.

10 Pines

Type Inference



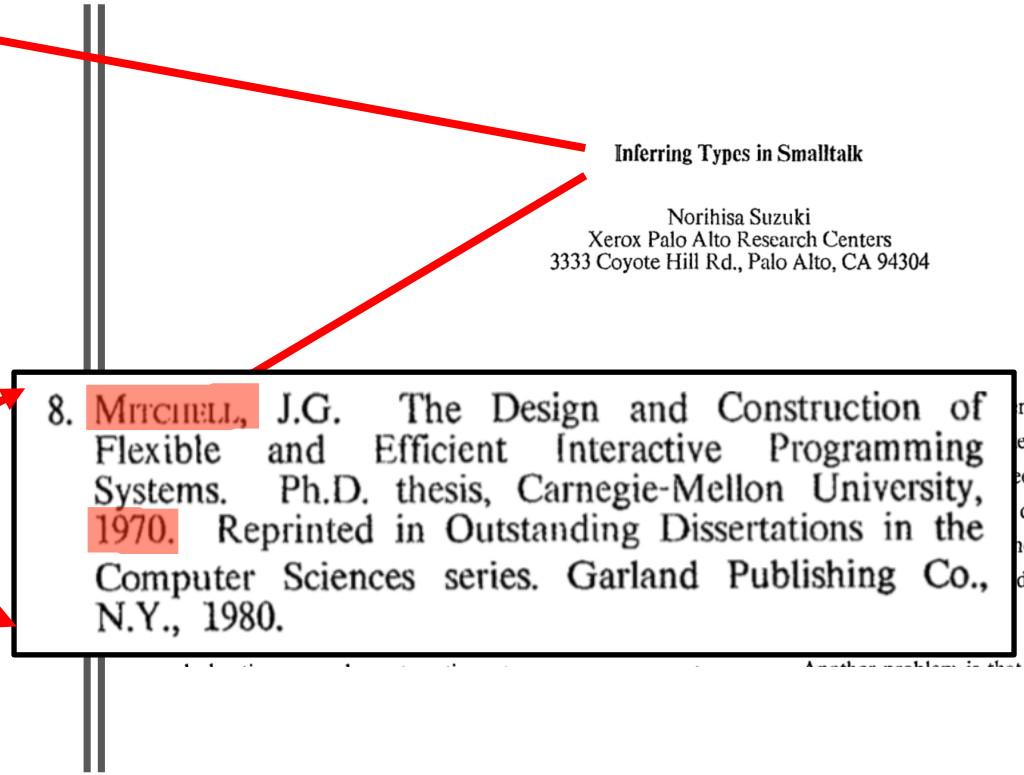
8. Conclusion

After embarking on this project, Al Perlis suggest to me another approach to obtain more information on types. The approach is, we run the system against some examples and record all the types of arguments and the results. This will probably converge quite quickly and we can obtain information close to the actual types of the methods. This idea is also found in the thesis by Mitchell [8].

from among all those that use the same set of primitive operators.

I Introduction

$(x_1, x_2, x_3, \ldots, x_k) \leftarrow (\oplus_{i_1}(y_{11}, y_{12}, \ldots, y_{1d_1}), \oplus_{i_2}(y_{21}, y_{22}, \ldots, y_{2d_2}),$



Inferring Types in Smalltalk

Norihisa Suzuki
Xerox Palo Alto Research Centers
3333 Coyote Hill Rd., Palo Alto, CA 94304

8. MITCHELL, J.G. The Design and Construction of Flexible and Efficient Interactive Programming Systems. Ph.D. thesis, Carnegie-Mellon University, 1970. Reprinted in Outstanding Dissertations in the Computer Sciences series. Garland Publishing Co., N.Y., 1980.

n he discovers a run-time error
ed procedure, he can write the
eed the computation from the
discovered. However, there is
here will be no run-time errors.
d" systems which still had such

10 Pines

Disadvantages

- Slow
- Incomplete
- Invalid information
- …

Manual Type Annotation

A Type Declaration and Inference System for Smalltalk

Alan H. Borning
Computer Science Dept., University of Washington

Daniel H. H. Ingalls
Xerox Palo Alto Research Center

Abstract

An experimental system for declaring and inferring type in Smalltalk is described. (In the current Smalltalk language, the programmer supplies no type declarations.) The system provides the benefits of type declaration in regard to compile-time checking and documentation, while still retaining Smalltalk's flexibility. A type hierarchy, which is integrated with the existing Smalltalk class hierarchy, allows one type to inherit the traits of another type. A type may also have parameters, which are in turn other types.

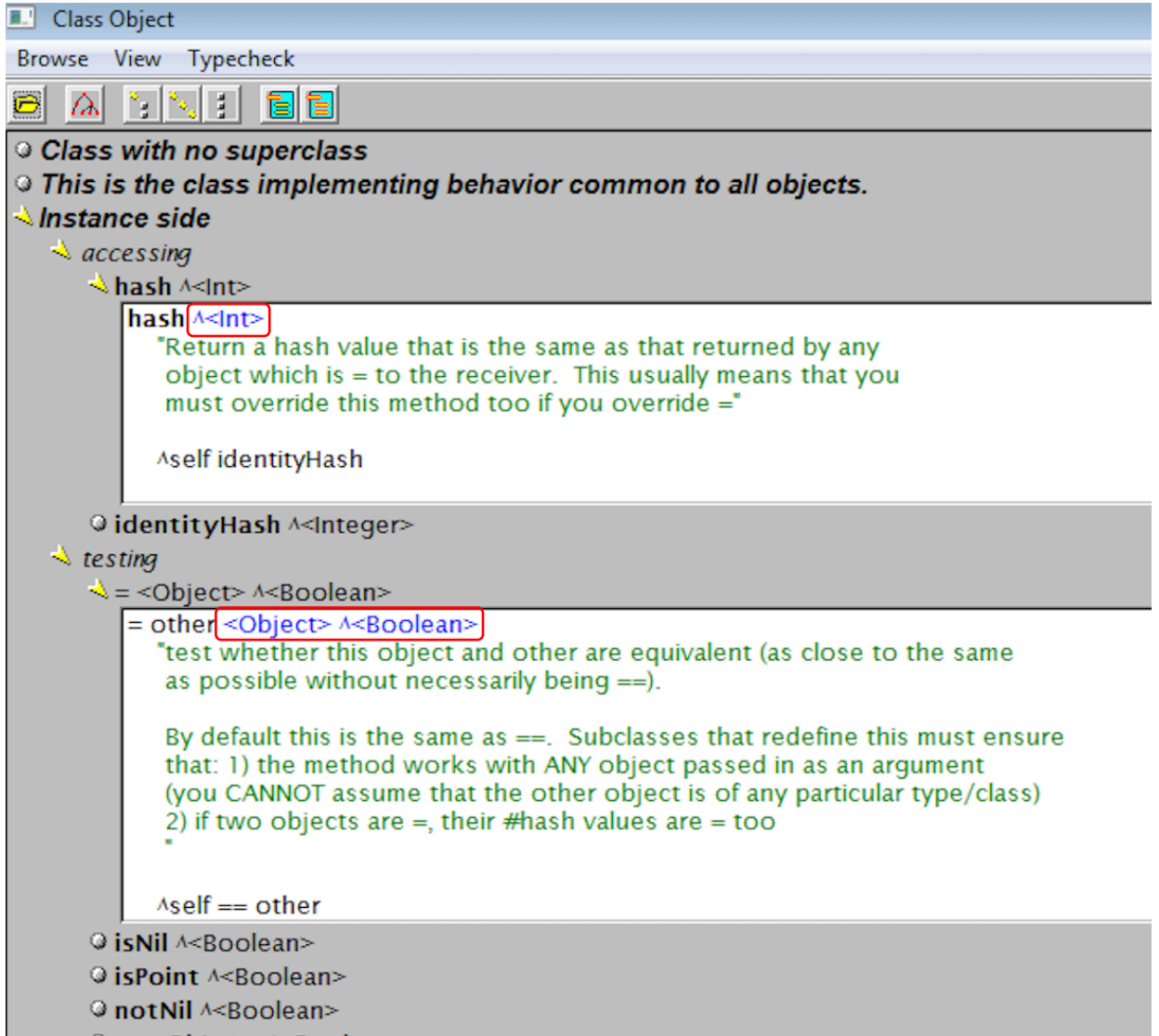
inappropriate class will only result i
"message not understood", it is n
the programmer to be informed of s
in question is being compiled, rath
used.

In this paper we present an expe
and inferring type in Smalltalk. In a
language, we do not wish to give

1982!!

```
Boolean
   supertype: Object
    & b <Boolean>
       ↑<Boolean>
    and: aBlock <Block to: Boolean>
       "This version of 'and' evaluates
        its argument only if necessary."
       ↑<Boolean>
    ifTrue: t <Block to: Object>
      ifFalse: f <Block to: Object>
       ↑<t resultType nearestCommonSupertype:
          f resultType>
    ifTrue: t <Block to: Object>
       ↑<t resultType>
       "If the receiver is 'false', then
        the result is nil.  (As described
        in Section 5, nil is allowed as an
        instance of any type, and hence it
        satisfies the declaration
        t resultType.)"
    ifFalse: f <Block to: Object>
       ↑<f resultType>
```

Manual Type Annotation



Strongtalk
1994

10 Pines

Manual Type Annotation

```
function add(x: number, y: number): number {
    return x + y;
}

let myAdd = function(x: number, y: number): number { return x + y; };
```

TypeScript
~ 2012/2014

10 Pines

Manual Type Annotation

```
5   class CreditCard:
6       def __init__(self, number: str, expiration_date: MonthOfYear):
7           self.number = number
8           self.expiration_date = expiration_date
9
10      def is_expired_on(self, limit: date) -> bool:
11          return self.is_expired_by_year(limit) or self.is_expired_by_month(limit)
12
13      def is_expired_by_month(self, limit: date) -> bool:
14          return limit.year == self.expiration_date.year and\
15                 limit.month > self.expiration_date.month
16
17      def is_expired_by_year(self, limit: date) -> bool:
18          return limit.year > self.expiration_date.year
19
```

Python 3.5 - Type Hints

10 Pines

Manual Type Annotation

```ruby
# typed: true
class A
  extend T::Sig

  sig {params(x: Integer).returns(String)}
  def bar(x)
    x.to_s
  end
end

def main
  A.new.barr(91)   # error: Typo!
  A.new.bar("91")  # error: Type mismatch!
end
```

Ruby with Sorbet ~ 2018

10 Pines

Let’s see an example with **Python 3.9** and **PyCharm 2020.3**

Disadvantages

- Language syntax has to be changed
- Code is harder to read due to type annotations
  - It is not like it was before …
- The programmer must annotate the types
- The programmer must maintain the annotations!
- They concentrate on type checking
- Tool improvements rely on each particular IDE

What is **Live Typing**?

There are mainly ~~two~~ three ways to have type information in dynamically typed languages:

- Type Inference
- Manual Type Annotation
- Automatic Type Collection

Live Typing

Automatic type collection (done by the VM)

+

Tools that improve the development experience

As we saw, it is **not a new** idea…
but a **particular** one with a
working implementation

Live Typing's goal is not **Type Checking**

Its goal is to **improve the tools** to facilitate the programmer's tasks

(although it provides a type checking solution)

10 Pines

Implementation

Instance variables

```
Behavior subclass: #ClassDescription
    instanceVariableNames: 'instanceVariables organization instanceVariablesRawTypes'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Kernel-Classes'
```

10 Pines

Instance variables – VM Change

- Every time a newObject is assigned to a variable, the VM stores ”newObject class” into “instanceVariablesRawTypes at: (self indexOf: variable)”

```
keepInstanceVariableTypeInformationFor: anAssignedObject in: rcvr at: instVarIndex

    <inline: true>

    | instVarsTypes rcvrClass rcvrClassTag |

    rcvrClassTag := objectMemory fetchClassTagOf: rcvr.
    rcvrClass := objectMemory classForClassTag: rcvrClassTag.

    instVarsTypes := objectMemory followObjField: InstanceVariablesRawTypesIndex ofObject: rcvrClass.
    self keepTypeInformationIn: instVarsTypes at: instVarIndex for: anAssignedObject.
```

10 Pines

Instance variables – VM Change

```
keepTypeInformationIn: allVarsTypes at: anIndex for: anAssignedObject

    <inline: true>

    | types |

    (self isInstanceOfClassArray: allVarsTypes) ifTrue: [
        anIndex < (objectMemory lengthOf: allVarsTypes) ifTrue:  [
            types := objectMemory followObjField: anIndex ofObject: allVarsTypes.
            self keepTypeInformationIn: types for: anAssignedObject ]]
```

10 Pines

Instance variables – VM Change

```
keepTypeInformationIn: types for: anAssignedObject

    | assignedObjectClass assignedObjectClassTag typesSize |

    types = objectMemory nilObject ifTrue: [ ^self ].

    assignedObjectClassTag := objectMemory fetchClassTagOf: anAssignedObject.
    assignedObjectClass := objectMemory classForClassTag: assignedObjectClassTag.

    typesSize := objectMemory lengthOf: types.
    0 to: typesSize-1 do: [ :index | | typeAtIndex |
        typeAtIndex := objectMemory followObjField: index ofObject: types.
        typeAtIndex == assignedObjectClass ifTrue: [ ^self ].
        typeAtIndex == objectMemory nilObject ifTrue: [
            ^objectMemory storePointer: index ofObject: types withValue: assignedObjectClass ]].
```

10 Pines

Instance variables

- instanceVariablesRawTypes can be nil.
  - It means we don’t want to store types for that class instance variables
- instanceVariablesRawTypes at: instVarIndex
  - Can be nil if we don’t want to store types for that instance variable
  - It can have different sizes per instance variable to adjust memory consumption and speed

Method Type Information

- New AdditionalMethodState instance variables:
  - variablesTypes: Keeps arguments and temporaries types. Same structure as instanceVariablesRawTypes
  - returnTypes: Keeps return types

```
Object variableSubclass: #AdditionalMethodState
    instanceVariableNames: 'method selector variablesTypes returnTypes'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Kernel-Methods'
''
```

Method Type Information – VM Changes

keepArgumentTypes

```
<inline: true>

| additionalMethodState tempVarsTypes maxNumberOfArguments types |

argumentCount > 0 ifTrue: [
    additionalMethodState := self additionalMethodStateOf: newMethod.
    additionalMethodState = objectMemory nilObject ifFalse: [
        tempVarsTypes := objectMemory followObjField: 2 ofObject: additionalMethodState.
        tempVarsTypes = objectMemory nilObject ifFalse: [
            (self isInstanceOfClassArray: tempVarsTypes) ifTrue: [
                maxNumberOfArguments := (objectMemory lengthOf: tempVarsTypes) min: (argumentCount-1).
                0 to: maxNumberOfArguments do: [ :argIndex |
                    types := objectMemory followObjField: argIndex ofObject: tempVarsTypes.
                    self keepTypeInformationIn: types for: (self internalStackValue: argIndex)]]]].
```

Method Type Information – VM Changes

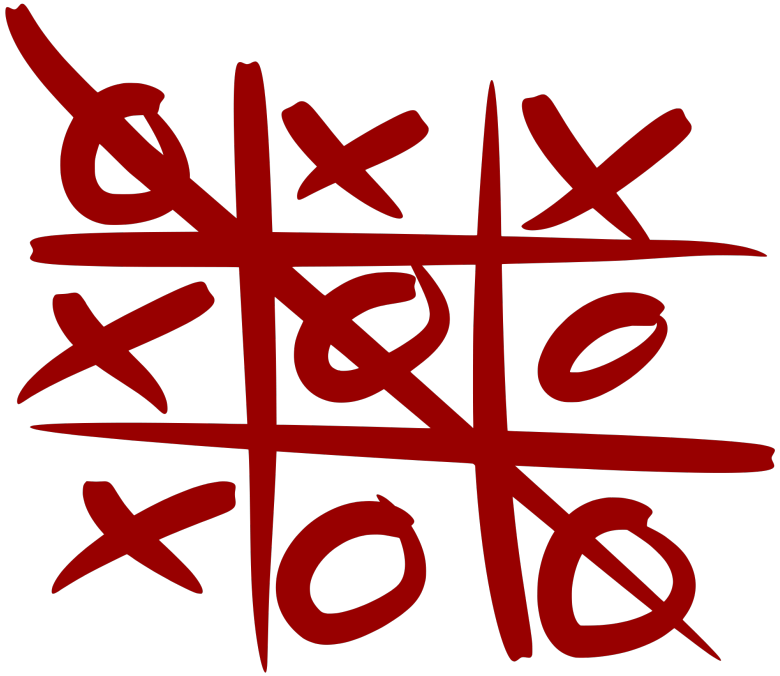
```
keepTypeAndSetTemporary: tempIndex in: theFP put: anAssignedObject

    <inline: true>

    | frameMethod additionalMethodState tempVarsTypes |

    self temporary: tempIndex in: theFP put: anAssignedObject.

    frameMethod := self frameMethod: theFP.
    additionalMethodState := self additionalMethodStateOf: frameMethod.
    additionalMethodState = objectMemory nilObject ifFalse: [
        tempVarsTypes := objectMemory followObjField: 2 ofObject: additionalMethodState.
        self keepTypeInformationIn: tempVarsTypes at: tempIndex for: anAssignedObject ]
```

Method Type Information – VM Changes

```
keepReturnObjectType

    <inline: true>

    | frameMethod additionalMethodState returnTypes |

    frameMethod := self frameMethod: localFP.
    additionalMethodState := self additionalMethodStateOf: frameMethod.
    additionalMethodState = objectMemory nilObject ifFalse: [
        returnTypes := objectMemory followObjField: 3 ofObject: additionalMethodState.
        self keepTypeInformationIn: returnTypes for: localReturnValue ]
```

10 Pines

Tools Examples



TicTacToe

Showing/Managing Types

Autocompletion

DynamicType
(SelfType, ClassType, InstanceType)

Actual Implementors

Actual Senders
Sure and Possible
(Per Message Send analysis)

Refactorings with Actual Scope

Type Checker

(we know for sure when *nil* is assigned!)

Type Check Morph & Behavior

Some statistics

- InstanceVariablesTypes numberOfTypesForAll
- InstanceVariablesTypes numberOfRawTypesForAll
- ...

Performance

	Typed VM	Stack VM	Difference
Aconcagua Tests	37 ms	22 ms	1.6 x
Chalten Tests	2400 ms	2204 ms	1.08 x
Refactoring Tests	56382 ms	39650 ms	1.42 x
TicTacToe Tests	3 ms	2 ms	1.5 x
Some Kernel Tests	220 ms	151 ms	1.45 x
Average			**1.41 x**

The important thing is that the programmer does not notice the difference when programming

10 Pines

Memory

Typed Image - Full	Common Image	Difference
25 MB	17 MB	1.47 x

Conclusion

With **no extra effort**,
we can have **type information**,
and **get rid off** most of the
disadvantages of a
dynamically type language

The programmer does not have to **maintain** the types.

Types do not **interfere** when reading code

Live Typing makes types **explicit**
You do not have to **remember** or **infer** them anymore

It is a very **simple** technique that **heavily improves** the **programming experience**

It does not change the **syntax**
It does not stop you from **compiling**
It does not force you to **use it**
Types are not in the **source code**

I humbly believe it
respects and **honors**
the Smalltalk **spirit**

I hope that, some day, we will stop saying that **Smalltalk** is **Dynamically Typed** and start saying it is **Lively Typed**

Future Work

Under development

- SOON Annotate types in closure parameters and variables (Ines Sosa)
- SOON Support for Parameterized types (Generics) is needed for collections, association, etc. (Collection<T>, Association<K,V>, etc.) (Ana Felisatti & Mariano De Sousa)
- SOON Implement it on the JIT VM (?)
- SOON More refactorings types aware like inline (Fernando Balboa)

10 Pines

More Ideas

- Add more type cast cases in the Type Checker
- Check for parameter types (Freeze annotated types)
- Use Type Checker infrastructure to improve even more the autocomplete
  - Suggest only the objects that type check for parameters
- Import type info from production images to development images
- Improve Type Checker to warn about dead code
- Delete method with transitive closure of actual sends in that method
- Change the COMPILER (not the VM) to generate and initialize the PIC at compile time!!

Download it from:

- CuisUniversity: http://www.cuisuniversity.org/descargas

- The VM: https://github.com/hernanwilkinson/LiveTyping/tree/master/Smalltalk/VMs
- The repo: https://github.com/hernanwilkinson/LiveTyping

10 Pines

Questions?

Thanks!

@HernanWilkinson – hernan.wilkinson@10pines.com – www.10pines.com

10 Pines