A call-by-need basis

Olivier Danvy

& Ian Zerny (now at Google)

Programming-languages Group

23 May 2014
Reminders (1/4)

- A programming language
  is a notation for expressing computations.

Analogy: language and thought.
Reminders (1/4)

• A programming language is a notation for expressing computations.

• Programs are written in this notation.

Analogy: recipes in a cookbook.
Reminders (2/4)

Syntactic units:

• expressions
• commands / statements
• declarations
• types
• etc.
Reminders (3/4)

Named and parameterized syntactic units:

- functions
- procedures / methods
- modules
- classes
- etc.
Reminders (4/4)

Formal and actual parameters:

- call by value
- call by name
- call by need
- etc.
The topic of this talk: call by need

A notation to express computations that are

• demand driven, and

• where intermediate results are memoized.
The topic of this talk: call by need

A notation to express computations that are

- demand driven, and
- where intermediate results are memoized.

Are there any questions?
Pay attention: I will say zis only once
The two key features of call by need

- demand-driven computation
- memoization of intermediate results
A bewildering amount of related work

• in theory (semantics)
• in practice (implementation)
• in the middle (abstract machines)
Theory and practice of call by need

- In theory, we know why it works.
- In practice, we know how to make it works.

But do theory and practice agree?

Surprisingly, nobody knew.
Our contribution

A grand unified account of call by need solving

a problem that was open since the 1970’s.
Our unified account

Surprisingly simple in retrospect:

• The **syntactic correspondence** and the **functional correspondence**, as developed here at AU.

• **Lock-step equivalence** (bisimilarity), as taught here at AU.

And also, non-trivially: K.I.S.S.
Impact

• peer-reviewed articles in conferences
• peer-reviewed articles in journals
• tips of the hat in various scientific blogs
• followup peer-reviewed articles by others
• invited talks for Olivier
• elite-research funding for Ian
Another sort of impact

- Path-dependent types in Scala.
- Wild cards in Java.
Another sort of impact

- Path-dependent types in Scala.
- Wild cards in Java.

They originate in Erik Ernst’s PhD thesis.
Another sort of impact

- Path-dependent types in Scala.
- Wild cards in Java.

They originate in Erik Ernst’s PhD thesis.

Good bye, Erik.
Another sort of impact

- Path-dependent types in Scala.
- Wild cards in Java.

They originate in Erik Ernst’s PhD thesis.

Good bye, Erik.

And thanks.