Type Inference in Object-Oriented Languages with Classes for Linguistic Engineering

Pierre Crescenzo
Pierre.Crescenzo@unice.fr
http://www.crescenzo-pierre.nom.fr/

Jean-Louis Paquelin
paquelin@i3s.unice.fr
http://www.anima.net/jlpaquelin/

Laboratoire I3S (UNSA-CNRS)
2000, route des lucioles
Les Algorithmes, bâtiment Euclide B
BP 121
F-06903 Sophia Antipolis CEDEX (France)

Context and Goal

- Context

  $\Longrightarrow$ Linguistic Engineering

- Goal

  $\Longrightarrow$ Type Inference in Object-Oriented Languages with Classes

A General Example from Botany (1/2)

- We can see a plant (*object*).
  $\Longrightarrow$ its species (*type*) are Plant

- It has green leaves (*value* and *attribute*)
  $\Longrightarrow$ its species are Chlorophyllian (*subtype* of Plant)

- and flowers (*attribute*).
  $\Longrightarrow$ its species are Phanerogam (*subtype* of Chlorophyllian)

A General Example from Botany (2/2)

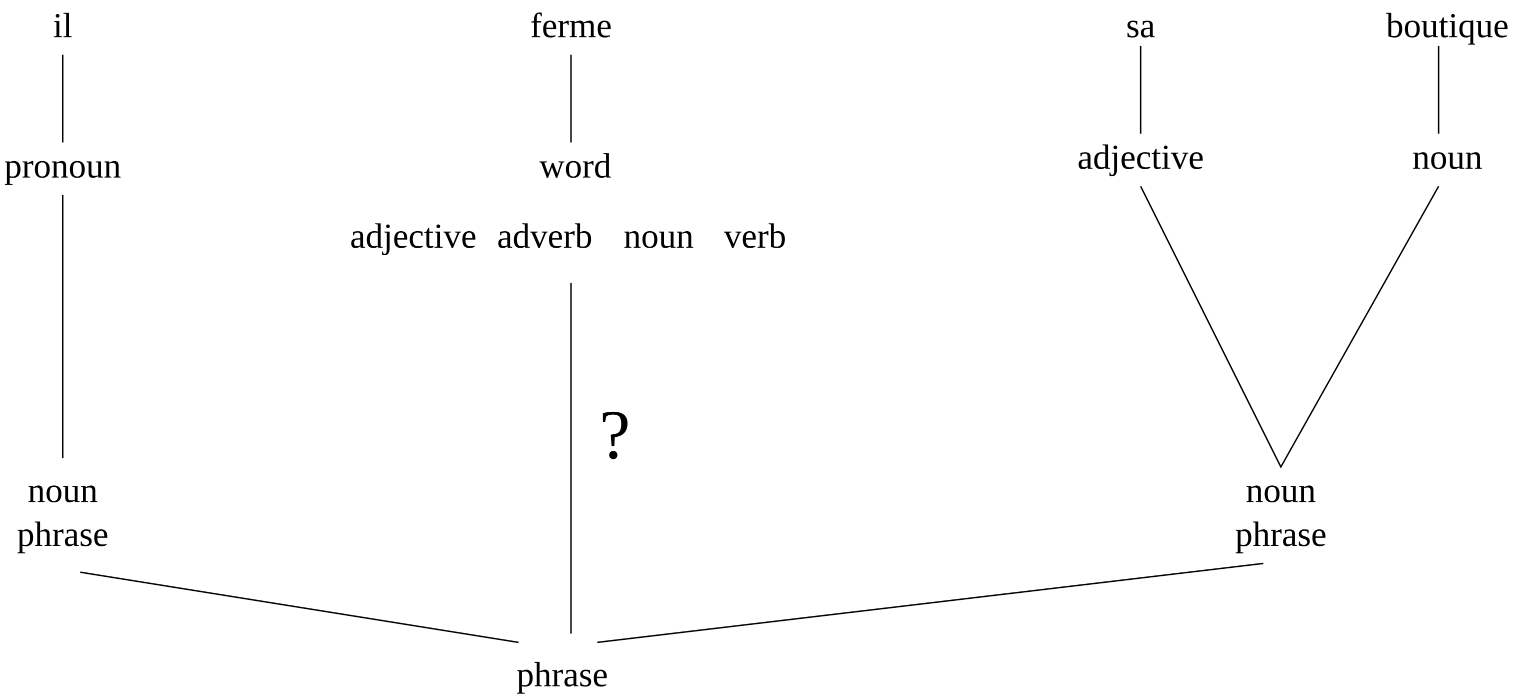
At the end of the analysis, we can deduce:

– either the exact species of the plant,

– or a set of possible species for the plant,

– or a species discovery or an analysis error.

An Example from Linguistics



he closes his shop

About Object-Orientation

- **Reification**
- **Encapsulation**
- **Specialization**

- Class and Type
- Primitives: Attribute and Value, Method and Behavior
- Object and Instance
- Utilization Links: Aggregation and Composition
- Importation Link : Inheritance
- Compatibility
- Migration

First Steps to a Solution

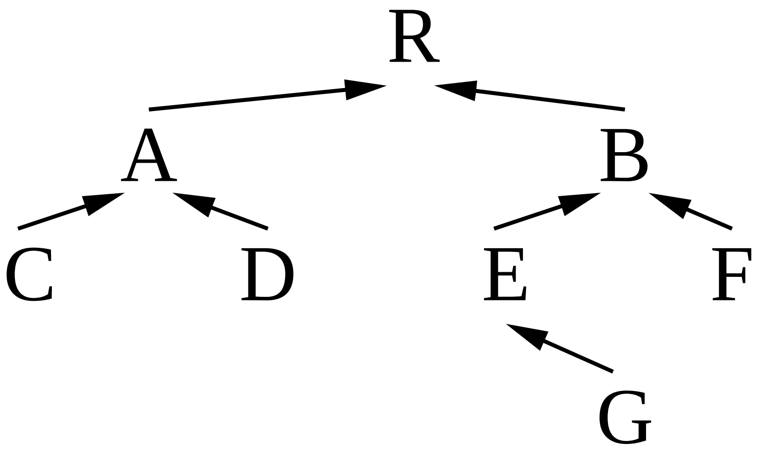
1. Instanciation with the most general type

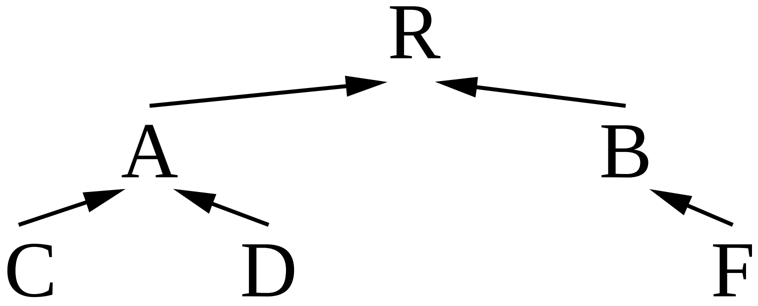
2. Data acquisition

3. Type refining (migration)

4. Results interpretation

Our Type System: Typological Domain (1/2)





x is not of type E

Our Type System: Typological Domain (2/2)

Results Interpretation:

- The final typological domain

  - contains only one type, or

  - is empty, or

  - contains several types.

Constraint-Controlled Migration

$\mathsf{o.p} = \mathsf{3}$

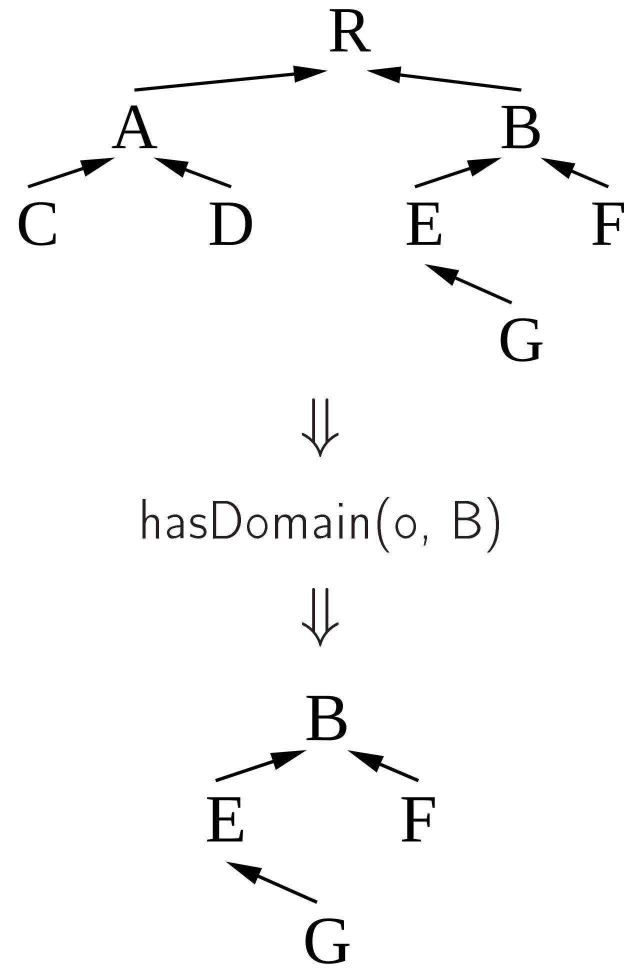
$\Rightarrow$ o has a primitive named p

$\Rightarrow$ p of o is an $\mathsf{Integer}$

hasDomain Constraint

hasDomain(o, D)

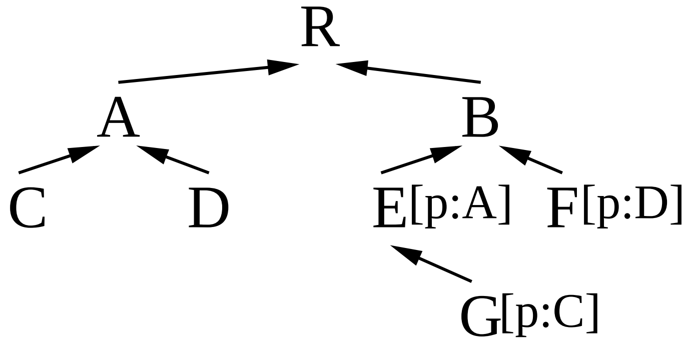
1. The new typological domain (TD) of o is the intersection of its current TD with D.
2. This constraint is recursively propagated over all the objects which reference and are referenced by the new o.



hasPrimitive Constraint

hasPrimitive(o, p)

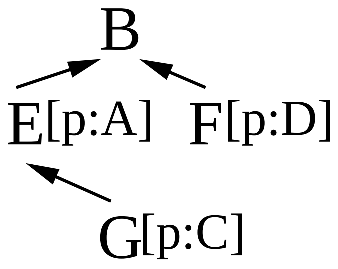
1. If o hasn't a p primitive and there is at least one p primitive in the current TD of o, then a p primitive is added to o. The new TD of o and p are refined.
2. This constraint is recursively propagated over all the objects which reference and are referenced by the new o and p.



⇓

hasPrimitive(o, p)

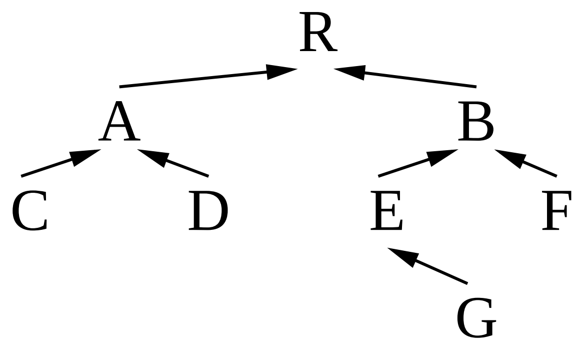
⇓



hasNotType Constraint

hasNotType(o, T)

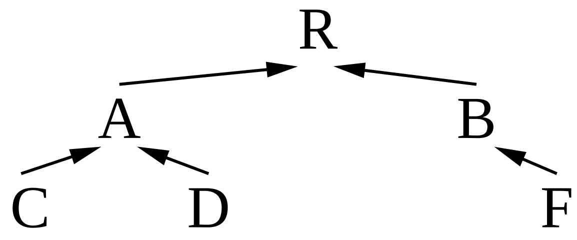
1. The new TD of o is the current TD of o minus the TD descended from T.
2. This constraint is recursively propagated over all the objects which reference and are referenced by the new o.



⇓

hasNotType(o, E)

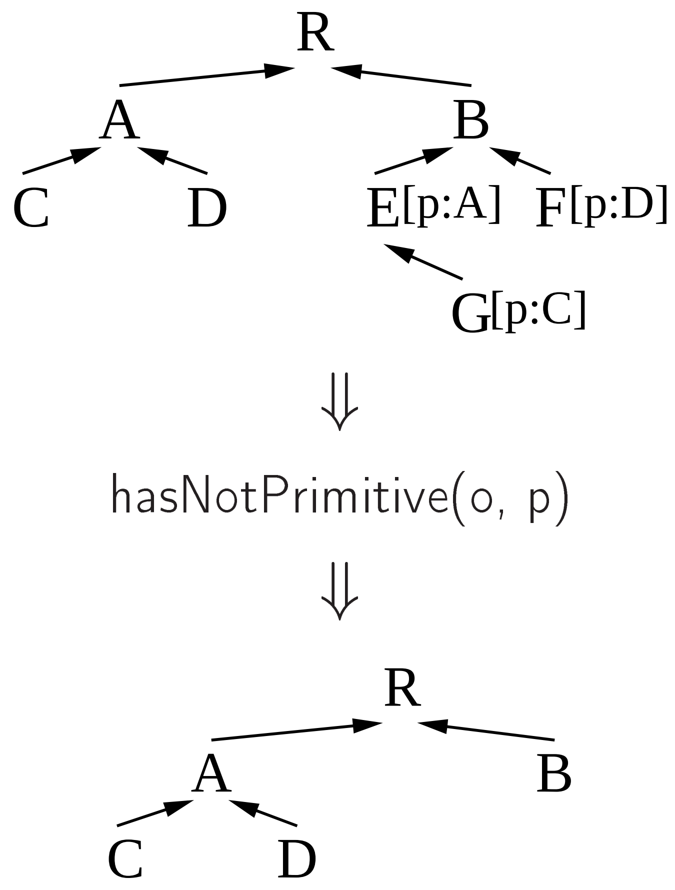
⇓



hasNotPrimitive Constraint

hasNotPrimitive(o, p)

1. The new TD of o is the smallest subtree (of the current TD of o) which doesn't contains p.
2. This constraint is recursively propagated over all the objects which reference and are referenced by the new o.



⇓

hasNotPrimitive(o, p)

⇓

About Language Operators

creation of o with type T	allocate(o)
	hasDomain(o, domainOf(T))
	initialize(o)

access to p primitive of o	hasPrimitive(o, p) o.p

assignment of o' to o	hasDomain(o, domainOf(o'))
	o = o'

Conclusion and Prospects

Our current work:

- extends the type system of Object-Oriented Languages with Classes,

- provides a type inference mechanism,

- could be adapted to handle exceptions, and

- will be prototyped in CLOS then implemented as an extension of Java.