“Real World” as an argument for covariant specialization in programming and modeling

or

Specialization, from Aristotle to UML

Roland Ducournau
LIRMM – Université Montpellier 2

Plan

1. From Aristotle to KR, Specialization is Covariant,
2. According to type theory, Subtyping is Contravariant,
3. The conflict and its causes,
4. Are Programming Languages Type Safe or Covariant?
5. Is UML Aristotle's champion?
6. Have we anything to expect from type theory?
7. Perspectives: real covariant programming and modeling languages

Plan

1. From Aristotle to KR, Specialization is Covariant,
2. According to type theory, Subtyping is Contravariant,
3. The conflict and its causes,
4. Are Programming Languages Type Safe or Covariant?
5. Is UML Aristotle's champion?
6. Have we anything to expect from type theory?
7. Perspectives: real covariant programming and modeling languages

Plan

1. From Aristotle to KR, Specialization is Covariant,
2. According to type theory, Subtyping is Contravariant,
3. The conflict and its causes,
4. Are Programming Languages Type Safe or Covariant?
5. Is UML Aristotle's champion?
6. Have we anything to expect from type theory?
7. Perspectives: real covariant programming and modeling languages

Plan

1. From Aristotle to KR, Specialization is Covariant,
2. According to type theory, Subtyping is Contravariant,
3. The conflict and its causes,
4. Are Programming Languages Type Safe or Covariant?
5. Is UML Aristotle's champion?
6. Have we anything to expect from type theory?
7. Perspectives: real covariant programming and modeling languages

Plan

1. From Aristotle to KR, Specialization is Covariant,
2. According to type theory, Subtyping is Contravariant,
3. The conflict and its causes,
4. Are Programming Languages Type Safe or Covariant?
5. Is UML Aristotle's champion?
6. Have we anything to expect from type theory?
7. Perspectives: real covariant programming and modeling languages

Plan

1. From Aristotle to KR, Specialization is Covariant,
2. According to type theory, Subtyping is Contravariant,
3. The conflict and its causes,
4. Are Programming Languages Type Safe or Covariant?
5. Is UML Aristotle's champion?
6. Have we anything to expect from type theory?
7. Perspectives: real covariant programming and modeling languages

Plan

1. From Aristotle to KR, Specialization is Covariant,
2. According to type theory, Subtyping is Contravariant,
3. The conflict and its causes,
4. Are Programming Languages Type Safe or Covariant?
5. Is UML Aristotle's champion?
6. Have we anything to expect from type theory?
7. Perspectives: real covariant programming and modeling languages

Specialization is Covariant (1/3)

- Specialization has ancient roots in Aristotelian tradition:
  Socrates is a human, humans are mortals, thus Socrates is a mortal

Specialization is Covariant (1/3)

- Specialization has ancient roots in Aristotelian tradition:
  Socrates is a human, humans are mortals, thus Socrates is a mortal

- In object-oriented jargon, inclusion of extensions:
  instances of a class are also instances of its superclasses

$$B \prec A \Longrightarrow Ext(B) \subseteq Ext(A)$$

Specialization is Covariant (2/3)

- Classes have properties, yielding to inclusion of intensions:

$$B \prec A \Longrightarrow Int(A) \subseteq Int(B)$$

this is inheritance

Specialization is Covariant (2/3)

- Classes have properties, yielding to inclusion of intensions:

$$B \prec A \Longrightarrow Int(A) \subseteq Int(B)$$

  this is inheritance

- Properties have a domain:

$$B \prec A \ \ \& \ \ P \in Int(A) \Longrightarrow Dom(B,p) \subseteq Dom(A,p)$$

  this is covariant specialization of domains

Specialization is Covariant (3/3)

Knowledge representation, especially Description Logics
give a formal semantics to specialization

Subtyping is Contravariant (1/2)

- Subtyping is ruled by substitutability:
  any value of a subtype may be bound to an entity typed by a supertype
  without a run-time type error

Subtyping is Contravariant (1/2)

- Subtyping is ruled by substitutability:
  any value of a subtype may be bound to an entity typed by a supertype without a run-time type error

- Let a method $m_A(t) : t'$ defined in class A,
  redefined as $m_B(u) : u'$ in a subclass B, then

$$B <: A \Rightarrow t <: u \;\; \& \;\; u' <: t'$$

  this is contravariance rule:
  return type is covariant but parameter type is contravariant.

Subtyping is Contravariant (2/2)

- Attributes must be invariant:
  covariant as read method, contravariant as write method;

Subtyping is Contravariant (2/2)

- Attributes must be invariant:
  covariant as read method, contravariant as write method;

- Contravariance for classes is needed if one wants:

Subtyping is Contravariant (2/2)

- Attributes must be invariant:
  covariant as read method, contravariant as write method;

- Contravariance for classes is needed if one wants:
  - type safety

Subtyping is Contravariant (2/2)

- Attributes must be invariant:
  covariant as read method, contravariant as write method;

- Contravariance for classes is needed if one wants:
  - type safety
  - inheritance is subtyping: $B \prec A \Longrightarrow type(B) <: type(A)$

Subtyping is Contravariant (2/2)

- Attributes must be invariant:
  covariant as read method, contravariant as write method;

- Contravariance for classes is needed if one wants:
  - type safety
  - inheritance is subtyping: $B \prec A \Longrightarrow type(B) <: type(A)$

  Specialization also is substitutability

The conflict and its causes

- domains are covariant and ruled by an existential quantifier:
  there exists a value ... (e.g. a cow eats some grass)

The conflict and its causes

- domains are covariant and ruled by an existential quantifier:
  there exists a value ... (e.g. a cow eats some grass)

- types are contravariant and ruled by a universal quantifier:
  any value can be substituted ... (e.g. all cows eat all grass)

The conflict and its causes

- domains are covariant and ruled by an existential quantifier:
  there exists a value ... (e.g. a cow eats some grass)
- types are contravariant and ruled by a universal quantifier:
  any value can be substituted ... (e.g. all cows eat all grass)
- domains are not types but ...

The conflict and its causes

- domains are covariant and ruled by an existential quantifier:
  there exists a value ... (e.g. a cow eats some grass)

- types are contravariant and ruled by a universal quantifier:
  any value can be substituted ... (e.g. all cows eat all grass)

- domains are not types but ...

- ... types are a good approximate (upper bound)

Type errors are in “real world”

An example: the mad cow disease!

Type errors are in “real world”

An example: the mad cow disease!

analysis and design are left to the audience!

Programming Languages (1/3) : C++ & JAVA

- they are more than contravariant, invariant

  C++ is actually covariant for return type, but nobody knows it!

Programming Languages (1/3) : C++ & JAVA

- they are more than contravariant, invariant

  C++ is actually covariant for return type, but nobody knows it!

- they are not type safe, due to casting:

  if you want covariant parameters, use downcasts!

  in JAVA, don’t worry about genericity, use downcasts!

Programming Languages (1/3) : C++ & JAVA

- they are more than contravariant, invariant

  C++ is actually covariant for return type, but nobody knows it!

- they are not type safe, due to casting:

  if you want covariant parameters, use downcasts!

  in JAVA, don't worry about genericity, use downcasts!

- they can simulate type overriding with static overloading:

  if you are not too demanding regarding semantics:

  OO semantics lies in dynamic type whereas static overloading is ruled by static types

Programming Languages (2/3) : EIFFEL

- the only covariant language,

Programming Languages (2/3) : EIFFEL

- the only covariant language,
- but it pretends to type safety, with the catcall rule,
  polymorphic calls of a covariant method are forbidden

Programming Languages (2/3) : EIFFEL

- the only covariant language,
- but it pretends to type safety, with the catcall rule,
  polymorphic calls of a covariant method are forbidden
- covariant overriding would be possible but unusable!

Programming Languages (2/3) : EIFFEL

- the only covariant language,
- but it pretends to type safety, with the catcall rule,
  polymorphic calls of a covariant method are forbidden
- covariant overriding would be possible but unusable!
- fortunately, the rule does not work in separate compilation
  and it is not implemented in SMALL EIFFEL (global compilation)

Programming Languages (3/3)

They are mostly politically correct!

Programming Languages (3/3)

They are mostly politically correct!

- all pretends to be type safe, but they are not,

Programming Languages (3/3)

They are mostly politically correct!

- all pretends to be type safe, but they are not,
- in C++ and JAVA, a sound, complete, but clumsy, simulation of covariance is possible, using downcasts and static overloading

Programming Languages (3/3)

They are mostly politically correct!

- all pretends to be type safe, but they are not,
- in C++ and JAVA, a sound, complete, but clumsy, simulation of covariance is possible, using downcasts and static overloading
- it would be better to be covariant with explicit handling of type errors!

Modeling languages (1/2)

In which camp are modeling languages and UML?

Modeling languages (1/2)

In which camp are modeling languages and UML?

- Aristotle, KR and covariance?
- type theory and contravariance?
- JAVA and invariance?

Modeling languages (1/2)

In which camp are modeling languages and UML?

- Aristotle, KR and covariance?
- type theory and contravariance?
- JAVA and invariance?

Any idea ?

Modeling languages (1/2)

In which camp are modeling languages and UML?

- Aristotle, KR and covariance?
- type theory and contravariance?
- JAVA and invariance?

Any idea ? The answer is: JAVA's invariance!

Modeling languages (1/2)

In which camp are modeling languages and UML?

- Aristotle, KR and covariance?
- type theory and contravariance?
- JAVA and invariance?

Any idea ? The answer is: JAVA's invariance!

EIFFEL is not even quoted in UML v1.4

Modeling languages (2/2)

They should obviously be in the camp of “real world”!

The future of type theory

Many variations around subtyping and polymorphism:

- mytype and matching (i.e. mytype in contravariant position):
  safe in match-bounded genericity, LOOM's match-polymorphism and exact types
  unsafe as subtyping and in EIFFEL's anchored types

- SATHER's "classes are not types, inheritance is not subtyping":
  covariant specialization without polymorphism

- multiple selection with Castagna's multi-methods

The future of type theory

Many variations around subtyping and polymorphism:

- mytype and matching (i.e. mytype in contravariant position):
  safe in match-bounded genericity, LOOM's match-polymorphism and exact types
  unsafe as subtyping and in EIFFEL's anchored types
- SATHER's "classes are not types, inheritance is not subtyping":
  covariant specialization without polymorphism
- multiple selection with Castagna's multi-methods

Anyway type errors are in "real world"!

Real covariant languages (1/2)

Programming language:

- neither casting nor static overloading,
- a covariant type system, à la EIFFEL, maybe in a multi-method framework
- explicit handling of type errors: should be considered as soon as analysis
- syntactic means to impose invariance
- many other OO improvements on C++ and JAVA:
  self encapsulation à la SMALLTALK, class variables, ...

Real covariant languages (2/2)

Analysis is a customer for programming.

Modeling languages should impose their proper specifications instead of taking them from programming languages.

The end