Specifications via Realizability

Andrej Bauer
Department of Mathematics and Physics
University of Ljubljana, Slovenia

Christopher A. Stone
Computer Science Department
Harvey Mudd College, USA

CLASE (@ ETAPS), Edinburgh UK, April 2005

Motivation & Background

Computable and constructive mathematics deals with computable aspects of mathematics. We can extract programs from constructive proofs. This is often done in an ad-hoc manner:

Motivation & Background

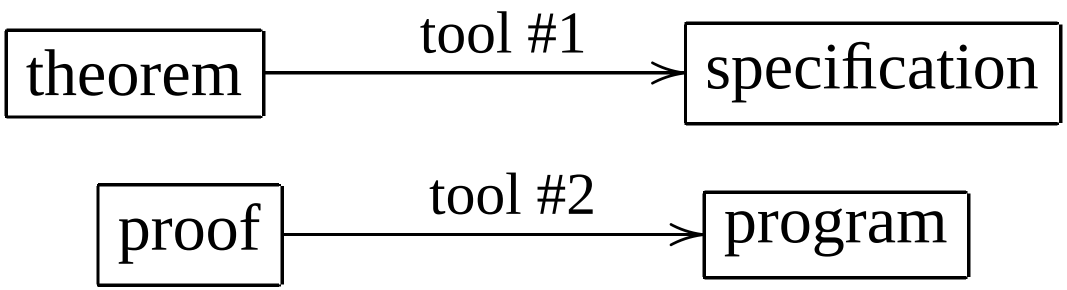
Computable and constructive mathematics deals with computable aspects of mathematics. We can extract programs from constructive proofs. This is often done in an ad-hoc manner:

The following would be better:

Motivation & Background

Computable and constructive mathematics deals with computable aspects of mathematics. We can extract programs from constructive proofs. This is often done in an ad-hoc manner:

The following would be better:



We are going to speak about tool #1 *only*.

Motivation & Background cont.

Why do we even need a tool for translation of theorems to specifications?

1. We want to express theorems directly in *full first-order logic* rather than a specification language.

Motivation & Background cont.

Why do we even need a tool for translation of theorems to specifications?

1. We want to express theorems directly in *full first-order logic* rather than a specification language.

2. It turns out that theorems and constructions of computable mathematics get *too complicated* for manual translation.

   Try writing down a specifi cation for the solution operator of ordinary linear differential equations on smooth manifolds.

Motivation & Background cont.

Why do we even need a tool for translation of theorems to specifications?

1. We want to express theorems directly in *full first-order logic* rather than a specification language.

2. It turns out that theorems and constructions of computable mathematics get *too complicated* for manual translation.

   Try writing down a specifi cation for the solution operator of ordinary linear differential equations on smooth manifolds.

Why didn't you extract programs from proofs?

1. We might have if we already had tool #1.

2. It is often easier to write a program than a formalized proof.

3. We are hoping others have done it already.

Overview

1. Theories & specifications
2. Realizability translation
3. Concluding remarks

Theories

We axiomatize mathematical structures in (constructive) first-order logic with (predicative) set theory.

- logic: $\wedge \implies \vee \; \exists \; \forall \; \top \; \bot \; =$.
- sets: $A \times B$, $A \to B$, $A + B$, $\{x : A \mid \phi(x)\}$, $A/\neg\neg\rho$.

This language is close to what is used in practice, except for missing dependent types.

A *theory* is a list of sets, predicates/relations, constants and axioms.

Example

```
theory DenseLinearOrder =
thy
  set s
  relation (<) : s * s
  implicit x, y, z : s

  axiom transitive x y z = (x < y and y < z) => x < z

  axiom assymetric x y   = not (x < y and y < x)

  axiom linear x y z     = (x < y) => (x < z or z < y)

  axiom dense x y        = x < y => some z.(x < z and z < y)
end
```

Capabilities *not* shown in previous example

- A theory may be *parameterized* by a model of another theory.

  E.g., the theory of vector spaces *over* a field F.

Capabilities *not* shown in previous example

- A theory may be *parameterized* by a model of another theory.

  E.g., the theory of vector spaces *over* a field F.

- An axiom may express a *universal property* by quantifying over all structures of a given kind.

  Finite lists over a set A are the *initial algebra* for the functor $X \mapsto A + X$.

Capabilities *not* shown in previous example

- A theory may be *parameterized* by a model of another theory.

  E.g., the theory of vector spaces *over* a field F.

- An axiom may express a *universal property* by quantifying over all structures of a given kind.

  Finite lists over a set A are the *initial algebra* for the functor $X \mapsto A + X$.

Thus our system allows theories and axioms to be parameterized by models of theories.

Specifi cations

- Specifications are ML signatures with assertions.
- Assertions are *negative* formulas:

$$\bot \quad \top \quad = \quad \wedge \quad \Longrightarrow \quad \forall$$

Specifi cations

- Specifications are ML signatures with assertions.
- Assertions are *negative* formulas:

$$\bot \quad \top \quad = \quad \wedge \quad \Longrightarrow \quad \forall$$

- The classical and constructive meanings of negative formulas coincide.

  Benefi t: programmers who are not familiar with constructive logic will understand such specifi cations.

Specifi cations

- Specifications are ML signatures with assertions.
- Assertions are *negative* formulas:

$$\bot \quad \top \quad = \quad \wedge \quad \Longrightarrow \quad \forall$$

- The classical and constructive meanings of negative formulas coincide.

  Benefi t: programmers who are not familiar with constructive logic will understand such specifi cations.

- Parameterized specifications are signatures for ML functors with assertions.

Overview

✓ Theories & specifications

☞ Realizability translation

3. Concluding remarks

Realizability translation

- We translate theories to specifications using the *realizability interpretation*, originally defined by S.C. Kleene.
- A common alternative is the *Curry-Howard isomorphism*, a.k.a. "propositions-as-types".

Realizability translation

- We translate theories to specifications using the *realizability interpretation*, originally defined by S.C. Kleene.
- A common alternative is the *Curry-Howard isomorphism*, a.k.a. "propositions-as-types".

These two are similar but *not* equivalent and in fact the Curry-Howard isomorphism is less suitable for our needs:

Realizability translation

- We translate theories to specifications using the *realizability interpretation*, originally defined by S.C. Kleene.
- A common alternative is the *Curry-Howard isomorphism*, a.k.a. “propositions-as-types”.

These two are similar but *not* equivalent and in fact the Curry-Howard isomorphism is less suitable for our needs:

- Not every programming language is “just λ-calculus”.

  Certain algorithms in computable analysis *require* programming features like exceptions, timeouts, and decompilation.

Realizability translation

- We translate theories to specifications using the *realizability interpretation*, originally defined by S.C. Kleene.
- A common alternative is the *Curry-Howard isomorphism*, a.k.a. "propositions-as-types".

These two are similar but *not* equivalent and in fact the Curry-Howard isomorphism is less suitable for our needs:

- Not every programming language is "just λ-calculus".

  Certain algorithms in computable analysis *require* programming features like exceptions, timeouts, and decompilation.

- In computable mathematics *partial* functions are unavoidable.

  One *cannot* make every function total by some trivial trick such as prescribing a default value outside of domain of defi nition.

Realizability interpretation

1. A set A is interpreted by an underlying type of realizers $|A|$ together with a partial equality predicate $=_A$ on $|A|$.
   - $t =_A s$ means "t and s realize (represent) the same element of A".
   - Also write $t \Vdash_A x$ to mean "t realizes $x \in A$".
   - Propositions-as-types: set $=$ type.

Realizability interpretation

1. A set A is interpreted by an underlying type of realizers $|A|$ together with a partial equality predicate $=_A$ on $|A|$.
   - $t =_A s$ means "t and s realize (represent) the same element of A".
   - Also write $t \Vdash_A x$ to mean "t realizes $x \in A$".
   - Propositions-as-types: set $=$ type.
2. To every predicate ϕ we assign a type $|\phi|$ and specify when a term of type $|\phi|$ *realizes* ϕ.
   - We write $t \Vdash \phi$ when t realizes ϕ.
   - Some terms of type $|\phi|$ may not be valid realizers, e.g., because they diverge.
   - Propositions-as-types: proof $=$ program.

Realizability interpretation cont.

Consider a subset $S = \{x : A \bigm| \phi(x)\}$:

$$|S| = |A| \times |\phi|$$

$$(t_1, t_2) \Vdash_S \iota_S(x) \quad \text{iff} \quad t_1 \Vdash_A x \text{ and } t_2 \Vdash \phi(x)$$

Implication:

$$|\phi \implies \psi| = |\phi| \to |\psi|$$

$$t \Vdash \phi \implies \psi \quad \text{iff} \quad \text{for all } u \in |\phi|, \text{ if } u \Vdash \phi \text{ then } t\,u \Vdash \psi$$

Existential quantifi er:

$$|\exists x \in A.\, \phi(x)| = |\phi| \times |\psi|$$

$$(t_1, t_2) \Vdash \exists x \in A.\phi(x) \quad \text{iff} \quad t_1 \Vdash_A x \text{ and } t_2 \Vdash \phi(x)$$

The translation procedure

Sets are translated to the corresponding datatypes.

For translation of propositions, we use:

Theorem:
In realizability interpretation, every ϕ *is equivalent to* $\exists r \in |\phi|.\, \phi'(r)$,
where $\phi'(r)$ *is a negative formula.*

Intuitive meaning:
r is the *computational content* of ϕ and $\phi'(r)$ says "r realizes ϕ".

The translation procedure

Sets are translated to the corresponding datatypes.

For translation of propositions, we use:

Theorem:

In realizability interpretation, every ϕ *is equivalent to* $\exists r \in |\phi|.\, \phi'(r)$, *where* $\phi'(r)$ *is a negative formula.*

Intuitive meaning:

r is the *computational content* of ϕ and $\phi'(r)$ says "r realizes ϕ".

A theorem ϕ is translated to the specification

`val` $r : |\phi|$

`(* Assertion` $\phi'(r)$ `*)`

Overview

- ✓ Theories & signatures
- ✓ Realizability translation
- ☞ Concluding remarks

Related Work

- Realizability:

  Kleene, Troelstra, Hyland, van Oosten, Longley, . . .

- Constructive and computable mathematics:

  Bishop & Bridges, Markov, Pour El & Richard, Ko, Weihrauch, Schröder, Hertling, Brattka, Scott, Edalat, . . .

- Extraction of signatures and programs:
  - Schwichtenberg, Hayashi, Constable, Coquand, Huet, . . .
  - Poernomo, Crossley & Wirsing 2002 (extraction of SML structures and programs)
  - Cruz-Filipe & Spitters (extraction from Fundamental theorem of algebra)

Contributions

We provide a tool, RZ, for automated translation of mathematical theories to specifications.

- RZ should hopefully prove useful in bringing constructive mathematics closer to programmers.
- RZ should hopefully be a good source of interesting specifications.
- RZ demonstrates how the realizability interpretation can be used as an alternative to the Curry-Howard isomorphism.

Future Work

- Experiment with non-trivial theories.

  Real numbers, differentiable functions, Banach and Hilbert spaces, (weak) set theories, . . .

- Implement dependent types.

  Note: under realizability interpretation the dependent types translate to *simple* types, so we do not need a programming language with dependent types.

- Hook up RZ with a program extraction tool.