Representing Streams in Second Order Logic (Coinduction and Coalgebra in Second Order Logic)

Herman Geuvers

Radboud University Nijmegen
and
Eindhoven University of Technology
The Netherlands

Seminar “Representing Streams”,
Lorentz Centre, Leiden Dec 14 2012

Representing Streams

- Concrete approaches
- Abstract approach . . . category theory, coalgebra
- Now: abstract approach using second order logic.

Representing Streams

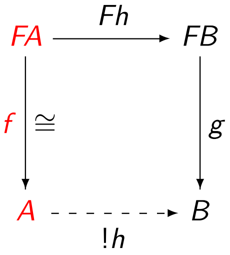
- Concrete approaches
- Abstract approach . . . category theory, coalgebra
- Now: abstract approach using second order logic.

Contents:

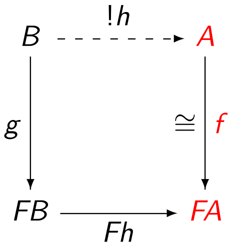
- Algebra and coalgebra, induction and coinduction
- Defining algebras and coalgebras in Second order logic
- Extracting (correct) programs from proofs
- Bisimulation is the natural equality on coalgebras

Algebras and Coalgebras

Initial F-algebra: (A, f) s.t. $\forall (B, g)$, $\exists! h$ s.t. the diagram commutes:

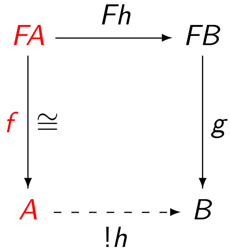


Final F-coalgebra: (A, f) s.t. $\forall (B, g)$, $\exists! h$ s.t. the diagram commutes:



Algebras

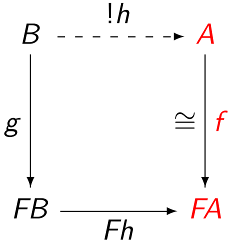
Initial F-algebra:



- constructor function, f to A, is basic
- function definition principle: recursion, to define a function h on A.
- proof principle: induction, proving $\forall x \in A \ldots$

Coalgebras

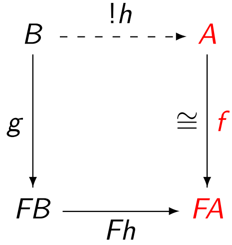
Final F-coalgebra:



- destructor function, f on A, is basic
- function definition principle: corecursion, to define a function h to A.
- proof principle: coinduction, proving ??

Coalgebras

Final F-coalgebra:



- destructor function, f on A, is basic
- function definition principle: corecursion, to define a function h to A.
- proof principle: coinduction, proving ??
  the basic proof principle for coalgebras in bisimulation ...
  Can we reconcile these two dual phenomena?

Second Order Logic

- First order language of terms, with countably many constants and functions symbols.
- Quantification over predicates, relations, ...
  $\forall P \forall x (P(x) \rightarrow P(x))$, $\exists R \forall x R(x, x)$.

Second Order Logic

- First order language of terms, with countably many constants and functions symbols.
- Quantification over predicates, relations, ...
  $\forall P \forall x (P(x) \rightarrow P(x))$, $\exists R \forall x R(x, x)$.

Deduction rules as expected

$$\frac{\Gamma \vdash \forall P \varphi}{\Gamma \vdash \varphi[P := \psi(x)]} \quad \frac{\Gamma \vdash \varphi}{\Gamma \vdash \forall P \varphi} \; P \text{ not free in } \Gamma$$

Natural numbers

We can recover our inductive types as definable predicates in SOL. For Natural numbers, let 0 be a constant symbol and s a unary function symbol. Define

$$N(x) := \forall P(P(0) \land \forall y P(y) \to P(s(y))) \to P(x)$$

We can now prove the following.

$$N(0), \tag{1}$$

$$\forall y N(y) \to N(s(y)). \tag{2}$$

Natural numbers

We can recover our inductive types as definable predicates in SOL. For Natural numbers, let 0 be a constant symbol and s a unary function symbol. Define

$$N(x) := \forall P(P(0) \wedge \forall y P(y) \rightarrow P(s(y))) \rightarrow P(x)$$

We can now prove the following.

$$N(0), \tag{1}$$
$$\forall y N(y) \rightarrow N(s(y)). \tag{2}$$

Krivine, Parigot, Leivant: one can actualy program with these types, using the Curry-Howard formulas-as-types (and proofs-as-terms) embedding.

Natural numbers

We can recover our inductive types as definable predicates in SOL. For Natural numbers, let 0 be a constant symbol and s a unary function symbol. Define

$$N(x) := \forall P(P(0) \wedge \forall y P(y) \to P(s(y))) \to P(x)$$

We can now prove the following.

$$N(0), \tag{1}$$
$$\forall y N(y) \to N(s(y)). \tag{2}$$

Krivine, Parigot, Leivant: one can actualy program with these types, using the Curry-Howard formulas-as-types (and proofs-as-terms) embedding.
The proof of (1) is Zero, the proof of (2) is the Successor.

Intended Model

- Krivine, Parigot: our intended model is some untyped Turing complete functional language, like untyped λ-calculus or combinatory logic.
- Krivine: AF2 (second order logic with proof objects, interpreted as programs)
- Parigot: “ProPre” system, “Programmation avec de Preuves”

Intended Model

- Krivine, Parigot: our intended model is some untyped Turing complete functional language, like untyped λ-calculus or combinatory logic.
- Krivine: AF2 (second order logic with proof objects, interpreted as programs)
- Parigot: "ProPre" system, "Programmation avec de Preuves"

We can define new correct algorithms by specifying them:
Example

$$\begin{aligned} \mathrm{plus}(x, 0) &= x \\ \mathrm{plus}(x, s(y)) &= s(\mathrm{plus}(x, y)) \end{aligned}$$

Now we prove

$$\forall xy N(x) \wedge N(y) \rightarrow N(\mathrm{plus}(x, y))$$

This proof, interpreted as a λ-term, is a correct implementation of the Plus function in our computational model.

Proving programs correct

Theorem Krivine
Given a set of equations E specifying the binary function f on naturals, and a proof p of

$$\forall x, y(N(x) \wedge N(y) \rightarrow N(f(x, y)),$$

the interpretation of p as an untyped λ-term, $\overline{p}$ is a program that computes f (i.e. $\overline{p}$ satisfies E).
[This works for all inductice data-types (lists, trees, . . .)]

Streams

This can also be done coalgebracially.
Let A be some unary predicate, h and t unary function symbols.
Define the unary predicate Str:

$$\mathrm{Str}(x) := \exists P, \forall y, P(y) \to A(h(y)) \wedge P(t(y)) \wedge P(x).$$

We can now prove the following.

$$\forall y, \mathrm{Str}(y) \to A(h(y)), \tag{3}$$

$$\forall y, \mathrm{Str}(y) \to \mathrm{Str}(t(y)). \tag{4}$$

Under the Curry-Howard embedding, the proof of (3) is the Head function and the proof of (4) is the Tail function.

Proving stream programs correct

The Krivine method can be extended to coinductive data types.
Example for streams.

$$\begin{aligned} h(f(x,y)) &= x \\ t(f(x,y)) &= y \end{aligned}$$

We can prove

$$\forall x, y(A(x) \wedge \mathrm{Str}(y) \to \mathrm{Str}(f(x,y)))$$

and this proof 'is' a program that implements the 'cons' function on streams.

The more general picture

Definition For P and Q predicates: $P \subseteq Q$ iff $\forall x(P(x) \rightarrow Q(x))$.
Definition A predicate scheme $\Phi(P)$ is monotone iff

$$\forall P, Q(P \subseteq Q \rightarrow \Phi(P) \subseteq \Phi(Q)).$$

[A predicate scheme is just a formula with a specific open place for a unary predicate.]

Inductive and coinductive predicates in SOL

Let $\Phi(P)$ be a monotone predicate scheme.
Then Φ has a **least fixed point** and a **greatest fixed point**.

Inductive and coinductive predicates in SOL

Let $\Phi(P)$ be a monotone predicate scheme.
Then Φ has a **least fixed point** and a **greatest fixed point**.
Definition

$$\begin{aligned} \mathrm{lfp}(\Phi)(x) &:= \forall P, \Phi(P) \subseteq P \to P(x), \\ \mathrm{gfp}(\Phi)(x) &:= \exists P, P \subseteq \Phi(P) \wedge P(x). \end{aligned}$$

Inductive and coinductive predicates in SOL

Let $\Phi(P)$ be a monotone predicate scheme.
Then Φ has a **least fixed point** and a **greatest fixed point**.
Definition

$$\begin{aligned} \mathrm{lfp}(\Phi)(x) &:= \forall P, \Phi(P) \subseteq P \to P(x), \\ \mathrm{gfp}(\Phi)(x) &:= \exists P, P \subseteq \Phi(P) \wedge P(x). \end{aligned}$$

Lemma

- $\forall P, \Phi(P) \subseteq P \to \mathrm{lfp}(\Phi) \subseteq P$.
- $\forall x, \Phi(\mathrm{lfp}(\Phi))(x) \leftrightarrow \mathrm{lfp}(\Phi)(x)$.
- $\forall P, P \subseteq \Phi(P) \to P \subseteq \mathrm{gfp}(\Phi)$.
- $\forall x, \Phi(\mathrm{gfp}(\Phi))(x) \leftrightarrow \mathrm{gfp}(\Phi)(x)$.

Category theory in SOL

Which predicate schemes are monotone?

Definition The following polynomial functors all give rise to a monotone predicate scheme

$$F(X) ::= X \mid A \mid F_1(P) + F_2(P) \mid F_1(P) \times F_2(P)$$

where A is a constant; this includes U, the unit object.
This means that we can define $F(P)$ as a monotone predicate. For example

$$\begin{aligned} U(x) &:= x = u \\ F_1 + F_2\,(P)(x) &:= (x = \mathrm{in}_1(y) \wedge F_1(P)(y)) \vee (x = \mathrm{in}_2(y) \wedge F_2(P)(y)) \\ F_1 \times F_2\,(P)(x) &:= x = F_1(P)(\pi_1(x)) \wedge F_2(P)(\pi_2(y)) \end{aligned}$$

Back to the naturals and the streams

The naturals are the initial $F(X) = 1 + X$ algebra
The streams over A are the final $G(X) = A \times X$ coalgebra

Back to the naturals and the streams

The naturals are the initial $F(X) = 1 + X$ algebra
The streams over A are the final $G(X) = A \times X$ coalgebra
We define $F(P)(x)$ and $G(P)(x)$ and notice that we have 0, s, h and t such that for P a predicate variable,

$$\begin{aligned} F(P)(x) &= x = 0 \vee \exists y (P(y) \wedge x = s(y)), \\ G(P)(x) &= A(h(x)) \wedge P(t(x)). \end{aligned}$$

Note that these predicate schemes can be viewed as 'rule sets' as follows

$$\frac{}{F(P)(0)} \quad \frac{P(y)}{F(P)(s(y))} \quad \frac{A(h(x)) \quad P(t(x))}{G(P)(x)}$$

Properties

Lemma

$$\begin{aligned} N(x) &\leftrightarrow \forall P(F(P) \subseteq P \rightarrow P(x)), \\ \mathrm{Str}(x) &\leftrightarrow \exists P((P \subseteq G(P)) \wedge P(x)). \end{aligned}$$

In fact: N is the **smallest** set closed under F and Str is the **largest** set closed under G.

Lemma

- $\forall P(F(P) \subseteq P \rightarrow N \subseteq P)$.
- $\forall x(F(N)(x) \leftrightarrow N(x))$.
- $\forall P(P \subseteq G(P) \rightarrow P \subseteq \mathrm{Str})$.
- $\forall x(G(\mathrm{Str})(x) \leftrightarrow \mathrm{Str}(x))$.

Generalizing to relations

Polynomial functors T also 'give rise to' a monotone **relation scheme** T_r.
Example of the naturals,
$F(P)(x) := x = 0 \vee \exists y(P(y) \wedge x = s(y))$. Let R be a binary relation variable.

$$\begin{aligned} F_r(R)(x, y) \quad &:= \quad x = y = 0 \vee \\ &\qquad \exists p, q(R(p, q) \wedge x = s(p) \wedge y = s(q)). \end{aligned}$$

$$N_r(y_1, y_2) := \forall R(F_r(R) \subseteq R) \rightarrow R(y_1, y_2)$$

Generalizing to relations

Polynomial functors T also 'give rise to' a monotone **relation scheme** T_r.
Example of the naturals,
$F(P)(x) := x = 0 \vee \exists y(P(y) \wedge x = s(y))$. Let R be a binary relation variable.

$$F_r(R)(x,y) \quad := \quad x = y = 0 \vee \exists p, q(R(p,q) \wedge x = s(p) \wedge y = s(q)).$$

$$N_r(y_1, y_2) := \forall R(F_r(R) \subseteq R) \rightarrow R(y_1, y_2)$$

This canonical relation on N is equivalent to

$$\forall R(\; R(0,0) \wedge \forall x_1 x_2(R(x_1, x_2) \rightarrow R(s(x_1), s(x_2)))$$

$$\rightarrow R(y_1, y_2)\;).$$

Generalizing to relations

Example of the streams, $G(P)(x) := A(h(x)) \wedge P(t(x))$. Let R be a binary relation variable.

$$G_r(R)(x,y) \quad := \quad (h(x) =_A h(y)) \wedge R(t(x), t(y)).$$

$$\mathrm{Str}_r(y_1, y_2) := \exists R(R \subseteq G_r(R) \wedge R(y_1, y_2))$$

Generalizing to relations

Example of the streams, $G(P)(x) := A(h(x)) \wedge P(t(x))$. Let R be a binary relation variable.

$$G_r(R)(x, y) \quad := \quad (h(x) =_A h(y)) \wedge R(t(x), t(y)).$$

$$\mathrm{Str}_r(y_1, y_2) := \exists R(R \subseteq G_r(R) \wedge R(y_1, y_2))$$

Note: The relation Str_r is **bisimulation equivalence** on streams over A.

Generalizing to relations

Example of the streams, $G(P)(x) := A(h(x)) \wedge P(t(x))$. Let R be a binary relation variable.

$$G_r(R)(x, y) \quad := \quad (h(x) =_A h(y)) \wedge R(t(x), t(y)).$$

$$\mathrm{Str}_r(y_1, y_2) := \exists R(R \subseteq G_r(R) \wedge R(y_1, y_2))$$

Note: The relation Str_r is **bisimulation equivalence** on streams over A.
Note:

$$\mathrm{Str}_r(x, y) \Leftrightarrow \forall n \in \mathbf{N} h(t^n(x)) =_A h(t^n(y))$$

The latter can be defined (inductively) in SOL.

Conclusions, Further work

- Bisimulation is the natural equality on streams
- This approach naturally generates a definition of equality on other coalgebras.
- Characterize classes of streams in SOL?