Coinduction and rational trees

Régis Spadotti

I.R.I.T

November 18^{th}, 2013

Outline

Motivating example

Abstract Syntax

- Consider an abstract syntax representing a *very* simple process algebra (CSP-like):

```
Proc ::= STOP          -- deadlock
       | A ⇒ Proc      -- communication
       | Proc + Proc   -- choice
```

Type definition

- This abstract syntax T is naturally inductively defined in proof assistants

```
data Proc : Set where
  STOP  : Proc
  _⇒_   : A → Proc → Proc
  _+_   : Proc → Proc → Proc
```

- We can compute asynchronous parallel composition of processes

```
_|||_ : Proc → Proc → Proc
STOP    ||| _       = STOP
_       ||| STOP    = STOP
(P + Q) ||| R       = (P ||| R) + (Q ||| R)
R       ||| (P + Q) = (R ||| P) + (R ||| Q)
(a ⇒ P) ||| (b ⇒ Q) = a ⇒ (P ||| b ⇒ Q) + b ⇒ (a ⇒ P ||| Q)
```

Motivating example

- Our syntax models only **finite processes** which eventually terminate
- Inductive reasoning works quite well in this context

Extension

- Say we want to extend our syntax to model **infinite processes** that may never terminate
- New syntax is given by

$$
\begin{array}{lll}
\text{Proc} ::= & \text{STOP} & \text{-- deadlock} \\
\mid & \text{A} \Rightarrow \text{Proc} & \text{-- action} \\
\mid & \text{Proc} + \text{Proc} & \text{-- choice} \\
\mid & \mathbf{X} & \text{-- var} \\
\mid & \mathbf{rec\ X.\ Proc} & \text{-- recursive process}
\end{array}
$$

- Semantically, $\mathbf{rec}\ \text{X}.\ \text{a} \Rightarrow \text{X}$ represents the infinite stream $\text{a} \Rightarrow \text{a} \Rightarrow \cdots$
- $[\![\ \mathbf{rec}\ \text{X}.\ \text{P}\ ]\!] \approx [\![\ \text{P}\ [\ \text{X} := \mathbf{rec}\ \text{X}.\ \text{P}\ ]\ ]\!]$

Motivating example

Problem

The natural induction principle derived from this abstract syntax definition does not capture the semantics of **rec** .

Motivating example

Problem

The natural induction principle derived from this abstract syntax definition does not capture the semantics of **rec** .

Parallel composition revisited

$$
\begin{array}{lll}
\_|||\_ : \mathsf{Proc} \to \mathsf{Proc} \to \mathsf{Proc} & & \\
\ldots \quad ||| \ldots & = & \ldots \\
\textbf{rec}\ \mathsf{X}.\ \mathsf{P} \ ||| \ \mathsf{Q} & = & ? \\
\mathsf{Q} \quad ||| \ \textbf{rec}\ \mathsf{X}.\ \mathsf{P} & = & ?
\end{array}
$$

Motivating example

Problem

The natural induction principle derived from this abstract syntax definition does not capture the semantics of **rec** .

Parallel composition revisited

$$
\begin{array}{lcll}
\_ ||| \_ & : & \mathsf{Proc} \to \mathsf{Proc} \to \mathsf{Proc} & \\
\ldots & ||| \ldots & = & \ldots \\
\textbf{rec}\ \mathsf{X}.\ \mathsf{P} & ||| \ \mathsf{Q} & = & \mathsf{P}\ [\ \mathsf{X} := \textbf{rec}\ \mathsf{X}.\ \mathsf{P}\ ]\ ||| \ \mathsf{Q} \\
\mathsf{Q} & ||| \ \textbf{rec}\ \mathsf{X}.\ \mathsf{P} & = & \mathsf{Q} ||| \mathsf{P}\ [\ \mathsf{X} := \textbf{rec}\ \mathsf{X}.\ \mathsf{P}\ ]
\end{array}
$$

$\mathsf{P}\ [\ \mathsf{X} := \textbf{rec}\ \mathsf{X}.\ \mathsf{P}\ ]$ is not a subterm of $\textbf{rec}\ \mathsf{X}.\ \mathsf{P}$

Motivating example

Problem

The natural induction principle derived from this abstract syntax definition does not capture the semantics of **rec** .

Parallel composition revisited

$$
\begin{array}{lcl}
\_ \,|||\, \_ & : & \mathsf{Proc} \to \mathsf{Proc} \to \mathsf{Proc} \\
\ldots \,|||\, \ldots & = & \ldots \\
\mathbf{rec}\ \mathsf{X}.\ \mathsf{P} \,|||\, \mathsf{Q} & = & \mathsf{P}\ [\ \mathsf{X} := \mathbf{rec}\ \mathsf{X}.\ \mathsf{P}\ ] \,|||\, \mathsf{Q} \\
\mathsf{Q} \,|||\, \mathbf{rec}\ \mathsf{X}.\ \mathsf{P} & = & \mathsf{Q} \,|||\, \mathsf{P}\ [\ \mathsf{X} := \mathbf{rec}\ \mathsf{X}.\ \mathsf{P}\ ]
\end{array}
$$

$\mathsf{P}\ [\ \mathsf{X} := \mathbf{rec}\ \mathsf{X}.\ \mathsf{P}\ ]$ is not a subterm of $\mathbf{rec}\ \mathsf{X}.\ \mathsf{P}$

Solutions

- Well-founded induction
- Coinduction

Motivating example

Coinductive representation

$$
\begin{array}{l}
\textbf{codata}\ \mathit{Proc}^{\omega}\ :\ \mathrm{Set}\ \textbf{where} \\
\quad \mathrm{STOP}\ :\ \mathit{Proc}^{\omega} \\
\quad \_\Rightarrow\_\ :\ \mathrm{A} \to \mathit{Proc}^{\omega} \to \mathit{Proc}^{\omega} \\
\quad \_+\_\ :\ \mathit{Proc}^{\omega} \to \mathit{Proc}^{\omega} \to \mathit{Proc}^{\omega}
\end{array}
$$

- Representation close to the finite one
- We can define the semantics of processes as $[\![\_]\!]\ :\ \mathrm{Proc{+}rec} \to \mathit{Proc}^{\omega}$

Parallel composition revisited (again !)

$$
\begin{array}{lcl}
\_|||\_\ :\ \mathit{Proc}^{\omega} \to \mathit{Proc}^{\omega} \to \mathit{Proc}^{\omega} & & \\
\mathrm{STOP} \ ||| \ \_ & = & \mathrm{STOP} \\
\_ \ ||| \ \mathrm{STOP} & = & \mathrm{STOP} \\
(\mathrm{P} + \mathrm{Q}) \ ||| \ \mathrm{R} & = & (\mathrm{P} \ ||| \ \mathrm{R}) + (\mathrm{Q} \ ||| \ \mathrm{R}) \\
\mathrm{R} \ ||| \ (\mathrm{P} + \mathrm{Q}) & = & (\mathrm{R} \ ||| \ \mathrm{P}) + (\mathrm{R} \ ||| \ \mathrm{Q}) \\
(\mathrm{a} \Rightarrow \mathrm{P}) \ ||| \ (\mathrm{b} \Rightarrow \mathrm{Q}) & = & \mathrm{a} \Rightarrow (\mathrm{P} \ ||| \ \mathrm{b} \Rightarrow \mathrm{Q}) + \mathrm{b} \Rightarrow (\mathrm{a} \Rightarrow \mathrm{P} \ ||| \ \mathrm{Q})
\end{array}
$$

Same definition as in the finite case

Motivating example

Question

Can we define function $f^{sem} : \star \longrightarrow \omega Proc$ on the semantics and *derive* a function $f^{syn} : \star \longrightarrow Proc - rec$ defined on the syntax such that $[\![f^{syn}]\!] \triangleq f^{sem}$?

Issue

- There are terms $(t : Proc^{\omega})$ which are not representable with the inductive definition
  - The process $(1 \Rightarrow 2 \Rightarrow 3 \Rightarrow \cdots)$ is not representable with **rec**
- Analogy with $\mathbb{Q}$ vs $\mathbb{R}$

Need to narrow $Proc^{\omega}$ to rational terms

Outline

Rational terms

Let's generalize the syntax.

Definition

A *signature* is defined as a dependent record

```
record Signature : Set₁ where
  constructor _,_
  field
    Label : Set            -- set of symbols
    |_|   : Label → ℕ      -- arity function
```

Extension

Given a signature we define its *extension* as an endofunctor

```
⟨_⟩ : Signature → Set → Set
⟨ S ⟩ X = Σ l. Vec X |l|
```

Abbott, Michael and al. (2003) Categories of Containers

Combinators

We can define various combinators to compose signatures such as :

$$\begin{array}{l} \mathsf{id} \quad : \mathsf{Signature} \\ \mathsf{const} \ : \mathsf{Set} \to \mathsf{Signature} \end{array}$$

$$\_ \uplus \_ \quad \_ \times \_ \ : \mathsf{Signature} \to \mathsf{Signature} \to \mathsf{Signature}$$

$\ldots$

Initial algebra and final coalgebra

data μ (C : Container) : Set **where**
  [_] : $\langle$ C $\rangle$ (μ C) $\to$ μ C

codata ν (C : Container) : Set **where**
  [_] : $\langle$ C $\rangle$ (ν C) $\to$ ν C

Rational terms

Example

If we consider the process algebra defined previously the set of symbols is given by

```
data ProcLabel : Set where
  STOP : ProcLabel
  _⇒   : A → ProcLabel
  _+_  : ProcLabel
```

and the arity function is given by

```
|_| : ProcLabel → ℕ
|STOP| = 0
|_⇒|  = 1
|_+_|  = 2
```

Using combinators

We could also use combinators to obtain the signature

```
ProcSig : Signature
ProcSig = const ⊤ ⊎ const A × id ⊎ id × id
```

With the *same* signature we get both the inductive and coinductive representation

$$\begin{aligned} \mathsf{Proc} &\cong \mu\ \mathsf{ProcSig} \\ \mathit{Proc}^{\omega} &\cong \nu\ \mathsf{ProcSig} \end{aligned}$$

Equality over coinductive terms

Bisimulation $\_ \approx \_ : \forall \{S\} \to \nu\, S \to \nu\, S \to \text{Set}$ is defined coinductively as

$$\frac{\forall\, i.\ v \cdot i \approx v' \cdot i}{[\ s\ ,\ v\ ] \approx [\ s\ ,\ v'\ ]}$$

Definition

The *subterm relation* $\_ \preccurlyeq \_ : \forall \{S\} \to \nu\, S \to \nu\, S \to \text{Set}$ is defined inductively as

$$\frac{}{t \preccurlyeq t}\ \preccurlyeq\text{-refl} \qquad \frac{\exists\, i.\ t_1 \preccurlyeq v \cdot i}{t_1 \preccurlyeq [\ s\ ,\ v\ ]}\ \preccurlyeq\text{-sub}$$

Rational term

A term $(t : \nu\, S)$ is *rational* if it has finately many subterms (w.r.t $\_ \approx \_$).

We call $\mathcal{R}\, \Sigma$ the restriction of $\nu\, \Sigma$ to rational terms.

Outline

Rational terms define a subset of coinductive terms ($\mathcal{R}\ \mathsf{S} \subseteq \nu\ \mathsf{S}$)

Question

Let $\phi\ :\ \nu\ \mathsf{S}_1 \rightarrow \nu\ \mathsf{S}_2$.

Does $\phi\ \mathsf{t} \in \mathcal{R}\ \mathsf{S}_2$ when $\mathsf{t} \in \mathcal{R}\ \mathsf{S}_1$?

We study various tree transformations which preserve rationality

Expressing transformations

- Rewriting rule

$$\sigma(x_1, \ldots, x_n) \mapsto \sigma'(y_1, \ldots, y_m)$$

Tree transducers

A tree transducer is defined as tuple $\langle\ Q\ ,\ q_0\ ,\ \Sigma\ ,\ \Delta\ ,\ R\ \rangle$ where

- Q is a finite set of states
- $q_0 \in Q$ is an initial state
- Σ and Δ are an input and output signature respectively.
- R is a finite set of complete and deterministic rewriting rules

We write $(\tau\ :\ \Sigma \Rightarrow \Delta)$ to make explicit the input/output signatures.

Semantics

The semantics of a tree transducer τ is given by the functions

$$\begin{array}{ll} [\![\ \tau\ ]\!] & :\ \mu\,\Sigma \to \mu\,\Delta \\ [\![\ \tau\ ]\!]^{\omega} & :\ \nu\,\Sigma \to \nu\,\Delta \end{array}$$

Engelfriet, J. (1977) Top-down tree transducers with regular look-ahead

Specifying rewriting rules[a]

$$\begin{aligned}
&\mathsf{Trans} : \mathsf{Signature} \to \mathsf{Set} \to \mathsf{Signature} \to \mathsf{Set}\ _ \\
&\mathsf{Trans}\ \Sigma\ \mathsf{Q}\ \Delta = \forall\ \{\alpha\} \to \mathsf{Q} \to \langle \Sigma \rangle\ \alpha \to \langle \Delta \rangle^*\ (\mathsf{Q} \times \alpha)
\end{aligned}$$

$$\begin{aligned}
&\textbf{where}\ \langle _ \rangle^* : \mathsf{Signature} \to \mathsf{Set} \to \mathsf{Set} \quad \text{-- free monad} \\
&\qquad \langle \mathsf{S} \rangle^*\ \mathsf{X} = \mu\ (\mathsf{const}\ \mathsf{X} \uplus \mathsf{S})
\end{aligned}$$

Example

Compute the length of a list

$$\begin{aligned}
&\mathsf{TransLength} : \forall\ \{\mathsf{A}\} \to \mathsf{Trans}\ (\mathsf{ListSig}\ \mathsf{A})\ \top\ \mathbb{N}\mathsf{Sig} \\
&\mathsf{TransLength}\ \mathsf{len}\ [] = \mathsf{zero} \\
&\mathsf{TransLength}\ \mathsf{len}\ (\mathsf{x} :: \mathsf{xs}) = \mathsf{suc}\ (\mathsf{len} \cdot \mathsf{xs})
\end{aligned}$$

Theorem

Let $\tau : \Sigma \Rightarrow \Delta$ be a finite state tree transducer.

$$\forall (\mathsf{t} : \nu\ \Sigma).\ \mathsf{t} \in \mathcal{R}\ \Sigma \to [\![\ \tau\ ]\!]^{\omega}\ \mathsf{t} \in \mathcal{R}\ \Delta$$

Bahr, Patrick and Day, Laurence E. (2013) Programming Macro Tree Transducers

Product of terms

The previous approach can be applied to compute product of terms

$$[\![\tau]\!]^{\omega} \ : \ \nu\, \Sigma \to \nu\, \Delta \to \nu\, \Gamma$$

As a result, it is possible to define parallel composition of processes and prove that it does preverse rationality.

Summary

A rational term consist of

- an (in)finite term
- a proof that its underlying structure is finite (i.e finitely many subterms)

Tree transducers may be used as

- a tool to define function that computes rational terms by construction
- as a proof method to prove that an arbitrary function preserves rationality

From inductive terms to rational terms

- We defined a semantic function $[\![\_]\!] : \text{Proc+rec} \rightarrow Proc^{\omega}$.
- We can prove that :

$$\forall\, t.\ [\![\, t\,]\!] \in \mathcal{R}$$

From rational terms to inductive terms

Conversely, we can define a function $[\![\_]\!]^{-1} : \mathcal{R} \rightarrow \text{Proc+rec}$
$\Rightarrow$ by induction on the set of subterms

Outline

Even more expressive rewriting rules

- ϵ-rules (production)

$$\text{Trans-}\epsilon\ \Sigma\ Q\ \Delta\ =\ \forall\ \{\alpha\} \to Q \to \alpha \to \langle\Delta\rangle^*\ (Q \times \alpha)$$

- Deeper context

$$\text{Trans n}\ \Sigma\ Q\ \Delta\ =\ \forall\ \{\alpha\} \to Q \to \langle\Sigma\rangle^{n}\ \alpha \to \langle\Delta\rangle^*\ (Q \times \alpha)$$

Decidability on rational terms

- Quantifiers on terms
  - All P t: P holds on each subterm of t
  - Any P t: P holds on one subterm of t
- If the predicate P is decidable then All P t and Any P t are decidable provided t is rational.

Questions ?