Bisimulations for Algebraic Effects

Some Preliminary Results

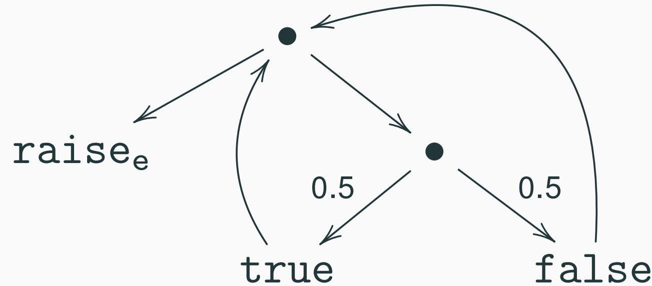
Francesco Gavazzo
University of Bologna & INRIA

Joint work with U. Dal Lago and R. Tanaka

Introduction

What is this talk about?

$\texttt{coinGame} = (\texttt{raise}_e \oplus (\texttt{true} \oplus_{0.5} \texttt{false}))::\texttt{coinGame}$



$\texttt{coinGame1} = ((\texttt{raise}_e \oplus \texttt{true}) \oplus_{0.5} (\texttt{false} \oplus \texttt{raise}_e))::\texttt{coinGame1}$

Question

What is the relationship between `coinGame` and `coinGame1`?

What is this talk about?

simpleGame = λn,score ->
 (simpleGame 1 0 $\oplus$ score) $\oplus_{\frac{1}{2n}}$
 (simpleGame n+1 score+1 $\oplus_{0.5}$ simpleGame n+1 score)

game = λn,score ->
 ((game 1 0 $\oplus_{\frac{2}{n+1}}$ game n+1 score+1) $\oplus_{\frac{n+1}{2n}}$ game n+1 score) $\oplus$
 (score $\oplus_{\frac{1}{n}}$ (game n+1 score+1 $\oplus_{0.5}$ game n+1 score))

Question

What is the relationship between game and simpleGame?

What is this talk about?

```
spine = λn -> n ⊕ spine (n+1)
nonDetNat = spine 0
```

```
nats = λn -> n::nats (n+1)
nonDetNat1 = foldr ⊕ _ (nats 0) -- nats 0 infinite list
```

```
doubleSpine = λn -> (n ⊕ n) ⊕ doubleSpine (n+1)
doubleNonDetNat = doubleSpine 0
```

Moral

Reasoning about programs with side effects is hard.

What is this talk about?

- Monads are used to deal with side effects in a functional way.
- However, due to their generality, monads come equipped with few functionalities (`join, return, >>=...`)
  $\leadsto$ monadic programming is not easy, and reasoning about monadic programs is even harder.
- To make programming easier, one often defines specific operations on monads (small domain specific language for families of side effects).
- G.D. Plotkin and A.J. Power: Adequacy for algebraic effects $\leadsto$ side effects generated by algebraic operations.

What is this talk about?

Example

- Exceptions: $T(X) = X + E$, $raise_e$
- Non-determinism: $T(X) = \mathscr{P}(X)$, $\oplus$
- Probabilistic Computations: $T(X) = \mathscr{D}(X)$, $\oplus_p$
- Global State: $T(X) = S \Rightarrow (S \times S)$, $write$, $read$

Question

Are things simpler if we focus on monadic programs built using domain specific operations? Yes, they are!

Goals

- State of the art: coinductive techniques for e.g. λ-calculi with non-deterministic choice, errors, probabilistic choice etc.
- But these results are very specific to the languages considered.

Goal

Come up with general coinductive techniques parametric over monads and operations

$\rightsquigarrow$ Coinductive reasoning for families of domain specific monadic languages for free.

Results

- We studied CbN effectful untyped λ-calcului.
- We defined a general and intuitive notion of applicative bisimulation and prove it is sound and complete wrt contextual equivalence.
- The framework is general and surprisingly easy.
- It generalizes most of the specific notions of bisimulations given so far.
- We developed a number of up-to techniques that allow simple equational reasoning over programs.

$$
\begin{aligned}
\lambda x.\lambda y.x \oplus (x \oplus y) \sim \lambda x.\lambda y.y \oplus x \quad & \Leftarrow \quad x \oplus (x \oplus y) \approx y \oplus x \\
& \Leftarrow \quad (x \oplus x) \oplus y \approx y \oplus x \\
& \Leftarrow \quad x \oplus y \approx y \oplus x \\
& \Leftarrow \quad y \oplus x \approx y \oplus x
\end{aligned}
$$

Bisimulations for Algebraic Effects

Language and Semantics

- The basic language we consider is untyped λ-calculus.
- Side effects via a monad T. The result of a program is then an elements of $T(\mathit{Values})$.

Example

- Non-termination: values are elements of $\mathit{Values} + \{\bot\}$.
- Non-determinism: values are elements of $\mathscr{P}(\mathit{Values})$.
- Probabilistic non-determinism: values are elements of $\mathscr{D}(\mathit{Values})$.

Language and Semantics

- We need a monad T and a signature Σ of operation symbols:

$$\mathtt{t} ::= \mathtt{x} \mid \lambda \mathtt{x}.\mathtt{t} \mid \mathtt{t}\mathtt{t} \mid \mathtt{op}(\mathtt{t}, \ldots, \mathtt{t})$$

- For each n-ary operation $\mathtt{op}$ an interpretation: $\mathtt{op}^T : T(X)^n \to T(X)$
- We model computations using domain-theoretic concepts: we require $T(X)$ to be a dcpo.
- Concretely, we want a functor $T : \mathrm{Set} \to \Sigma\text{-dcpo}$ such that T carries a monadic structure under the forgetful functor.

Remark

Most of the monads used in programming satisfy these conditions (e.g. all the above ones).

Some Examples

- Non-termination: $T(X) = X + \{\bot\}$
  No operations. $X + \{\bot\}$ is a dcpo (flat domain) with least element $\bot$.
- Non-determinism: $T(X) = \mathscr{P}(X)$
  There is a binary operation $\oplus$ for non-deterministic choice interpreted as set-theoretic union. $\mathscr{P}(X)$ is a dcpo under the inclusion order.
- Probabilistic computations: $T(X) = \mathscr{D}(_)$
  We have a family of binary operations $\oplus_p$ parametrized over $p \in [0,1]$. $\oplus_p$ is interpreted as probabilistic choice ($x \oplus_p y = p \cdot x + (1-p) \cdot y$). $\mathscr{D}(X)$ is a dcpo under the pointwise order.

Language and Semantics

- Having to reason about side effects, we focus on computations rather than on programs.
- Indeed side effects occur during the execution of a program.

Definition

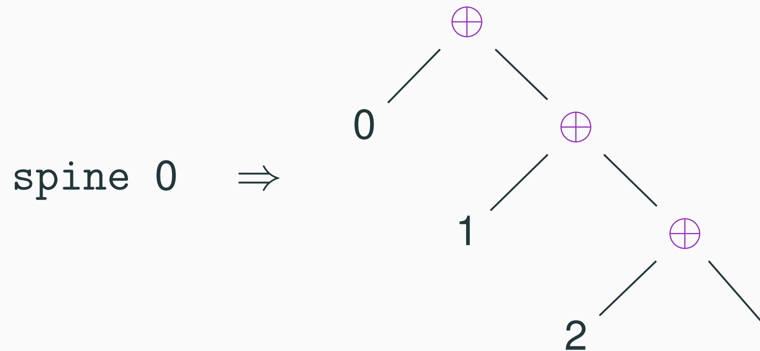
The execution of a program is the possibly infinite tree recording all the side effects that could occur during a run of the program.

Example

Consider the program

```
spine = λn -> n ⊕ (spine n+1)
```

The evaluation of the program `spine` 0 is the infinite tree



i.e. the infinite term $0 \oplus 1 \oplus 2 \oplus \cdots \oplus n \oplus \cdots$

Language and Semantics

- Computation trees (infinitary Σ-terms) over values $CT(\mathscr{V})$ are defined coinductively:

$$\frac{}{\bot \in CT(\mathscr{V})} \qquad \frac{}{\lambda x.\mathtt{t} \in CT(\mathscr{V})} \qquad \frac{\mathtt{ct}_1, \ldots, \mathtt{ct}_n \in CT(\mathscr{V})}{\mathtt{op}(\mathtt{ct}_1, \ldots, \mathtt{ct}_n) \in CT(\mathscr{V})}$$

Theorem

$CT(X)$ is a dcpo under the approximation order ($\bot$ is the totally undefined tree). Moreover, it is initial in the category of Σ-dcpo and final in the category of Σ-algebras.

Initiality $\Rightarrow$ inductive operational semantics

Finality $\Rightarrow$ coinductive operational semantics

Observational Equivalences

- We are interested in observationally indistinguishable programs $\rightsquigarrow$ contextual equivalence;

"With respect to some choice of observation, two terms $\mathtt{t}, \mathtt{u}$ are contextual equivalent, notation $\mathtt{t} \equiv^{ctx} \mathtt{u}$, if for all contexts $\mathtt{C}[\,]$, the observations on $\mathtt{C}[\mathtt{t}]$ and $\mathtt{C}[\mathtt{u}]$ agree ..."

— Winskel, The Formal Semantics of Programming Languages

- The notion of observation is left on an informal level.
- What is observable of a program is usually implicitly determined by contextual equivalence itself.

Observational Equivalences

- In pure untyped λ-calculus contextual equivalence is defined by

$$\mathtt{t} \equiv^{ctx} \mathtt{u} \quad \text{iff} \quad \forall \mathtt{C}[\,].\ \mathtt{C}[\mathtt{t}]\downarrow\ \Leftrightarrow\ \mathtt{C}[\mathtt{u}]\downarrow.$$

  We thus observe convergence.

- In probabilistic λ-calculus contextual equivalence is defined by

$$\mathtt{t} \equiv^{ctx} \mathtt{u} \quad \text{iff} \quad \forall \mathtt{C}[\,].\ Pr(\mathtt{C}[\mathtt{t}]\downarrow) = Pr(\mathtt{C}[\mathtt{u}]\downarrow).$$

  We thus observe the probability of convergence.

- In non-deterministic λ-calculus contextual equivalence is defined by

$$\mathtt{t} \equiv^{ctx} \mathtt{u} \quad \text{iff} \quad \forall \mathtt{C}[\,].\quad \exists \mathtt{v}.\ \mathtt{C}[\mathtt{t}]\downarrow \mathtt{v}\ \Leftrightarrow\ \exists \mathtt{w}.\ \mathtt{C}[\mathtt{u}]\downarrow \mathtt{w}.$$

  We thus observe may convergence.

Our Starting Point

- General and formal notion of observation parametric over monads and operations.

- Two basic assumptions:
  - We can only observe side effects
  - We observe the the executions of a program.

Observational Equivalences

- In untyped calculi it is usually assumed values to be observationally indistinguishable. We define a notion of observation according to this assumption (but this is not required).

Definition

Let $1 = \{*\}$ be a one element set. Observation is the function $\mathit{obs} : CT(\mathscr{V}) \to T(1)$ defined by

$$
\begin{aligned}
\mathit{obs}(\bot) &= \bot_T \\
\mathit{obs}(\mathtt{v}) &= \eta(*) \\
\mathit{obs}(\mathtt{op}(\mathtt{ct}_1, \ldots, \mathtt{ct}_n)) &= \mathtt{op}^T(\mathit{obs}(\mathtt{ct}_1), \ldots, \mathit{obs}(\mathtt{ct}_n))
\end{aligned}
$$

Some Examples

- Pure λ-calculus: $obs : CT(\mathscr{V}) \to 1 + \{\bot\}$

$$obs(\bot) = \bot$$

$$obs(\mathtt{v}) = *$$

$$obs(\mathtt{t}) = \{*\} \quad \Leftrightarrow \quad \mathtt{t}\downarrow$$

- Non-deterministic λ-calculus: $obs : CT(\mathscr{V}) \to \mathscr{P}(1)$

$$\begin{aligned} obs(\bot) &= \emptyset \\ obs(\mathtt{v}) &= \{*\} \\ obs(\mathtt{ct}_1 \oplus \mathtt{ct}_2) &= obs(\mathtt{ct}_1) \cup obs(\mathtt{ct}_2) \end{aligned}$$

$$obs(\mathtt{t}) = \{*\} \quad \Leftrightarrow \quad \mathtt{t}\ \textit{may}\ \text{converge}$$

Some Examples

- Probabilistic λ-calculus: $obs : CT(\mathscr{V}) \to \mathscr{D}(1) \cong [0,1]$

$$obs(\bot) = 0$$
$$obs(\mathtt{v}) = 1$$
$$obs(\mathtt{ct}_1 \oplus_p \mathtt{ct}_2) = p \cdot obs(\mathtt{ct}_1) + (1-p) \cdot obs(\mathtt{ct}_2)$$

$$obs(\mathtt{t}) = Pr[\mathtt{t}\downarrow]$$

- λ-calculus with states: $obs : CT(\mathscr{V}) \to (1 \times S)^S \cong S \rightharpoonup S$

  Omitting details, $obs(\mathtt{t})$ is a partial function associating initial and final states to the execution of $\mathtt{t}$, if $\mathtt{t}$ converges, and undefined otherwise.

Observational Equivalences

- Observing a term corresponds to observing its execution:

$$\mathit{obs}(\mathtt{t}) \stackrel{\mathit{def}}{=} \mathit{obs}(\mathit{eval}(\mathtt{t})).$$

- We can now take Winskel's definition and make it formal:

Definition

With respect to a choice of observation *obs*, two terms $\mathtt{t}, \mathtt{u}$ are contextual equivalent, notation $\mathtt{t} \equiv^{\mathit{ctx}} \mathtt{u}$, if for all contexts $\mathtt{C}[\ ]$

$$\mathit{obs}(\mathtt{C}[\mathtt{t}]) = \mathit{obs}(\mathtt{C}[\mathtt{u}]).$$

From Contextual Equivalence to Applicative Bisimulation

- Proving that two programs are contextual equivalent is hard (due to universal quantification).
- We can give a sound and complete coinductive characterization of contextual equivalence $\rightsquigarrow$ easier proof techniques.
- Two programs are bisimilar if observations cannot distinguish them, and this property is invariant under further computations (i.e. application).

Remark

Usually bisimulation defined by means of relations that implicitly encode a notion of observation. This leads to definitions involving relation lifting and makes proofs quite involved.
Using an explicit notion of observation over executions of programs gives surprisingly easy proofs.

Example

$$\mathtt{t}_1 \overset{def}{=} \lambda \mathtt{x.t} \oplus \mathtt{u} \qquad \mathtt{t}_2 \overset{def}{=} \lambda \mathtt{x.t} \oplus \lambda \mathtt{x.u}$$

$$\Downarrow \qquad \Downarrow$$

$$\mathtt{ct}_1 \overset{def}{=} \lambda \mathtt{x.t} \oplus \mathtt{u} \qquad \mathtt{ct}_2 \overset{def}{=} \oplus(\lambda \mathtt{x.t},\ \lambda \mathtt{x.u})$$

We have $obs(\mathtt{ct}_1) = obs(\mathtt{ct}_2)$.This property is invariant under application:

$$\mathtt{t}_1\mathtt{s} \Rightarrow \oplus(\mathtt{t}\{\mathtt{s}/\mathtt{x}\},\ \mathtt{u}\{\mathtt{s}/\mathtt{x}\}) \qquad \mathtt{t}_2\mathtt{s} \Rightarrow \oplus(\mathtt{t}\{\mathtt{s}/\mathtt{x}\},\ \mathtt{u}\{\mathtt{s}/\mathtt{x}\})$$

where $\mathtt{t}\{\mathtt{u}/\mathtt{x}\}$ denotes $eval(\mathtt{t}[\mathtt{u}/\mathtt{x}])$.

Bisimulation

Definition

A symmetric relation $\mathscr{R} \subseteq CT(\mathscr{V}) \times CT(\mathscr{V})$ is an applicative bisimulation if whenever $\mathtt{ct}_1 \mathrel{\mathscr{R}} \mathtt{ct}_2$ we have

- $obs(\mathtt{ct}_1) = obs(\mathtt{ct}_2)$;
- For any term $\mathtt{t}$, $\mathtt{ct}_1\{\mathtt{t}/\mathtt{x}\} \mathrel{\mathscr{R}} \mathtt{ct}_2\{\mathtt{t}/\mathtt{x}\}$.

Bisimilarity $\sim$ is defined as the largest applicative bisimulation.

- $\mathtt{ct}\{\mathtt{t}/\mathtt{x}\}$ is the computation tree obtained from $\mathtt{ct}$ by replacing all leaves (which are of the form $\lambda\mathtt{x.u}$) with $eval(\mathtt{u}[\mathtt{t}/\mathtt{x}])$.

Some Results

Theorem (Full Abstraction)

$$\mathtt{t} \equiv^{ctx} \mathtt{u} \quad \Leftrightarrow \quad \mathtt{t} \sim \mathtt{u}$$

- The proof is via CIU using a straightforward generalization of Howe's technique for the pure λ-calculus.

Theorem (Up-to Equations)
Every equational theory over computation trees which is sound in the monad is contained in $\sim$.

- Powerful up-to technique.
- Extremely useful since several monads have an equational presentation.
- Syntactical proofs for contextual equivalence.

Examples (Optional)

- Equational theory for non-deterministic choice

$$
\begin{aligned}
x \oplus x &\approx x \\
x \oplus y &\approx y \oplus x \\
x \oplus (y \oplus z) &\approx (x \oplus y) \oplus z
\end{aligned}
$$

This theory is sound in $\mathscr{P}(\mathscr{V})$ with the $\oplus$ interpreted as $\cup$. As a consequence, we have

$$\mathtt{ct}_1 \approx \mathtt{ct}_2 \quad \Rightarrow \quad \mathtt{ct}_1 \sim \mathtt{ct}_2$$

Examples (Optional)

- Equational theory for probabilistic non-determinism:

$$\begin{aligned} x \oplus_1 y &\approx x \\ x \oplus_p x &\approx x \\ x \oplus_p y &\approx y \oplus_{1-p} x \\ (x \oplus_p y) \oplus_q z &\approx x \oplus_{pq} \left(y \oplus_{\frac{q(1-p)}{1-pq}} z\right) \end{aligned}$$

  This theory is sound in $\mathscr{D}(\mathscr{V})$ with the $\oplus$ interpreted as probabilistic choice. As a consequence, we have

$$\mathtt{ct}_1 \approx \mathtt{ct}_2 \quad \Rightarrow \quad \mathtt{ct}_1 \sim \mathtt{ct}_2$$

- The equational theory for probabilistic and non-deterministic choice is obtained from the above one plus the equation

$$x \oplus (y \oplus_p z) \approx (x \oplus_p y) \oplus (x \oplus_p z)$$

Examples (Optional)

Recall the following programs

```
simpleGame = λn,score ->
 (simpleGame 1 0 ⊕ score) ⊕_{1/2n}
 (simpleGame n+1 score+1 ⊕_{0.5} simpleGame n+1 score)

game = λn,score ->
 ((game 1 0 ⊕_{2/(n+1)} game n+1 score+1) ⊕_{(n+1)/2n} game n+1 score) ⊕
 (score ⊕_{1/n} (game n+1 score+1 ⊕_{0.5} game n+1 score))
```

We show simpleGame $\sim$ game. Let

$M =$ simpleGame 1 0 $\qquad N =$ score

$P =$ simpleGame n+1 score+1 $\qquad Q =$ simpleGame n+1 score

Examples (Optional)

From

$$(M \oplus N) \oplus_{\frac{1}{2n}} (P \oplus_{\frac{1}{2}} Q) \approx (M \oplus_{\frac{1}{2n}} (P \oplus_{\frac{1}{2}} Q)) \oplus (N \oplus (P \oplus_{\frac{1}{2}} Q))$$

$$\{ \text{ by } x \oplus (y \oplus_p z) \approx (x \oplus_p y) \oplus (x \oplus_p z) \ \}$$

$$\approx ((M \oplus_{\frac{2}{1+2n}} P) \oplus_{\frac{1+2n}{4n}} Q) \oplus (N \oplus_{\frac{1}{2n}} (P \oplus_{\frac{1}{2}} Q))$$

$$\{ \text{ by } x \oplus_p (y \oplus_q z) \approx (x \oplus_{\frac{p}{p+q-pq}} y) \oplus_{p+q-pq} z \ \}$$

Follows

```
game ~ λn,score ->
  ((game 1 0 ⊕_{2/(1+2n)} game n+1 score+1) ⊕_{(1+2n)/4n} game n+1 score) ⊕
  (score ⊕_{1/2n} (game n+1 score+1 ⊕_{1/2} game n+1 score))
```

and thus `simpleGame ~ game`.

Conclusions and Future Works

Conclusions

- We have proposed a notion of bisimulation that generalizes most of the existing ones.
- Such a notion of bisimulation is built over a formal notion of observation parametric over monads and operations. This abstract treatment is of methodological importance for the theory behind operational equivalences of programs.
- We proved a full Abstraction theorem and developed useful up-to techniques.

Future Works

- Quite a lot! But two main directions:
- Get rid of the inductive (dcpo) structure behind the notion of observation, thus relying only on finality of $CT(\mathscr{V})$.
- To do so, we need a better categorical formulation of the above theory.
- The other direction would like to get ride of the monadic and categorical structure, thus relying only on equational specifications.
- Here the problem is to come up with a good extension of a finitary equational theory $\approx$ to infinitary terms.

Thank you very much for the attention!