Simpoulet: an attempt at proving environmental bisimulations in Coq

Jacques Garrigue (Nagoya University)

Pierre-Marie Pédrot (ENS Lyon)

Environmental Bisimulation (1)

- A technique for proving program equivalence.
- Particularly interesting as it allows proving equivalences with higher-order stateful programs, with type abstraction (*e.g.* ML programs with modules)

  Eijiro Sumii: *A Complete Characterization of Observational Equivalence in Polymorphic Lambda Calculus with General References* **[CSL 2009]**

Environmental Bisimulation (2)

- Prove that two programs are equivalent by proving that they are bisimilar

- Use strong forms of bisimulations that take advantage of the fact programs are typed

- Allow considering programs modulo reduction/context/allocation, making proof easier

Definition

X is an environmental simulation if

1. For any $(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \in X$,

   (a) [Reduction] If $s \triangleright M \longrightarrow t \triangleright N$ then $s' \triangleright M' \xrightarrow{*} t' \triangleright N'$ for some t' and N' with $(\Delta, \mathcal{R}, t \triangleright N, t' \triangleright N', \tau) \in X$

   (b) [Evaluation] If $M = V$ then $s' \triangleright M' \xrightarrow{*} t' \triangleright V'$ for some t' and V' with $(\Delta, \mathcal{R} \cup \{(V, V', \tau)\}, s, t') \in X$

2. For any $(\Delta, \mathcal{R}, s, s') \in X$,

   (a) [Application] If $(\lambda x{:}\Delta^1(\tau_1).M, \lambda x{:}\Delta^2(\tau_1).M', \tau_1 \rightarrow \tau_2) \in \mathcal{R}$, then $(\Delta, \mathcal{R}, s \triangleright [V/x]M, s' \triangleright [V'/x]M', \tau_2) \in X$ for any $(V, V', \tau_1) \in (\Delta, \mathcal{R})^\star$

(e) [Allocation]
$(\Delta, \mathcal{R} \cup \{(l, l', \tau \ \ \text{ref})\}, s \uplus \{l \mapsto V\}, s' \uplus \{l' \mapsto V'\}) \in X$ for any $l \notin \text{dom}(s)$, $l' \notin \text{dom}(s')$ and $(V, V', \tau) \in (\Delta, \mathcal{R})^\star$

(f) If $(l, l', \tau \text{ ref}) \in \mathcal{R}$ then
[Dereference] $(\Delta, \mathcal{R} \cup (s(l), s'(l'), \tau)\}, s, s') \in X$
[Update] $(\Delta, \mathcal{R}, s\{l \mapsto V\}, s'\{l' \mapsto V'\}) \in X$ for any $(V, V', \tau) \in (\Delta, \mathcal{R})^\star$

where X is typed, *i.e.* there exists Σ and Σ' such that

- $\Sigma \vdash M : \Delta^1(\tau)$ and $\Sigma' \vdash M' : \Delta^2(\tau)$
- $\Sigma \vdash s$ and $\Sigma' \vdash s'$
- $\Sigma \vdash V : \Delta^1(\tau)$ and $\Sigma' \vdash V' : \Delta^2(\tau)$ for all $(V, V', \tau) \in \mathcal{R}$

and $(\Delta, \mathcal{R})^\star$ is the context closure of $\mathcal{R}$

$$\{([\bar{V}/\bar{x}]\Delta^1(C), [\bar{V}'/\bar{x}]\Delta^2(C), \tau) \mid \text{dom}(\Delta), \bar{x} : \bar{\tau} \vdash C : \tau, (\bar{V}, \bar{V}', \bar{\tau}) \in \mathcal{R}\}$$

Up-to techniques

Proofs are made easier by allowing a larger relation on the right hand side.

- Up-to reduction: a configuration pair is in the extended relation if it reduces to a related pair.

- Up-to context: a configuration is in the extended relation if there is a context and a list of related pairs such that it can be obtained by substituting each side of the pairs in the context.

- Up-to allocation: allow some extra allocated reference cells, initialized with related values.

Characterization Theorem

Theorem 1 *Environmental bisimilarity (the largest environmental bisimulation) equals observational equivalence.*

Goals

- Prove the soundness and completeness of environmental bisimulation (including up-to techniques).
- Provide a toolkit to prove equivalences of programs.

Formalization

(Work by Pierre-Marie Pédrot)

First we need to define a typed language, with a small step semantics.

We used *locally nameless co-finitely quantified* syntax **[Aydemir *et al.*]**.

- use De Bruijn indices for local variables
- use co-finite quantification for global variables

We also avoided putting types inside terms.

LNCFQ Syntax

Judgement: $S, \Sigma, \Gamma \vdash M : \tau$ where

- S is a set of type variables : Set[Var]
- Σ is the store typing : Map[Var,Typ]
- Γ is the typing environment : Map[Var,Typ]

$$\frac{(\forall x \notin L) \quad S, \Sigma, \Gamma \uplus \{x \mapsto \tau\} \vdash M^x : \tau'}{S, \Sigma, \Gamma \vdash \lambda M : \tau \rightarrow \tau'}$$

$$\frac{\begin{array}{l} S, \Sigma, \Gamma \vdash M : \exists \tau \\ (\forall x \notin L_1, \alpha \notin L_2) \quad S \uplus \{\alpha\}, \Sigma, \Gamma \uplus \{x \mapsto \tau^\alpha\} \vdash N^x : \tau' \end{array}}{S, \Sigma, \Gamma \vdash \text{open } M \text{ in } N : \tau'}$$

Advantages of LNCFQ Syntax

- Limiting indices to local variables avoids both substitution and shifting in many cases

- Making the choice of variables co-finite makes proofs of preservation easier: when weakening one just has to enlarge the avoidance set

- However one needs many commutation lemmas between De Bruijn instantiation and variable substitution

Type system proofs

Proved type soundness (preservation and progress) with respect to small-step reduction.

- Despite the large number of typing (16) and reduction rules (23), the proofs stay small.

- Heavy use of automation to share tactics between different cases.

- Used reflection for tactics about finite sets and maps.

Formalization of simulations

- Converted from set-theoretic to inductive definitions
- Needed to separate the term and store part of relations
- Also needed care to take the typing into account
- One slight simplification: since types do not appear inside terms, context closure $(\Delta, \mathcal{R})^\star$ actually does not depend on Δ

Typing of the value relation

```
Record typing_vrel Δ1 Δ2 Σ1 Σ2 (R : vrel) : Prop := {
  typing_vrel_closed_l : forall X, ∅ ⊩ Δ1 X;
  typing_vrel_closed_r : forall X, ∅ ⊩ Δ2 X;
  typing_vrel_wf_l : wf_env ∅ Σ1 [∅];
  typing_vrel_wf_r : wf_env ∅ Σ2 [∅];
  typing_vrel_value_l : forall V1 V2 τ, R V1 V2 τ -> value V1;
  typing_vrel_value_r : forall V1 V2 τ, R V1 V2 τ -> value V2;
  typing_vrel_l : forall V1 V2 τ,
    R V1 V2 τ -> typing ∅ Σ1 [∅] V1 (τ ← Δ1);
  typing_vrel_r : forall V1 V2 τ,
    R V1 V2 τ -> typing ∅ Σ2 [∅] V2 (τ ← Δ2)
}.

Record typing_prel Δ1 Δ2 Σ1 Σ2 R s1 s2 M1 M2 τ := ...
Record typing_srel Δ1 Δ2 Σ1 Σ2 R s1 s2 := ...
```

Up-to techniques

The definitions are actually quite complicated.

Here is the relation for up-to renaming and reduction.

$$
\begin{aligned}
X^{\rightarrow} = {} & \{(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \mid (\Delta, \mathcal{R}, t \triangleright N, t' \triangleright N', \tau) \in X^{\pi}, \\
& \qquad s \triangleright M \xrightarrow{*} t \triangleright N, s' \triangleright M' \xrightarrow{*} t' \triangleright N'\} \\
& \cup \{(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \mid s \triangleright m \text{ diverges}\} \\
& \cup \{(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \mid (\Delta, \mathcal{R} \cup \{(V, V', \tau)\}, t, t') \in X^{\pi}, \\
& \qquad s \triangleright M \xrightarrow{*} t \triangleright V, s' \triangleright M' \xrightarrow{*} t' \triangleright V'\} \\
& \cup \{(\Delta, \mathcal{R}, s, s') \mid (\Delta, \mathcal{R}, s, s') \in X^{\pi} \\
X^{\pi} = {} & \{(\Delta, \mathcal{R}^{\pi}, \pi(s) \triangleright \pi(M), s' \triangleright M', \tau) \mid (\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \in X\} \\
& \cup \{(\Delta, \mathcal{R}^{\pi}, \pi(s), s' \mid (\Delta, \mathcal{R}, s, s') \in X\} \\
\mathcal{R}^{\pi} = {} & \{(\pi(V), V', \tau) \mid (V, V', \tau) \in \mathcal{R}\}
\end{aligned}
$$

Up-to-reduction/renaming closure

```
Inductive prel_red_closure : prel :=
| prel_red_red : forall π R s1 s1' s2 s2' t1 t1' t2 t2' τ,
  bijection π -> (prel_rename π Xp) R [s1' · t1'] [s2' · t2'] τ ->
  #reduction [s1 · t1] [s1' · t1'] ->
  #reduction [s2 · t2] [s2' · t2'] ->
  typable_prel R s1 s2 t1 t2 τ ->
  prel_red_closure R [s1 · t1] [s2 · t2] τ
| prel_red_div : forall (R : vrel) s1 s2 t1 t2 τ,
  chain reduction [s1 · t1] -> typable_prel R s1 s2 t1 t2 τ ->
  prel_red_closure R [s1 · t1] [s2 · t2] τ
| prel_red_eval : forall π (R : vrel) s1 s1' s2 s2' t1 t1' t2 t2' τ,
  bijection π -> #reduction [s1 · t1] [s1' · t1'] ->
  #reduction [s2 · t2] [s2' · t2'] -> value t1' -> value t2' ->
  (srel_rename π Xs) (R ∪ [t1' ~ t2' | τ])%vrel s1' s2' ->
  typable_prel R s1 s2 t1 t2 τ ->
  prel_red_closure R [s1 · t1] [s2 · t2] τ.
```

Proof for up-to-reduction/renaming

- Soundness theorem is close to 200 lines
- Using many hand-crafted automation tactics
- Lots of lemmas for renaming
- For reduction, soundness of typing is enough

Up-to-context closure

Extends the relation to each pair of term in any evaluation context.

$$
\begin{aligned}
X^\star = \{ & (\Delta, \mathcal{R}, s \triangleright [\bar{V}/\bar{x}]E[M], s' \triangleright [\bar{V}'/\bar{x}]E[M'], \tau) \mid \\
& (\Delta_0, \mathcal{S}, s \triangleright M, s' \triangleright M', \tau_0) \in X, \Delta \subseteq \Delta_0, \\
& \mathcal{R} \subseteq \mathcal{S}^\star, FTV(\mathcal{R}) \subseteq \mathrm{dom}(\Delta), (\bar{V}, \bar{V}', \bar{\tau}) \in \mathcal{S}, \\
& \mathrm{dom}(\Delta_0), \bar{x}{:}\bar{\tau} \vdash E : \tau, FTV(\tau) \subseteq \mathrm{dom}(\Delta) \} \\
\cup\ & \ldots
\end{aligned}
$$

This allows to prove easily many program equivalences.

Problem with up-to-context

Pierre-Marie could not prove it in Coq.

- Typing becomes very involved due to simultaneous substitution.
- Just proving soundness of up-to-context for the application case took 140 line, more of half of it for typing. (Not including infrastructure lemmata.)
- Similar for type application.
- Abandonned in the middle of the existential unpacking case.

A simple benchmark

– Since we couldn't prove up-to-context, most examples stay hard to prove.

– To ensure the usability of the formalization, I proved that the identity relation is an environmental bisimulation, using up-to-reduction.

Identity environmental relation

```
Let is_id (R : trm -> trm -> typ -> Prop) :=
  forall x y T, R x y T -> x = y ∧ value x.
Let has_fv (R : trm -> trm -> typ -> Prop) :=
  exists L, forall x y T, R x y T -> typ_fv T ⊆ L.

Inductive myprel : vrel -> program -> program -> typ -> Prop :=
  myprel1 : forall R Δ Σ s M τ, is_id R -> has_fv R ->
    store_typing ∅ Σ [∅] s ->
    typing ∅ Σ [∅] M (τ ← Δ) ->
    typing_vrel Δ Δ Σ Σ R ->
    myprel R [s · M] [s · M] τ.
Inductive mysrel : vrel -> store -> store -> Prop :=
  mysrel1 : forall R Δ Σ s, is_id R -> has_fv R ->
    store_typing ∅ Σ [∅] s ->
    typing_vrel Δ Δ Σ Σ R ->
    mysrel R s s.
```

Identity environmental relation

```
Let is_id (R : trm -> trm -> typ -> Prop) :=
  forall x y T, R x y T -> x = y ∧ value x.
Let has_fv (R : trm -> trm -> typ -> Prop) :=
  exists L, forall x y T, R x y T -> typ_fv T ⊆ L.
```

- `is_id` ensures that `R` is a subrelation of the identity.
- `has_fv` ensures that we can find fresh type variables.
- Everything is well-typed.

Proof of reflexivity

```
Lemma up2red_sim_myrel : up2red_simulation myprel mysrel.
Corrolary env_sim_myrel :
  environmental_simulation (prel_red_closure myprel mysrel)
                           (srel_red_closure mysrel).
```

- The proof took 600 lines (including extra infrastructure lemmata).
- The proof is mostly about typing and renaming.
- It may seem trivial, but hopefully we can generalize the techniques used to automatize typing and renaming in proofs.

Conclusion

- The original goal was 2-fold:
  - Proving the soundness and completeness of environmental bisimulation (including up-to techniques).
  - Providing a toolkit to prove equivalences of programs.
- Eventually, only half of the first part was done.
  - Typing and simultaneous substitution are tricky.
  - If we can overcome that, there is some hope.

For the curious

All the proofs are in a public repository:

`http://sourceforge.net/projects/simpoulet/`