A Practical Approach to Co-induction in Twelf

Alberto Momigliano
Laboratory for Foundations of Computer Science
University of Edinburgh &
DSI, University of Milan

Funded in part by EU-project Mobius (IST-2005-015905)
TYPES 2006, Nottingham, April 18-21, 2006
Motivation

- Common complaint (see the POPLmark challenge): *Twelf* is a great system but it cannot do “⟨insert your favorite theorem prover feature⟩”, so we’ll suffer thru a first-order encoding to utilize systems where that feature is native).

- We’ll show a way to do proofs by co-induction in Twelf here and now.

- The basic idea (dating back to Milner’s original CCS [1980]): define, when possible, your co-inductive relation *inductively*, by mimicking the construction of *gfix* by ordinal powers up to $\omega$ (see also Miller et al 1997).

- No change to the Twelf’s meta-theory, hence the *totality* checker is available and can certify relational type families as proofs.

- No free lunch: It’s a bit awkward and better seen as an incentive to develop the appropriate meta-theory. Still, all proofs in Milner [1980] are inductive.
Technical background

- Recall the set-theoretic characterization of a (co)inductive definition. Let $f$ be a monotone endo-function on a complete lattice $P$:

  $$lfix(f) = \bigwedge\{x \mid f(x) \leq x\}.$$  
  Dually, $gfix(f) = \bigvee\{x \mid x \leq f(x)\}$

- Fix a universe $\mathcal{U}$. Its powerset is a complete lattice. A rule set [Aczel 77] is any set $\mathcal{R} \subseteq \mathcal{U} \times 2^{\mathcal{U}}$ (here denumerable); let $\Phi_{\mathcal{R}} : 2^{\mathcal{U}} \to 2^{\mathcal{U}}$ and define

  $$\Phi_{\mathcal{R}}(A) = \{a \in \mathcal{U} \mid (a, G) \in \mathcal{R}, G \subseteq A\}$$

- The set co-inductively defined by $\mathcal{R}$ over $\mathcal{U}$ is $gfix(\Phi_{\mathcal{R}})$, namely $CId(\mathcal{R}) = \bigvee\{A \mid A \subseteq \Phi_{\mathcal{R}}(A)\}$. As a proof-rule:

  $$\frac{\exists A \cdot a \in A \quad A \subseteq \Phi_{\mathcal{R}}(A)}{a \in CId(\mathcal{R})} CI$$
The trick

- Recall the notion of *ordinal power* $f^\uparrow \downarrow \alpha$ of a function $f$ on a complete lattice. From Tarski’s theorem, if $f$ is monotone, by repeated application to the empty set, it will converge to the set inductively defined by the rule set; if it is continuous, it will converge in at most $\omega$ steps. Note that $\Phi_R$ is continuous.

- What about the dual? Can we characterize $gfix$ via iteration of the operator to the universe of discourse? Yes, provided it satisfies co-continuity (preservation of meets): $f(\bigvee X) = \bigvee (fX)$ for every directed $X \subseteq \mathcal{U}$.

  \[
  \begin{align*}
  f \downarrow 0 &= \mathcal{U} \\
  f \downarrow n+1 &= \Phi_R(f \downarrow n) \\
  f \downarrow \omega &= \bigcap \{f \downarrow k \mid k \in \omega\} = gfix(\Phi_R)
  \end{align*}
  \]

- In practical terms, we are looking for decidable conditions on the “shape” of the rule set, so that co-continuity holds. One such example is “finite branching”, as we will see.
First example: divergence in the untyped $\lambda$-calculus

\[
\begin{align*}
\frac{\uparrow e_1}{\downarrow (e_1 e_2)} & \quad \text{div} - \text{app1} \\
\frac{e_1 \downarrow \lambda x. e}{\uparrow (e_1 e_2)} & \quad \text{div} - \text{app2}
\end{align*}
\]

- In words: a lambda never diverges. An application diverges if $e_1$ diverges; otherwise it converges to a lambda, its application to $e_2$ diverges.

- The $\text{lfix}$ is empty, yet the $\text{gfix}$ of this rules encode divergence. However, it can be shown (trust me, it follows from determinism of evaluation) that the associated operator is co-continuous, so the set can be also computed inductively.

- So, let’s write some Twelf code. First declarations for expressions and lazy evaluation. I assume familiarity with Twelf’s idea of encoding theorems as relations between type families that need to be verified as total functions.
Evaluation in the lazy $\lambda$-calculus

exp : type.
lam : (exp -> exp) -> exp. %%% Note HOAS here
app : exp -> exp -> exp.

%block L1 : block {x:exp}. %%% Ignore this for now
%worlds (L1) (exp).

eval : exp -> exp -> type.
%mode +{E:exp} -{V:exp} eval E V.

ev_lam : eval (lam E) (lam E).

ev_app : eval (app E1 E2) V
<- eval E1 (lam E)
<- eval (E E2) V. %% subst as meta-level application
Divergence in the untyped $\lambda$-calculus: inductive encoding

%% fixed point indexes
index : type.

zz : index.
ss : index -> index.

%% divergence has additional argument 'index'
ndiverge : index -> exp -> type.
%mode ndiverge +N +E.

divbase : ndiverge zz E.

div_app1 : ndiverge (ss N) (app E1 E2)
<- ndiverge N E1.

div_app2 : ndiverge (ss N) (app E1 E2)
<- eval E1 (lam E)
<- ndiverge N (E E2).
Adequacy, I

• Finally, say that $\text{diverge } e$ iff $\forall n : \text{index. ndiverge } n \ e$

• Adequacy: one direction, induction on “n”, using only the fix point property of divergence. Hence encode the latter and prove it entails the inductive version:

$$\text{div} : \text{exp} \rightarrow \text{type.}$$

$$\text{dv_app1} : \text{div} (\text{app } E_1 E_2)$$
$$\quad \leftarrow \text{div } E_1 .$$

$$\text{dv_app2} : \text{div} (\text{app } E_1 E_2)$$
$$\quad \leftarrow \text{eval } E_1 (\text{lam } E_1')$$
$$\quad \leftarrow \text{div } (E_1' E_2) .$$

$$\text{dvdiv} : \{N: \text{index}\} \text{ div } E \rightarrow \text{ndiverge } N \ E \rightarrow \text{type.}$$

$$\text{d0} : \text{dvdiv } \text{zz } \_ \_ \text{divbase.}$$

$$\text{d1} : \text{dvdiv } (\text{ss } N) (\text{dv_app1 } D) (\text{dv_app1 } DN)$$
$$\quad \leftarrow \text{dvdiv } N \ D \ DN .$$

$$\text{d2} : \text{dvdiv } (\text{ss } N) (\text{dv_app2 } D VV) (\text{dv_app2 } DN VV)$$
$$\quad \leftarrow \text{dvdiv } N \ D \ DN .$$

%total N (dvdiv N P Q).
Adequacy, II

- Other way is meta-theoretical: need to apply CI rule, i.e. to show that \( ndiverge \) is a “simulation”. This follows from definitions and from the fact that the (big-step) evaluation is determinate (a fortiori, finitely branching).

- CAVEAT: co-induction is defined via universal quantification. It \textbf{cannot} be queried existentially as a standard logic program. The preservation of the invariant must be checked at \textbf{every} stage of the fixed point construction.

- To show, e.g. \( diverge \omega \) we need to prove, by induction, \( ndiverge n \omega \), for all \( n \).
Proving $\Omega$ diverges

- Theorem: the $\Omega$ combinator diverge. The standard formal proof (in Hybrid) requires to guess the right simulation, which is in this case $\{\omega\}$ and afterward a 10 commands script. In Coq you can use the $\text{CoFix}$ tactics and guarded induction, but of course it clashes with HOAS and the overall soundness of the latter still an issue.

- You write the theorem as relation in Twelf, where the first 2 cases would not occur in an co-inductive proof:

  $\omega = \text{app} (\text{lam} [x] (\text{app} x x)) (\text{lam} [x] (\text{app} x x))$.

  $\text{ndiverge} I \omega \rightarrow \text{type}$.

  $\text{divomegaR}: \{I : \text{index}\}$

  $\text{dub : ndivomegaR zz divbase}$.

  $\text{dd : ndivomegaR (ss zz) (div_app1 divbase)}$.

  $\text{dus : ndivomegaR (ss I) (div_app2 D1 (ev_lam))}$

  $\leftarrow \text{ndivomegaR I D1}$. 
Proving $\Omega$ diverges, cont’ed

• ... and have it checked for totality:

```twelf
%mode +{I:index} -{Q:diverge I omega} (divomegaR I Q).
%worlds () (divomegaR _ _).
%total I (divomegaR I P).
```

• Luckily, Carsten’s meta-theorem prover will also find the realizer for you:

```twelf
%theorem div_omega: forall {N:index} exists {Pi : ndiverge N omega} true.
%prove 3 N (div_omega N _).
```

%%% Twelf’s answer:
%theorem div_omega : {N:index} diverge N omega -> type.
%prove 3 N (div_omega N _).
%mode +{N:index} -{Pi:diverge N omega} (div_omega N Pi).
%QED
%skolem div_omega#1 : {N:index} diverge N omega.
```
Applicative simulation (Ong-Abramski)

- The largest relation defined by:

\[
\forall e'. e \downarrow \lambda x. e' \rightarrow \exists f': f \downarrow \lambda x. f' \land \forall m. e'[m/x] \leq f'[m/x] \quad \text{sim} \\
\]

- Let’s play the same trick: \( e \leq f \) implies \( \forall n : \text{index}. \ sim n e f \). Conversely, \( sim n e f \) is indeed a simulation.

- Note that, by the reduced syntax of LF (no existentials), we have to split the judgment into two mutual recursive ones, so that \( F' \) is correctly quantified.

- However, the use of hypothetical judgments obliterates the difference between simulation and its open extension [Lassen 99], which saves us some serious pain while formalising the proofs.
Applicative simulation: Twelf encoding

\[ \text{sim} : \text{index} \to \text{exp} \to \text{exp} \to \text{type} \]
\%mode \text{sim} +N +E +F.

\[ \text{simbody} : \text{index} \to (\text{exp} \to \text{exp}) \to \text{exp} \to \text{type} \]
\%mode \text{simbody} +N +E +F.

\[ \text{sim\_all} : \text{sim} \; \text{zz} \; \text{E} \; \text{F}. \quad \%\% \text{everything goes at step 0} \]

\[ \text{simf} : \text{sim} \; (\text{ss} \; \text{I}) \; \text{E} \; \text{F} \]
\[ \quad \leftarrow (\{\text{E}':\text{exp} \to \text{exp}\} \; \text{eval} \; \text{E} \; (\text{lam} \; \text{E}') \]
\[ \quad \quad \rightarrow \text{simbody} \; \text{I} \; \text{E}' \; \text{F}). \]

\[ \text{sb} : \text{simbody} \; \text{I} \; \text{E}' \; \text{F} \]
\[ \quad \leftarrow \text{eval} \; \text{F} \; (\text{lam} \; \text{F}') \]
\[ \quad \leftarrow (\{\text{m}:\text{exp}\} \; \text{sim} \; \text{I} \; (\text{E}' \; \text{m}) \; (\text{F}' \; \text{m})). \]
A tiny bit of meta-theory: reflexivity of simulation

% Reflexitivity of simulation

nsimrefl: \{N : index\} \{E : exp\} sim N E E -> type.

nsimr_z : nsimrefl zz _ sim_all.
nsimr_s : nsimrefl (ss N) _
    (simf ([e:exp -> exp][u : eval E1 (lam e)]
        sb ([x:exp] NS e u x) u))
    <- ([e:exp -> exp] {u : eval E1 (lam e)} {x:exp}
        nsimrefl N _ (NS e u x)).

%mode nsimrefl +I +E -D.
%block L2 : some \{E:exp\} block \{e:exp -> exp\}{u:eval E (lam e)} \{x:exp\}
%worlds (L1 | L2) (exp).
%worlds (L2) (nsimrefl _ _ _).
%total M (nsimrefl M _ _).
Conclusion: what have we learned?

- What I’ve presented today is little more than a patch.

- However, it shows that with a very little thought you do not need to rubbish a system such as Twelf for lacking a feature you may deem fundamental.

- It may be interesting to play out some more extensive examples (Howe’s proof) to see the limitations of this approach.

- At the same time, I think that there is mounting evidence that co-induction should be a first class citizen in Twelf-land.

- This may entail quite a different approach to totality checking, as the obvious fix, guarded induction, does not seem compatible with Twelf’s current operational semantics.