A Practical Approach to Co-induction in Twelf

Alberto Momigliano

Laboratory for Foundations of Computer Science

University of Edinburgh

Funded 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 is cannot do "$\langle$insert your favorite theorem prover feature$\rangle$" and somebody says, you may as well junk it.

- We are going to show a way to do proofs by co-induction in Twelf **now**.

- No change to the Twelf's meta-theory, hence the *totality* checker is available.

- The basic idea: dating back Milner's CCS [1980]: define, whenever possible, your co-inductive relation, *inductively*. Mentioned also in Miller et al 1997.

- 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. In general, proof by co-induction are sporadic (only 3 co-inductive lemmas in Howe's proof of congruence of applicative bisimulation)

Technical development

- Start with a set-theoretic characterization of a (co)inductive definition. A *rule set* $\mathcal{R}$ [Aczel 77], a possibly (denumerable) infinite set of pairs $\langle G, a \rangle$ (notation: $a \leftarrow G$) on an universe $\mathcal{U}$, such that $a \in \mathcal{U}, G \subseteq 2^{\mathcal{U}}$.

- there is an alternative characterization via fix points of monotone operators: let $\Phi_{\mathcal{R}} : 2^{\mathcal{U}} \rightarrow 2^{\mathcal{U}}$ and define $\Phi_{\mathcal{R}}(A) = \{a \in \mathcal{U} \mid a \leftarrow G \in \mathcal{R}, G \subseteq A\}$

- The set *co-inductively* defined by $\mathcal{R}$ is the greatest $\mathcal{R}$-dense set, namely $CId(\mathcal{R}) = \bigvee\{A \mid A \subseteq \Phi_{\mathcal{R}}(A)\}$

$$\frac{\exists A \,.\, a \in A \qquad A \subseteq \Phi_{\mathcal{R}}(A)}{a \in CId(\mathcal{R})}\, CI$$

Technical development, cont'ed

- From Tarski's theorem, if $\Phi_{\mathcal{R}}$ 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 at most in ω steps.

- 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 meet's)

$$\begin{array}{rcl} T_0 & = & \mathcal{U} \\ T_{n+1} & = & \Phi_{\mathcal{R}}(T_n) \\ T_\omega & = & \cap\{T_k \mid k \in \omega\} \quad = gfix(\Phi_{\mathcal{R}}) \end{array}$$

- In practical terms, we are looking for decidable conditions on the "shape" of the definition, so that co-continuity holds. One such example is "finite branching", as we will see.

First example: divergence in the untyped λ-calculus

$$\frac{\Uparrow\ e_1}{\Uparrow\ (e_1\ e_2)}\,\mathtt{div}-\mathtt{app1} \qquad \frac{e_1 \Downarrow \lambda x.e \qquad\qquad \Uparrow\ e[e_2/x]}{\Uparrow\ (e_1\ e_2)}\,\mathtt{div}-\mathtt{app2}$$

- The gfix of this rules encode divergence. However, it can be shown (trust me, it follows from determinism if evaluation) that the associated operator is co-continuous, so the set can be 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 theorem as relation between type families that need to be verified as total functions.

Evaluation in the lazy λ-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 E1')
    <- eval (E1' E2) V.    %% note subst as  meta-level application
```

Divergence in the untyped λ-calculus: inductive encoding

```
%% fixed point indexes
index : type.

zz : index.
ss : index -> index.

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

divbase    : ndiverge zz E.    %% everything diverges at stage zero

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

- Finally, say that $\mathit{diverge}\ e$ iff $\forall k : \mathtt{index}.\ \mathtt{ndiverge\ k\ e}$. Why is this correct? One direction, easy induction on "k" (formalised in Isabelle/HOL with the newly revamped Hybrid06 package, where $\Uparrow$ is implemented as a HOL's co-inductive definition):

$$\Uparrow\ \mathtt{e} \rightarrow \forall k : \mathtt{index}.\ \mathtt{ndiverge\ k\ e}$$

- Other way: need to apply `CI` rule, hence to show that `ndiverge` is a "simulation". This follows from definitions and from the fact that the (big-step) evaluation is determinate.

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

Proving Ω diverges

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

- You write this relation in Twelf ...

```
omega = app (lam [x] (app x x)) (lam [x] (app x x)).

divomegaR: {I : index} ndiverge I omega -> type.
%mode  ndivomegaR +I  -Q.

dub : ndivomegaR zz divbase.
dd :  ndivomegaR (ss zz) (div_app1 divbase).
dus : ndivomegaR (ss I) (div_app2 D1 (ev_lam))
      <- ndivomegaR I D1.
```

Proving Ω diverges, cont'ed

- . . . and have it checked for totality:

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

- Luckily, the Carsten's meta-theorem prover will also find it for you:

```
%theorem ndiv_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:

$$\frac{\forall e'.e \Downarrow \lambda x.e' \rightarrow \exists f' :\Downarrow f\ \lambda x.f' \wedge \forall m.e'[m/x] \leq f'[m/x]}{e \leq f}\ \texttt{sim}$$

- Let's play the same trick: $e \leq f$ implies $\forall n : \texttt{index}.\ sim\ n\ e\ f$. 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 so that F' is correctly quantified.

- However, the use of hypothethical judgments obliterates the difference between simulation and its *open* extension, which saves us some serious pain while formalising the proofs.

Applicative simulation: Twelf encoding

```
sim : index -> exp -> exp  -> type.
%mode sim +N +E +F.

simbody : index -> (exp -> exp) -> exp -> type.
%mode simbody +N +E +F.

sim_all : sim zz E F.                   %% everything goes at step 0

simf : sim (ss I) E F
      <- ({E':exp -> exp} eval E (lam E')
                 -> simbody I E' F).
sb   : simbody I E' F
          <- eval F (lam F')
          <- ({m:exp} sim I (E' m) (F' m)).
```

A tiny bit of meta-theory: reflexivity of simulation

```
% Reflexitivity of simulation

nsimrefl: {N : index} {E : exp} sim N E E -> type.
%mode nsimrefl +I +E -D.

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


%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 shown today is little more than a patch.
- However, it shows that with a very little thought you do nnot need to rubbish a system such as Twelf for lacking a feature you may deem fundamental.
- It may be intersting 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.