Abstract Soundness

Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel

February 13, 2017

Abstract

This is a formalized coinductive account of the abstract development of Brotherston et al. [2], in a slightly more general form since we work with arbitrary infinite proofs, which may be acyclic. This work is described in detail in an article by the authors [1]. The abstract proof can be instantiated for various formalisms, including first-order logic with inductive predicates.

References

[1] J. C. Blanchette, A. Popescu, and D. Traytel. Soundness and completeness proofs by coinductive methods. *J. Autom. Reasoning*, 58(1):149–179, 2017.

[2] J. Brotherston, N. Gorogiannis, and R. L. Petersen. A generic cyclic theorem prover. In R. Jhala and A. Igarashi, editors, *APLAS 2012*, volume 7705 of *Lecture Notes in Computer Science*, pages 350–367. Springer, 2012.

Contents

1 Abstract Soundness

locale *Soundness* = *RuleSystem-Defs eff rules* **for**
eff :: *'rule* ⇒ *'sequent* ⇒ *'sequent fset* ⇒ *bool*

and *rules* :: *'rule stream* +
fixes *structure* :: *'structure set*
and *sat* :: *'structure* ⇒ *'sequent* ⇒ *bool*
assumes *local-soundness*:
⋀*r s sl*.
⟦*r* ∈ *R*; *eff r s sl*; ⋀*s'*. *s'* |∈| *sl* ⟹ ∀ *S* ∈ *structure*. *sat S s'*⟧
⟹
∀ *S* ∈ *structure*. *sat S s*
begin

abbreviation *ssat s* ≡ ∀ *S* ∈ *structure*. *sat S s*

lemma *epath-shift*:
assumes *epath* (*srs* @− *steps*)
shows *epath steps*
⟨*proof*⟩

theorem *soundness*:
assumes *f*: *tfinite t* **and** *w*: *wf t*
shows *ssat* (*fst* (*root t*))
⟨*proof*⟩

end

2 Soundness of Infinite Proof Trees

context
begin

private definition *num P xs* ≡ *LEAST n*. *list-all* (*Not o P*) (*stake n xs*) ∧ *P* (*xs*!!*n*)

private lemma *num*:
assumes *ev*: *ev* (λ*xs*. *P* (*shd xs*)) *xs*
defines *n* ≡ *num P xs*
shows
(*list-all* (*Not o P*) (*stake n xs*) ∧ *P* (*xs*!!*n*)) ∧
(∀ *m*. *list-all* (*Not o P*) (*stake m xs*) ∧ *P* (*xs*!!*m*) ⟶ *n* ≤ *m*)
⟨*proof*⟩ **lemma** *num-stl*[*simp*]:
assumes *ev* (λ*xs*. *P* (*shd xs*)) *xs* **and** ¬ *P* (*shd xs*)
shows *num P xs* = *Suc* (*num P* (*stl xs*))
⟨*proof*⟩

corecursive *decr0* **where**
decr0 Ord minSoFar js =
(*if* ¬ (*ev* (λ*js*. (*shd js*, *minSoFar*) ∈ *Ord* ∧ *shd js* ≠ *minSoFar*)) *js*
then undefined
else if ((*shd js*, *minSoFar*) ∈ *Ord* ∧ *shd js* ≠ *minSoFar*)

then shd js ## decr0 Ord (shd js) js
else decr0 Ord minSoFar (stl js))
⟨*proof*⟩

end

lemmas *well-order-on-defs* =
well-order-on-def linear-order-on-def partial-order-on-def
preorder-on-def trans-def antisym-def refl-on-def

lemma *sdrop-length-shift*[*simp*]:
sdrop (*length xs*) (*xs* @− *s*) = *s*
⟨*proof*⟩

lemma *ev-iff-shift*:
ev φ *xs* ⟷ (∃ *xl xs2*. *xs* = *xl* @− *xs2* ∧ φ *xs2*)
⟨*proof*⟩

locale *Infinite-Soundness* = *RuleSystem-Defs eff rules* **for**
eff :: *'rule* ⇒ *'sequent* ⇒ *'sequent fset* ⇒ *bool*
and *rules* :: *'rule stream*
+
fixes *structure* :: *'structure set*
and *sat* :: *'structure* ⇒ *'sequent* ⇒ *bool*
and δ :: *'sequent* ⇒ *'rule* ⇒ *'sequent* ⇒ (*'marker* × *bool* × *'marker*) *set*
and *Ord* :: *'ord rel*
and σ :: *'marker* × *'structure* ⇒ *'ord*
assumes
Ord: *well-order Ord*
and
descent:
⋀*r s sl S*.
⟦*r* ∈ *R*; *eff r s sl*; *S* ∈ *structure*; ¬ *sat S s*⟧
⟹
∃ *s' S'*.
s' |∈| *sl* ∧ *S'* ∈ *structure* ∧ ¬ *sat S' s'* ∧
(∀ *v v' b*.
(*v*,*b*,*v'*) ∈ δ *s r s'* ⟶
(σ(*v'*,*S'*), σ(*v*,*S*)) ∈ *Ord* ∧ (*b* ⟶ σ(*v'*,*S'*) ≠ σ(*v*,*S*)))

sublocale *Infinite-Soundness* < *Soundness* **where** *eff* = *eff* **and** *rules* = *rules*
and *structure* = *structure* **and** *sat* = *sat*
⟨*proof*⟩

context *Infinite-Soundness*
begin

coinductive *follow* :: *bool stream* ⇒ *'marker stream* ⇒ (*'sequent*,*'rule*)*step stream* ⇒ *bool* **where**
⟦*M'* = *shd Ms*; *s'* = *fst* (*shd steps*); (*M*,*b*,*M'*) ∈ δ *s r s'*; *follow bs Ms steps*⟧
⟹
follow (*SCons b bs*) (*SCons M Ms*) (*SCons* (*s*,*r*) *steps*)

definition *infDecr* :: *bool stream* ⇒ *bool* **where**
infDecr ≡ *alw* (*ev* (λ*bs*. *shd bs*))

definition *good* :: (*'sequent*,*'rule*)*dtree* ⇒ *bool* **where**
good t ≡ ∀ *steps*.
ipath t steps
⟶
ev (λ*steps'*. ∃ *bs Ms*. *follow bs Ms steps'* ∧ *infDecr bs*) *steps*

lemma *tfinite-good*: *tfinite t* ⟹ *good t*
⟨*proof*⟩

context
fixes *inv* :: *'sequent* × *'a* ⇒ *bool*
and *pred* :: *'sequent* × *'a* ⇒ *'rule* ⇒ *'sequent* × *'a* ⇒ *bool*
begin

primcorec *konigDtree* ::
(*'sequent*,*'rule*) *dtree* ⇒ *'a* ⇒ ((*'sequent*,*'rule*) *step* × *'a*) *stream* **where**
shd (*konigDtree t a*) = (*root t*, *a*)
|*stl* (*konigDtree t a*) =
(*let s* = *fst* (*root t*); *r* = *snd* (*root t*);
(*s'*,*a'*) = (*SOME* (*s'*,*a'*). *s'* |∈| *fimage* (*fst o root*) (*cont t*) ∧ *pred* (*s*,*a*) *r* (*s'*,*a'*) ∧ *inv* (*s'*,*a'*));
t' = (*SOME t'*. *t'* |∈| *cont t* ∧ *s'* = *fst* (*root t'*))
in konigDtree t' a'
)

lemma *stl-konigDtree*:
fixes *t* **defines** *s* ≡ *fst* (*root t*) **and** *r* ≡ *snd* (*root t*)
assumes *s'*: *s'* |∈| *fimage* (*fst o root*) (*cont t*) **and** *pred* (*s*,*a*) *r* (*s'*,*a''*) **and** *inv* (*s'*,*a''*)
shows ∃ *t' a'*. *t'* |∈| *cont t* ∧ *pred* (*s*,*a*) *r* (*fst* (*root t'*),*a'*) ∧ *inv* (*fst* (*root t'*),*a'*)
∧ *stl* (*konigDtree t a*) = *konigDtree t' a'*
⟨*proof*⟩

declare *konigDtree.simps*(*2*)[*simp del*]

lemma *konigDtree*:

assumes *1*: $\bigwedge r\ s\ sl\ a.$
$[\![r \in R;\ \mathit{eff}\ r\ s\ sl;\ \mathit{inv}\ (s,a)]\!] \Longrightarrow$
$\exists\, s'\ a'.\ s' \mid\in\mid sl \wedge \mathit{inv}\ (s',a') \wedge \mathit{pred}\ (s,a)\ r\ (s',a')$
and *2*: $\mathit{wf}\ t\ \mathit{inv}\ (\mathit{fst}\ (\mathit{root}\ t),\ a)$
shows
$\mathit{alw}\ (\lambda \mathit{stepas}.$
$\mathit{let}\ ((s,r),a) = \mathit{shd}\ \mathit{stepas};\ ((s',\text{-}),a') = \mathit{shd}\ (\mathit{stl}\ \mathit{stepas})\ \mathit{in}$
$\mathit{inv}\ (s,a) \wedge \mathit{pred}\ (s,a)\ r\ (s',a'))$
$(\mathit{konigDtree}\ t\ a)$
⟨*proof*⟩

lemma *konigDtree-ipath*:
assumes $\bigwedge r\ s\ sl\ a.$
$[\![r \in R;\ \mathit{eff}\ r\ s\ sl;\ \mathit{inv}\ (s,a)]\!] \Longrightarrow$
$\exists\, s'\ a'.\ s' \mid\in\mid sl \wedge \mathit{inv}\ (s',a') \wedge \mathit{pred}\ (s,a)\ r\ (s',a')$
and $\mathit{wf}\ t$ **and** $\mathit{inv}\ (\mathit{fst}\ (\mathit{root}\ t),\ a)$
shows $\mathit{ipath}\ t\ (\mathit{smap}\ \mathit{fst}\ (\mathit{konigDtree}\ t\ a))$
⟨*proof*⟩

end

lemma *follow-stl-smap-fst*[*simp*]:
$\mathit{follow}\ \mathit{bs}\ \mathit{Ms}\ (\mathit{smap}\ \mathit{fst}\ \mathit{stepSs}) \Longrightarrow$
$\mathit{follow}\ (\mathit{stl}\ \mathit{bs})\ (\mathit{stl}\ \mathit{Ms})\ (\mathit{smap}\ \mathit{fst}\ (\mathit{stl}\ \mathit{stepSs}))$
⟨*proof*⟩

lemma *epath-stl-smap-fst*[*simp*]:
$\mathit{epath}\ (\mathit{smap}\ \mathit{fst}\ \mathit{stepSs}) \Longrightarrow$
$\mathit{epath}\ (\mathit{smap}\ \mathit{fst}\ (\mathit{stl}\ \mathit{stepSs}))$
⟨*proof*⟩

lemma *infDecr-tl*[*simp*]: $\mathit{infDecr}\ \mathit{bs} \Longrightarrow \mathit{infDecr}\ (\mathit{stl}\ \mathit{bs})$
⟨*proof*⟩

fun *descent* **where** $\mathit{descent}\ (s,S)\ r\ (s',S') =$
$(\forall\, v\ v'\ b.$
$(v,b,v') \in \delta\ s\ r\ s' \longrightarrow$
$(\sigma(v',S'),\ \sigma(v,S)) \in \mathit{Ord} \wedge (b \longrightarrow \sigma(v',S') \neq \sigma(v,S)))$

lemma *descentE*[*elim*]:
assumes $\mathit{descent}\ (s,S)\ r\ (s',S')$ **and** $(v,b,v') \in \delta\ s\ r\ s'$
shows $(\sigma(v',S'),\ \sigma(v,S)) \in \mathit{Ord} \wedge (b \longrightarrow \sigma(v',S') \neq \sigma(v,S))$
⟨*proof*⟩

definition $\mathit{konigDown} \equiv \mathit{konigDtree}\ (\lambda(s,S).\ S \in \mathit{structure} \wedge \neg\ \mathit{sat}\ S\ s)\ \mathit{descent}$

lemma *konigDown*:
assumes $\mathit{wf}\ t$ **and** $S \in \mathit{structure}$ **and** $\neg\ \mathit{sat}\ S\ (\mathit{fst}\ (\mathit{root}\ t))$

shows
alw (λ*stepSs*. *let* ((*s*,*r*),*S*) = *shd stepSs*; ((*s*′,-),*S*′) = *shd* (*stl stepSs*) *in*
S ∈ *structure* ∧ ¬ *sat S s* ∧ *descent* (*s*,*S*) *r* (*s*′,*S*′))
(*konigDown t S*)
⟨*proof*⟩

lemma *konigDown-ipath*:
assumes *wf t* **and** *S* ∈ *structure* **and** ¬ *sat S* (*fst* (*root t*))
shows
ipath t (*smap fst* (*konigDown t S*))
⟨*proof*⟩

context
fixes *t S*
assumes *w*: *wf t* **and** *t*: *good t* **and** *S*: *S* ∈ *structure* ¬ *sat S* (*fst* (*root t*))
begin

lemma *alw-ev-Ord*:
obtains *ks* **where** *alw* (λ*ks*. (*shd* (*stl ks*), *shd ks*) ∈ *Ord*) *ks*
and *alw* (*ev* (λ*ks*. *shd* (*stl ks*) ≠ *shd ks*)) *ks*
⟨*proof*⟩

definition
ks ≡ *SOME ks*.
alw (λ*ks*. (*shd* (*stl ks*), *shd ks*) ∈ *Ord*) *ks* ∧
alw (*ev* (λ*ks*. *shd* (*stl ks*) ≠ *shd ks*)) *ks*

lemma *alw-ks*: *alw* (λ*ks*. (*shd* (*stl ks*), *shd ks*) ∈ *Ord*) *ks*
and *alw-ev-ks*: *alw* (*ev* (λ*ks*. *shd* (*stl ks*) ≠ *shd ks*)) *ks*
⟨*proof*⟩

abbreviation *decr* **where** *decr* ≡ *decr0 Ord*

lemmas *decr-simps* = *decr0.code*[*of Ord*]

context
fixes *js*
assumes *a*: *alw* (λ*js*. (*shd* (*stl js*), *shd js*) ∈ *Ord*) *js*
and *ae*: *alw* (*ev* (λ*js*. *shd* (*stl js*) ≠ *shd js*)) *js*
begin

lemma *decr-ev*:
assumes *m*: (*shd js*, *m*) ∈ *Ord*
shows *ev* (λ*js*. (*shd js*, *m*) ∈ *Ord* ∧ *shd js* ≠ *m*) *js*
(**is** *ev* (λ*js*. *?φ m js*) *js*)
⟨*proof*⟩

lemma *decr-simps-diff*[*simp*]:
assumes *m*: (*shd js*, *m*) ∈ *Ord*

and *shd js* $\neq$ *m*
shows *decr m js* = *shd js* ## *decr* (*shd js*) *js*
⟨*proof*⟩

lemma *decr-simps-eq*[*simp*]:
decr (*shd js*) *js* = *decr* (*shd js*) (*stl js*)
⟨*proof*⟩

end

lemma *stl-decr*:
assumes *a*: *alw* (λ*js*. (*shd* (*stl js*), *shd js*) ∈ *Ord*) *js*
and *ae*: *alw* (*ev* (λ*js*. *shd* (*stl js*) ≠ *shd js*)) *js*
and *m*: (*shd js*, *m*) ∈ *Ord*
shows
∃*js1 js2*. *js* = *js1* @− *js2* ∧ *set js1* ⊆ {*m*} ∧
(*shd js2*, *m*) ∈ *Ord* ∧ *shd js2* ≠ *m* ∧
shd (*decr m js*) = *shd js2* ∧ *stl* (*decr m js*) = *decr* (*shd js2*) *js2*
(**is** ∃*js1 js2*. *?φ js js1 js2*)
⟨*proof*⟩

corollary *stl-decr-shd*:
assumes *a*: *alw* (λ*js*. (*shd* (*stl js*), *shd js*) ∈ *Ord*) *js* **and**
ae: *alw* (*ev* (λ*js*. *shd* (*stl js*) ≠ *shd js*)) *js*
shows
∃*js1 js2*. *js* = *js1* @− *js2* ∧ *set js1* ⊆ {*shd js*} ∧
(*shd js2*, *shd js*) ∈ *Ord* ∧ *shd js2* ≠ *shd js* ∧
shd (*decr* (*shd js*) *js*) = *shd js2* ∧ *stl* (*decr* (*shd js*) *js*) = *decr* (*shd js2*) *js2*
⟨*proof*⟩

lemma *decr*:
assumes *a*: *alw* (λ*js*. (*shd* (*stl js*), *shd js*) ∈ *Ord*) *js* (**is** *?a js*)
and *ae*: *alw* (*ev* (λ*js*. *shd* (*stl js*) ≠ *shd js*)) *js* (**is** *?ae js*)
shows
alw (λ*js*. (*shd* (*stl js*), *shd js*) ∈ *Ord* ∧ *shd* (*stl js*) ≠ *shd js*) (*decr* (*shd js*) *js*)
(**is** *alw ?φ* -)
⟨*proof*⟩

lemma *alw-snth*:
assumes *alw* (λ*xs*. *P* (*shd* (*stl xs*)) (*shd xs*)) *xs*
shows *P* (*xs*!!(*Suc n*)) (*xs*!! *n*)
⟨*proof*⟩

lemma *F*: *False*
⟨*proof*⟩

end

theorem *infinite-soundness*:
assumes *wf t* **and** *good t* **and** $S \in$ *structure*
shows *sat S* (*fst* (*root t*))
⟨*proof*⟩

end

3 Soundness of Cyclic Proof Trees

datatype (*discs-sels*) ('*sequent*, '*rule*, '*link*) *ctree* =
Link '*link* |
cNode ('*sequent*,'*rule*) *step* ('*sequent*, '*rule*, '*link*) *ctree fset*

corecursive *treeOf* **where**
treeOf pointsTo ct =
(*if* ∃ *l l'*. *pointsTo l* = *Link l'*
(∗ *makes sense only if backward links point to normal nodes, not to backwards links*: ∗)
then undefined
else (*case ct of*
Link l ⇒ *treeOf pointsTo* (*pointsTo l*)
|*cNode step cts* ⇒ *Node step* (*fimage* (*treeOf pointsTo*) *cts*)
)
)
⟨*proof*⟩

declare *treeOf*.*code*[*simp*]

context *Infinite-Soundness*
begin

context
fixes *pointsTo* :: '*link* ⇒ ('*sequent*, '*rule*, '*link*)*ctree*
assumes *pointsTo*: ∀ *l l'*. *pointsTo l* ≠ *Link l'*
begin

function *seqOf* **where**
seqOf (*Link l*) = *seqOf* (*pointsTo l*)
|
seqOf (*cNode* (*s*,*r*) -) = *s*
⟨*proof*⟩
termination
⟨*proof*⟩

coinductive *cwf* **where**
Node[*intro*!]: *cwf* (*pointsTo l*) ⟹ *cwf* (*Link l*)
|
cNode[*intro*]:

⟦$r \in R$; $eff\ r\ s\ (fimage\ seqOf\ cts)$; $\bigwedge ct'.\ ct' \mid\in\mid cts \Longrightarrow cwf\ ct'$⟧
$\Longrightarrow$
$cwf\ (cNode\ (s,r)\ cts)$

definition $cgood\ ct \equiv good\ (treeOf\ pointsTo\ ct)$

lemma *cwf-Link*: $cwf\ (Link\ l) \longleftrightarrow cwf\ (pointsTo\ l)$
⟨*proof*⟩

lemma *cwf-cNode-seqOf*:
$cwf\ (cNode\ (s,\ r)\ cts) \Longrightarrow eff\ r\ s\ (fimage\ seqOf\ cts)$
⟨*proof*⟩

lemma *treeOf-seqOf*[*simp*]:
$fst \circ root \circ treeOf\ pointsTo = seqOf$
⟨*proof*⟩

lemma *wf-treeOf*:
assumes *cwf ct*
shows $wf\ (treeOf\ pointsTo\ ct)$
⟨*proof*⟩

theorem *cyclic-soundness*:
assumes *cwf ct* **and** *cgood ct* **and** $S \in structure$
shows $sat\ S\ (seqOf\ ct)$
⟨*proof*⟩

end

end

4 Appendix: The definition of treeOf under more flexible assumptions about pointsTo

definition *rels* **where**
$rels\ pointsTo \equiv \{((pointsTo,\ pointsTo\ l'),\ (pointsTo,\ Link\ l')) \mid l'.\ True\}$

definition *rel* :: $(('link \Rightarrow ('sequent,\ 'rule,\ 'link)\ ctree) \times ('sequent,\ 'rule,\ 'link)\ ctree)\ rel$ **where**
$rel \equiv UNION\ \{pointsTo\ .\ wf\ \{(l,l').\ pointsTo\ l' = Link\ l\}\}\ rels$

lemma *wf-rels*[*simp*]:
assumes $wf\ \{(l,l').\ (pointsTo :: 'link \Rightarrow ('sequent,\ 'rule,\ 'link)ctree)\ l' = Link\ l\}$
(**is** *wf ?w*)
shows $wf\ (rels\ pointsTo)$ ⟨*proof*⟩

lemma *rel*: *wf rel*

```
  ⟨proof⟩

corecursive treeOf' where
  treeOf' pointsTo ct =
   (if ¬ wf {(l',l). pointsTo l = Link l'}
    (* makes sense only if backward links point to normal nodes, not to backwards
links: *)
     then undefined
     else (case ct of
             Link l ⇒ treeOf' pointsTo (pointsTo l)
            |cNode step cts ⇒ Node step (fimage (treeOf' pointsTo) cts)
          )
   )
  ⟨proof⟩
```