Abstract Soundness

Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel

March 17, 2025

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*:
$\bigwedge r\ s\ sl.$
$[\![r \in R;\ eff\ r\ s\ sl;\ \bigwedge s'.\ s'\ |{\in}|\ sl \Longrightarrow \forall S \in structure.\ sat\ S\ s']\!]$
$\Longrightarrow$
$\forall S \in structure.\ sat\ S\ s$
begin

abbreviation $ssat\ s \equiv \forall S \in structure.\ sat\ S\ s$

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 $num\ P\ xs \equiv LEAST\ n.\ list\text{-}all\ (Not\ o\ P)\ (stake\ n\ xs) \wedge P\ (xs!!n)$

private lemma *num*:
assumes *ev*: *ev* (*λxs. P* (*shd xs*)) *xs*
defines $n \equiv num\ P\ xs$
shows
$(list\text{-}all\ (Not\ o\ P)\ (stake\ n\ xs) \wedge P\ (xs!!n)) \wedge$
$(\forall m.\ list\text{-}all\ (Not\ o\ P)\ (stake\ m\ xs) \wedge P\ (xs!!m) \longrightarrow n \le m)$
⟨*proof*⟩ **lemma** *num-stl*[*simp*]:
assumes *ev* (*λxs. 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 =
$(if\ \neg\ (ev\ (\lambda js.\ (shd\ js,\ minSoFar) \in Ord \wedge shd\ js \neq minSoFar))\ js$
then undefined
$else\ if\ ((shd\ js,\ minSoFar) \in Ord \wedge shd\ js \neq minSoFar)$

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* $\longleftrightarrow$ ($\exists$ *xl xs2*. *xs* = *xl* @− *xs2* $\wedge$ 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:
$\bigwedge$*r s sl S*.
⟦*r* ∈ *R*; *eff r s sl*; *S* ∈ *structure*; ¬ *sat S s*⟧
$\Longrightarrow$
∃ *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* $\Rightarrow$ *'marker stream* $\Rightarrow$ *('sequent,'rule)step stream* $\Rightarrow$ *bool* **where**

$[\![M' = shd\ Ms;\ s' = fst\ (shd\ steps);\ (M,b,M') \in \delta\ s\ r\ s';\ follow\ bs\ Ms\ steps]\!]$

$\Longrightarrow$

follow (SCons b bs) (SCons M Ms) (SCons (s,r) steps)

definition *infDecr* :: *bool stream* $\Rightarrow$ *bool* **where**

$infDecr \equiv alw\ (ev\ (\lambda bs.\ shd\ bs))$

definition *good* :: *('sequent,'rule)dtree* $\Rightarrow$ *bool* **where**

$good\ t \equiv \forall steps.$

ipath t steps

$\longrightarrow$

$ev\ (\lambda steps'.\ \exists bs\ Ms.\ follow\ bs\ Ms\ steps' \wedge infDecr\ bs)\ steps$

lemma *tfinite-good*: *tfinite t* $\Longrightarrow$ *good t*

$\langle proof\rangle$

context

fixes *inv* :: *'sequent* $\times$ *'a* $\Rightarrow$ *bool*

and *pred* :: *'sequent* $\times$ *'a* $\Rightarrow$ *'rule* $\Rightarrow$ *'sequent* $\times$ *'a* $\Rightarrow$ *bool*

begin

primcorec *konigDtree* ::

('sequent,'rule) dtree $\Rightarrow$ *'a* $\Rightarrow$ *(('sequent,'rule) step* $\times$ *'a) stream* **where**

$shd\ (konigDtree\ t\ a) = (root\ t,\ a)$

$|stl\ (konigDtree\ t\ a) =$

$(let\ s = fst\ (root\ t);\ r = snd\ (root\ t);$

$(s',a') = (SOME\ (s',a').\ s'\ |\in|\ fimage\ (fst\ o\ root)\ (cont\ t) \wedge pred\ (s,a)\ r\ (s',a') \wedge inv\ (s',a'));$

$t' = (SOME\ t'.\ t'\ |\in|\ cont\ t \wedge s' = fst\ (root\ t'))$

$in\ konigDtree\ t'\ a'$

)

lemma *stl-konigDtree*:

fixes *t* **defines** $s \equiv fst\ (root\ t)$ **and** $r \equiv snd\ (root\ t)$

assumes s': $s'\ |\in|\ fimage\ (fst\ o\ root)\ (cont\ t)$ **and** $pred\ (s,a)\ r\ (s',a'')$ **and** $inv\ (s',a'')$

shows $\exists t'\ a'.\ t'\ |\in|\ cont\ t \wedge pred\ (s,a)\ r\ (fst\ (root\ t'),a') \wedge inv\ (fst\ (root\ t'),a') \wedge stl\ (konigDtree\ t\ a) = konigDtree\ t'\ a'$

$\langle proof\rangle$

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

lemma *konigDtree*:

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

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

end

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

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

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

fun *descent* **where** *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 *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 *konigDown* $\equiv$ *konigDtree* (λ(*s*,*S*). *S* $\in$ *structure* $\wedge$ $\neg$ *sat S s*) *descent*

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

shows
alw (*λstepSs. 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* (*λks.* (*shd* (*stl ks*), *shd ks*) ∈ *Ord*) *ks*
and *alw* (*ev* (*λks. shd* (*stl ks*) ≠ *shd ks*)) *ks*
⟨*proof*⟩

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

lemma *alw-ks*: *alw* (*λks.* (*shd* (*stl ks*), *shd ks*) ∈ *Ord*) *ks*
and *alw-ev-ks*: *alw* (*ev* (*λks. 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* (*λjs.* (*shd* (*stl js*), *shd js*) ∈ *Ord*) *js*
and *ae*: *alw* (*ev* (*λjs. shd* (*stl js*) ≠ *shd js*)) *js*
begin

lemma *decr-ev*:
assumes *m*: (*shd js*, *m*) ∈ *Ord*
shows *ev* (*λjs.* (*shd js*, *m*) ∈ *Ord* ∧ *shd js* ≠ *m*) *js*
(**is** *ev* (*λjs. ?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*) $\in$ *Ord*) *js*
and *ae*: *alw* (*ev* (λ*js*. *shd* (*stl js*) $\neq$ *shd js*)) *js*
and *m*: (*shd js*, *m*) $\in$ *Ord*
shows
$\exists$*js1 js2*. *js* = *js1* @− *js2* $\wedge$ *set js1* $\subseteq$ {*m*} $\wedge$
(*shd js2*, *m*) $\in$ *Ord* $\wedge$ *shd js2* $\neq$ *m* $\wedge$
shd (*decr m js*) = *shd js2* $\wedge$ *stl* (*decr m js*) = *decr* (*shd js2*) *js2*
(**is** $\exists$*js1 js2*. *?φ js js1 js2*)
⟨*proof*⟩

corollary *stl-decr-shd*:
assumes *a*: *alw* (λ*js*. (*shd* (*stl js*), *shd js*) $\in$ *Ord*) *js* **and**
ae: *alw* (*ev* (λ*js*. *shd* (*stl js*) $\neq$ *shd js*)) *js*
shows
$\exists$*js1 js2*. *js* = *js1* @− *js2* $\wedge$ *set js1* $\subseteq$ {*shd js*} $\wedge$
(*shd js2*, *shd js*) $\in$ *Ord* $\wedge$ *shd js2* $\neq$ *shd js* $\wedge$
shd (*decr* (*shd js*) *js*) = *shd js2* $\wedge$ *stl* (*decr* (*shd js*) *js*) = *decr* (*shd js2*) *js2*
⟨*proof*⟩

lemma *decr*:
assumes *a*: *alw* (λ*js*. (*shd* (*stl js*), *shd js*) $\in$ *Ord*) *js* (**is** *?a js*)
and *ae*: *alw* (*ev* (λ*js*. *shd* (*stl js*) $\neq$ *shd js*)) *js* (**is** *?ae js*)
shows
alw (λ*js*. (*shd* (*stl js*), *shd js*) $\in$ *Ord* $\wedge$ *shd* (*stl js*) $\neq$ *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* $\exists$ *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*: $\forall$ *l l'*. *pointsTo l* $\neq$ *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*]:

$\llbracket r \in R;\ \mathit{eff}\ r\ s\ (\mathit{fimage}\ \mathit{seqOf}\ \mathit{cts});\ \bigwedge ct'.\ ct'\ |\!\in\!|\ \mathit{cts} \Longrightarrow \mathit{cwf}\ ct' \rrbracket$
$\Longrightarrow$
cwf (*cNode* (*s*,*r*) *cts*)

definition *cgood ct* ≡ *good* (*treeOf pointsTo ct*)

lemma *cwf-Link*: *cwf* (*Link l*) ⟷ *cwf* (*pointsTo l*)
⟨*proof*⟩

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

lemma *treeOf-seqOf*[*simp*]:
fst ∘ *root* ∘ *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* ∈ *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 ≡ {((*pointsTo*, *pointsTo l'*), (*pointsTo*, *Link l'*)) | *l'*. *True*}

definition *rel* :: ((*'link* ⇒ (*'sequent*, *'rule*, *'link*) *ctree*) × (*'sequent*, *'rule*, *'link*) *ctree*) *rel* **where**
rel ≡ ⋃ (*rels* ‘ {*pointsTo*. *wf* {(*l*, *l'*). *pointsTo l'* = *Link l*}})

lemma *wf-rels*[*simp*]:
assumes *wf* {(*l*,*l'*). (*pointsTo* :: *'link* ⇒ (*'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*⟩