

Abstract Soundness

Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel

February 23, 2021

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	1
2	Soundness of Infinite Proof Trees	2
3	Soundness of Cyclic Proof Trees	8
4	Appendix: The definition of treeOf under more flexible assumptions about pointsTo	9

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  $\Rightarrow$  'sequent  $\Rightarrow$  bool
assumes local-soundness:
   $\bigwedge r s sl.$ 
   $\llbracket r \in R; \text{eff } r s sl; \bigwedge s'. s' \in | sl \implies \forall S \in \text{structure. sat } S s \rrbracket$ 
   $\implies$ 
   $\forall S \in \text{structure. sat } S s$ 
begin

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

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

```

```

private lemma num:
  assumes ev: ev ( $\lambda xs. P (\text{shd } xs)$ ) xs
  defines n  $\equiv \text{num } P xs$ 
  shows
    ( $\text{list-all } (\text{Not } o P) (\text{stake } n xs) \wedge P (xs!!n) \wedge$ 
 $(\forall m. \text{list-all } (\text{Not } o P) (\text{stake } m xs) \wedge P (xs!!m) \longrightarrow n \leq m)$ )
  <proof> lemma num-stl[simp]:
  assumes ev ( $\lambda xs. P (\text{shd } xs)$ ) xs and  $\neg P (\text{shd } xs)$ 
  shows num P xs = Suc (num P (stl xs))
  <proof>

```

```

corecursive decr0 where
  decr0 Ord minSoFar js =
    (if  $\neg (ev (\lambda js. (\text{shd } js, \text{minSoFar}) \in \text{Ord} \wedge \text{shd } js \neq \text{minSoFar}))$  js
     then undefined
     else if  $((\text{shd } js, \text{minSoFar}) \in \text{Ord} \wedge \text{shd } js \neq \text{minSoFar})$ 

```

```

    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  $\varphi$  xs  $\longleftrightarrow$  ( $\exists$  xl xs2. xs = xl @- xs2  $\wedge$   $\varphi$  xs2)
  ⟨proof⟩

locale Infinite-Soundness = RuleSystem-Defs eff rules for
  eff :: 'rule  $\Rightarrow$  'sequent  $\Rightarrow$  'sequent fset  $\Rightarrow$  bool
  and rules :: 'rule stream
  +
  fixes structure :: 'structure set
  and sat :: 'structure  $\Rightarrow$  'sequent  $\Rightarrow$  bool
  and  $\delta$  :: 'sequent  $\Rightarrow$  'rule  $\Rightarrow$  'sequent  $\Rightarrow$  ('marker  $\times$  bool  $\times$  'marker) set
  and Ord :: 'ord rel
  and  $\sigma$  :: 'marker  $\times$  'structure  $\Rightarrow$  'ord
  assumes
    Ord: well-order Ord
  and
    descent:
       $\bigwedge$  r s sl S.
         $\llbracket r \in R; \text{eff } r \text{ s sl}; S \in \text{structure}; \neg \text{sat } S \text{ s} \rrbracket$ 
           $\Longrightarrow$ 
           $\exists$  s' S'.
            s'  $\in$  sl  $\wedge$  S'  $\in$  structure  $\wedge$   $\neg \text{sat } S' \text{ s}' \wedge$ 
            ( $\forall$  v v' b.
              (v,b,v')  $\in$   $\delta$  s r s'  $\longrightarrow$ 
              ( $\sigma(v',S'), \sigma(v,S)$ )  $\in$  Ord  $\wedge$  (b  $\longrightarrow$   $\sigma(v',S') \neq \sigma(v,S)$ ))

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*
 $\llbracket M' = \text{shd } Ms; s' = \text{fst } (\text{shd } \text{steps}); (M, b, M') \in \delta \text{ } s \text{ } r \text{ } s'; \text{follow } bs \text{ } Ms \text{ steps} \rrbracket$
 \Longrightarrow
follow (*SCons* *b* *bs*) (*SCons* *M* *Ms*) (*SCons* (*s,r*) *steps*)

definition *infDecr* :: *bool stream* \Rightarrow *bool where*
infDecr \equiv *alw* (*ev* ($\lambda bs. \text{shd } bs$))

definition *good* :: (*'sequent,'rule*)*dtree* \Rightarrow *bool where*
good *t* \equiv \forall *steps*.
ipath *t* *steps*
 \longrightarrow
ev ($\lambda \text{steps}'. \exists bs \text{ } Ms. \text{follow } bs \text{ } Ms \text{ steps}' \wedge \text{infDecr } bs$) *steps*

lemma *tfinite-good*: *tfinite* *t* \Longrightarrow *good* *t*
 $\langle \text{proof} \rangle$

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*
 $\text{shd } (\text{konigDtree } t \text{ } a) = (\text{root } t, a)$
 $\text{stl } (\text{konigDtree } t \text{ } a) =$
 $(\text{let } s = \text{fst } (\text{root } t); r = \text{snd } (\text{root } t);$
 $(s', a') = (\text{SOME } (s', a'). s' \in | \text{fimage } (\text{fst } o \text{ root}) (\text{cont } t) \wedge \text{pred } (s, a) \text{ } r (s', a'))$
 $\wedge \text{inv } (s', a'));$
 $t' = (\text{SOME } t'. t' \in | \text{cont } t \wedge s' = \text{fst } (\text{root } t'))$
 $\text{in } \text{konigDtree } t' \text{ } a'$
 $)$

lemma *stl-konigDtree*:

fixes *t* **defines** $s \equiv \text{fst } (\text{root } t)$ **and** $r \equiv \text{snd } (\text{root } t)$
assumes $s': s' \in | \text{fimage } (\text{fst } o \text{ root}) (\text{cont } t)$ **and** $\text{pred } (s, a) \text{ } r (s', a')$ **and** *inv*
 (s', a')
shows $\exists t' \text{ } a'. t' \in | \text{cont } t \wedge \text{pred } (s, a) \text{ } r (\text{fst } (\text{root } t'), a') \wedge \text{inv } (\text{fst } (\text{root } t'), a')$
 $\wedge \text{stl } (\text{konigDtree } t \text{ } a) = \text{konigDtree } t' \text{ } a'$
 $\langle \text{proof} \rangle$

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

lemma *konigDtree*:

assumes 1: $\bigwedge r s sl a.$
 $\llbracket r \in R; \text{eff } r s sl; \text{inv } (s,a) \rrbracket \implies$
 $\exists s' a'. s' \mid \in \mid sl \wedge \text{inv } (s',a') \wedge \text{pred } (s,a) r (s',a')$
and 2: $wf t \text{ inv } (\text{fst } (\text{root } t), a)$
shows
 $alw (\lambda \text{stepas}.$
 $\text{let } ((s,r),a) = \text{shd } \text{stepas}; ((s',-),a') = \text{shd } (\text{stl } \text{stepas}) \text{ in}$
 $\text{inv } (s,a) \wedge \text{pred } (s,a) r (s',a')$
 $(\text{konigDtree } t a)$
 $\langle \text{proof} \rangle$

lemma *konigDtree-ipath*:
assumes $\bigwedge r s sl a.$
 $\llbracket r \in R; \text{eff } r s sl; \text{inv } (s,a) \rrbracket \implies$
 $\exists s' a'. s' \mid \in \mid sl \wedge \text{inv } (s',a') \wedge \text{pred } (s,a) r (s',a')$
and $wf t$ **and** $\text{inv } (\text{fst } (\text{root } t), a)$
shows $\text{ipath } t (\text{smap } \text{fst } (\text{konigDtree } t a))$
 $\langle \text{proof} \rangle$

end

lemma *follow-stl-smap-fst[simp]*:
 $\text{follow } bs Ms (\text{smap } \text{fst } \text{stepSs}) \implies$
 $\text{follow } (\text{stl } bs) (\text{stl } Ms) (\text{smap } \text{fst } (\text{stl } \text{stepSs}))$
 $\langle \text{proof} \rangle$

lemma *epath-stl-smap-fst[simp]*:
 $\text{epath } (\text{smap } \text{fst } \text{stepSs}) \implies$
 $\text{epath } (\text{smap } \text{fst } (\text{stl } \text{stepSs}))$
 $\langle \text{proof} \rangle$

lemma *infDecr-tl[simp]*: $\text{infDecr } bs \implies \text{infDecr } (\text{stl } bs)$
 $\langle \text{proof} \rangle$

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

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

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

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

shows

$alw (\lambda stepSs. let ((s,r),S) = shd\ stepSs; ((s',-),S') = shd\ (stl\ stepSs)\ in$
 $S \in structure \wedge \neg sat\ S\ s \wedge descent\ (s,S)\ r\ (s',S')$
 $(konigDown\ t\ S)$

$\langle proof \rangle$

lemma *konigDown-ipath*:

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

shows

$ipath\ t\ (smap\ fst\ (konigDown\ t\ S))$

$\langle proof \rangle$

context

fixes $t\ S$

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

begin

lemma *alw-ev-Ord*:

obtains ks **where** $alw\ (\lambda ks. (shd\ (stl\ ks), shd\ ks) \in Ord)$ ks

and $alw\ (ev\ (\lambda ks. shd\ (stl\ ks) \neq shd\ ks))\ ks$

$\langle proof \rangle$

definition

$ks \equiv SOME\ ks.$

$alw\ (\lambda ks. (shd\ (stl\ ks), shd\ ks) \in Ord)\ ks \wedge$

$alw\ (ev\ (\lambda ks. shd\ (stl\ ks) \neq shd\ ks))\ ks$

lemma *alw-ks*: $alw\ (\lambda ks. (shd\ (stl\ ks), shd\ ks) \in Ord)\ ks$

and *alw-ev-ks*: $alw\ (ev\ (\lambda ks. shd\ (stl\ ks) \neq shd\ ks))\ ks$

$\langle proof \rangle$

abbreviation *decr* **where** $decr \equiv decr0\ Ord$

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

context

fixes js

assumes $a: alw\ (\lambda js. (shd\ (stl\ js), shd\ js) \in Ord)\ js$

and $ae: alw\ (ev\ (\lambda js. shd\ (stl\ js) \neq shd\ js))\ js$

begin

lemma *decr-ev*:

assumes $m: (shd\ js, m) \in Ord$

shows $ev\ (\lambda js. (shd\ js, m) \in Ord \wedge shd\ js \neq m)\ js$

(**is** $ev\ (\lambda js. ?\varphi\ m\ js)\ js$)

$\langle proof \rangle$

lemma *decr-simps-diff[simp]*:

assumes $m: (shd\ js, m) \in Ord$

and $shd\ js \neq m$
shows $decr\ m\ js = shd\ js \#\# decr\ (shd\ js)\ js$
 $\langle proof \rangle$

lemma *decr-simps-eq[simp]*:
 $decr\ (shd\ js)\ js = decr\ (shd\ js)\ (stl\ js)$
 $\langle proof \rangle$

end

lemma *stl-decr*:
assumes $a: alw\ (\lambda js. (shd\ (stl\ js), shd\ js) \in Ord)\ js$
and $ae: alw\ (ev\ (\lambda js. shd\ (stl\ js) \neq shd\ js))\ js$
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 $\exists js1\ js2. ?\varphi\ js\ js1\ js2)$
 $\langle proof \rangle$

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

lemma *decr*:
assumes $a: alw\ (\lambda js. (shd\ (stl\ js), shd\ js) \in Ord)\ js$ **(is** $?a\ js)$
and $ae: alw\ (ev\ (\lambda js. shd\ (stl\ js) \neq shd\ js))\ js$ **(is** $?ae\ js)$
shows
 $alw\ (\lambda js. (shd\ (stl\ js), shd\ js) \in Ord \wedge shd\ (stl\ js) \neq shd\ js)\ (decr\ (shd\ js)\ js)$
(is $alw\ ?\varphi\ -)$
 $\langle proof \rangle$

lemma *alw-snth*:
assumes $alw\ (\lambda xs. P\ (shd\ (stl\ xs))\ (shd\ xs))\ xs$
shows $P\ (xs!!(Suc\ n))\ (xs!!\ n)$
 $\langle proof \rangle$

lemma *F: False*
 $\langle proof \rangle$

end

theorem *infinite-soundness*:
assumes *wf t and good t and $S \in \text{structure}$*
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 $\exists l l'. \text{pointsTo } l = \text{Link } l'$

— makes sense only if backward links point to normal nodes, not to backwards

links:

then *undefined*

else (case *ct* of

Link *l* $\Rightarrow \text{treeOf } \text{pointsTo } (\text{pointsTo } l)$

| *cNode* *step* *cts* $\Rightarrow \text{Node } \text{step } (\text{fimage } (\text{treeOf } \text{pointsTo}) \text{ cts})$

)

)

<proof>

declare *treeOf.code[simp]*

context *Infinite-Soundness*

begin

context

fixes *pointsTo* :: '*link*' \Rightarrow ('*sequent*', '*rule*', '*link*')*ctree*

assumes *pointsTo*: $\forall l l'. \text{pointsTo } l \neq \text{Link } l'$

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*) \Longrightarrow *cwf* (*Link* *l*)

|

cNode[*intro*]:

$\llbracket r \in R; \text{eff } r \text{ s } (\text{fimage seqOf } \text{cts}); \bigwedge \text{ct}'. \text{ct}' \in \text{cts} \implies \text{cwf } \text{ct}' \rrbracket$
 \implies
 $\text{cwf } (\text{cNode } (s, r) \text{ cts})$

definition $\text{cgood } \text{ct} \equiv \text{good } (\text{treeOf } \text{pointsTo } \text{ct})$

lemma *cwf-Link*: $\text{cwf } (\text{Link } l) \longleftrightarrow \text{cwf } (\text{pointsTo } l)$
 $\langle \text{proof} \rangle$

lemma *cwf-cNode-seqOf*:
 $\text{cwf } (\text{cNode } (s, r) \text{ cts}) \implies \text{eff } r \text{ s } (\text{fimage seqOf } \text{cts})$
 $\langle \text{proof} \rangle$

lemma *treeOf-seqOf[simp]*:
 $\text{fst} \circ \text{root} \circ \text{treeOf } \text{pointsTo} = \text{seqOf}$
 $\langle \text{proof} \rangle$

lemma *wf-treeOf*:
assumes $\text{cwf } \text{ct}$
shows $\text{wf } (\text{treeOf } \text{pointsTo } \text{ct})$
 $\langle \text{proof} \rangle$

theorem *cyclic-soundness*:
assumes $\text{cwf } \text{ct}$ **and** $\text{cgood } \text{ct}$ **and** $S \in \text{structure}$
shows $\text{sat } S (\text{seqOf } \text{ct})$
 $\langle \text{proof} \rangle$

end

end

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

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

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

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

lemma *rel*: $\text{wf } \text{rel}$
 $\langle \text{proof} \rangle$

corecursive *treeOf'* **where**

treeOf' *pointsTo* *ct* =

(if $\neg wf \{(l',l). \text{pointsTo } l = \text{Link } l'\}$

— makes sense only if backward links point to normal nodes, not to backwards

links:

then undefined

else (case ct of

Link l \Rightarrow *treeOf'* *pointsTo* (*pointsTo l*)

| *cNode step cts* \Rightarrow *Node step* (*fimage* (*treeOf'* *pointsTo*) *cts*)

)

)

<proof>