Separata: Isabelle tactics for Separation Algebra

By Zhé Hóu, David Sanán, Alwen Tiu, Rajeev Goré, Ranald Clouston

May 26, 2024

Abstract

We bring the labelled sequent calculus LS_{PASL} for propositional abstract separation logic to Isabelle. The tactics given here are directly applied on an extension of the separation algebra in the AFP. In addition to the cancellative separation algebra, we further consider some useful properties in the heap model of separation logic, such as indivisible unit, disjointness, and cross-split. The tactics are essentially a proof search procedure for the calculus LS_{PASL}. We wrap the tactics in an Isabelle method called separata, and give a few examples of separation logic formulae which are provable by separata.

Contents

theory *Separata*
imports *Main Separation-Algebra.Separation-Algebra HOL−Eisbach.Eisbach-Tools HOL−Library.Multiset*
begin

The tactics in this file are a simple proof search procedure based on the labelled sequent calculus LS_PASL for Propositional Abstract Separation Logic in Zhe's PhD thesis.

We define a class which is an extension to cancellative_sep_algebra with other useful properties in separation algebra, including: indivisible unit, disjointness, and cross-split. We also add a property about the (reverse) distributivity of the disjointness.

class *heap-sep-algebra* = *cancellative-sep-algebra* +

assumes *sep-add-ind-unit*: $[\![x + y = 0;\ x\ \#\#\ y]\!] \Longrightarrow x = 0$
assumes *sep-add-disj*: $x\#\#x \Longrightarrow x = 0$
assumes *sep-add-cross-split*:
$[\![a + b = w;\ c + d = w;\ a\ \#\#\ b;\ c\ \#\#\ d]\!] \Longrightarrow$
$\exists\ e\ f\ g\ h.\ e + f = a \wedge g + h = b \wedge e + g = c \wedge f + h = d \wedge$
$e\ \#\#\ f \wedge g\ \#\#\ h \wedge e\ \#\#\ g \wedge f\ \#\#\ h$
assumes *disj-dstri*: $[\![x\ \#\#\ y;\ y\ \#\#\ z;\ x\ \#\#\ z]\!] \Longrightarrow x\ \#\#\ (y + z)$
begin

1 Lemmas about the labelled sequent calculus.

An abbreviation of the + and ## operators in Separation_Algebra.thy. This notion is closer to the ternary relational atoms used in the literature. This will be the main data structure which our labelled sequent calculus works on.

definition *tern-rel*:: $'a \Rightarrow\ 'a \Rightarrow\ 'a \Rightarrow bool$ ((-,-▷-) *25*) **where**
tern-rel $a\ b\ c \equiv a\ \#\#\ b \wedge a + b = c$

lemma *exist-comb*: $x\ \#\#\ y \Longrightarrow \exists\, z.\ (x,y \triangleright z)$
⟨*proof*⟩

lemma *disj-comb*:
assumes *a1*: $(x,y \triangleright z)$
assumes *a2*: $x\ \#\#\ w$
assumes *a3*: $y\ \#\#\ w$
shows $z\ \#\#\ w$
⟨*proof*⟩

The following lemmas corresponds to inference rules in LS_PASL. Thus these lemmas prove the soundness of LS_PASL. We also show the invertibility of those rules.

lemma (**in** −) *lspasl-id*:
$Gamma \wedge (A\ h) \Longrightarrow (A\ h) \vee Delta$
⟨*proof*⟩

lemma (**in** −) *lspasl-botl*:
$Gamma \wedge (sep\text{-}false\ h) \Longrightarrow Delta$
⟨*proof*⟩

lemma (**in** −) *lspasl-topr*:
$gamma \Longrightarrow (sep\text{-}true\ h) \vee Delta$
⟨*proof*⟩

lemma *lspasl-empl*:
$Gamma \wedge (h = 0) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (sep\text{-}empty\ h) \longrightarrow Delta$
⟨*proof*⟩

lemma *lspasl-empl-inv*:
$Gamma \wedge (sep\text{-}empty\ h) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (h = 0) \longrightarrow Delta$
⟨*proof*⟩

The following two lemmas are the same as applying simp add: sep_empty_def.

lemma *lspasl-empl-der*: $sep\text{-}empty\ h \Longrightarrow h = 0$
⟨*proof*⟩

lemma *lspasl-empl-eq*: $(sep\text{-}empty\ h) = (h = 0)$
⟨*proof*⟩

lemma *lspasl-empr*:
$Gamma \longrightarrow (sep\text{-}empty\ 0) \vee Delta$
⟨*proof*⟩

end

lemma *lspasl-notl*:
$Gamma \longrightarrow (A\ h) \vee Delta \Longrightarrow$
$Gamma \wedge ((not\ A)\ h) \longrightarrow Delta$
⟨*proof*⟩

lemma *lspasl-notl-inv*:
$Gamma \wedge ((not\ A)\ h) \longrightarrow Delta \Longrightarrow$
$Gamma \longrightarrow (A\ h) \vee Delta$
⟨*proof*⟩

lemma *lspasl-notr*:
$Gamma \wedge (A\ h) \longrightarrow Delta \Longrightarrow$
$Gamma \longrightarrow ((not\ A)\ h) \vee Delta$
⟨*proof*⟩

lemma *lspasl-notr-inv*:
$Gamma \longrightarrow ((not\ A)\ h) \vee Delta \Longrightarrow$
$Gamma \wedge (A\ h) \longrightarrow Delta$
⟨*proof*⟩

lemma *lspasl-andl*:
$Gamma \wedge (A\ h) \wedge (B\ h) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge ((A\ and\ B)\ h) \longrightarrow Delta$
⟨*proof*⟩

lemma *lspasl-andl-inv*:
$Gamma \wedge ((A\ and\ B)\ h) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (A\ h) \wedge (B\ h) \longrightarrow Delta$
⟨*proof*⟩

lemma *lspasl-andr*:
$[\![Gamma \longrightarrow (A\ h) \vee Delta;\ Gamma \longrightarrow (B\ h) \vee Delta]\!] \Longrightarrow$
$Gamma \longrightarrow ((A\ and\ B)\ h) \vee Delta$
⟨*proof*⟩

lemma *lspasl-andr-inv*:
$Gamma \longrightarrow ((A\ and\ B)\ h) \vee Delta \Longrightarrow$
$(Gamma \longrightarrow (A\ h) \vee Delta) \wedge (Gamma \longrightarrow (B\ h) \vee Delta)$
⟨*proof*⟩

lemma *lspasl-orl*:
$[\![Gamma \wedge (A\ h) \longrightarrow Delta;\ Gamma \wedge (B\ h) \longrightarrow Delta]\!] \Longrightarrow$
$Gamma \wedge (A\ or\ B)\ h \longrightarrow Delta$
⟨*proof*⟩

lemma *lspasl-orl-inv*:
$Gamma \wedge (A\ or\ B)\ h \longrightarrow Delta \Longrightarrow$
$(Gamma \wedge (A\ h) \longrightarrow Delta) \wedge (Gamma \wedge (B\ h) \longrightarrow Delta)$
⟨*proof*⟩

lemma *lspasl-orr*:
$Gamma \longrightarrow (A\ h) \vee (B\ h) \vee Delta \Longrightarrow$
$Gamma \longrightarrow ((A\ or\ B)\ h) \vee Delta$
⟨*proof*⟩

lemma *lspasl-orr-inv*:
$Gamma \longrightarrow ((A\ or\ B)\ h) \vee Delta \Longrightarrow$
$Gamma \longrightarrow (A\ h) \vee (B\ h) \vee Delta$
⟨*proof*⟩

lemma *lspasl-impl*:
$[\![Gamma \longrightarrow (A\ h) \vee Delta;\ Gamma \wedge (B\ h) \longrightarrow Delta]\!] \Longrightarrow$
$Gamma \wedge ((A\ imp\ B)\ h) \longrightarrow Delta$
⟨*proof*⟩

lemma *lspasl-impl-inv*:
$Gamma \wedge ((A\ imp\ B)\ h) \longrightarrow Delta \Longrightarrow$
$(Gamma \longrightarrow (A\ h) \vee Delta) \wedge (Gamma \wedge (B\ h) \longrightarrow Delta)$
⟨*proof*⟩

lemma *lspasl-impr*:
$Gamma \wedge (A\ h) \longrightarrow (B\ h) \vee Delta \Longrightarrow$
$Gamma \longrightarrow ((A\ imp\ B)\ h) \vee Delta$
⟨*proof*⟩

lemma *lspasl-impr-inv*:
$Gamma \longrightarrow ((A\ imp\ B)\ h) \vee Delta \Longrightarrow$
$Gamma \wedge (A\ h) \longrightarrow (B\ h) \vee Delta$
⟨*proof*⟩

context *heap-sep-algebra*
begin

We don't provide lemmas for derivations for the classical connectives, as Isabelle proof methods can easily deal with them.

lemma *lspasl-starl*:
$(\exists\, h1\ h2.\ (Gamma \wedge (h1,h2 \triangleright h0) \wedge (A\ h1) \wedge (B\ h2))) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge ((A ** B)\ h0) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-starl-inv*:
$Gamma \wedge ((A ** B)\ h0) \longrightarrow Delta \Longrightarrow$
$(\exists\, h1\ h2.\ (Gamma \wedge (h1,h2 \triangleright h0) \wedge (A\ h1) \wedge (B\ h2))) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-starl-der*:
$((A ** B)\ h0) \Longrightarrow (\exists\, h1\ h2.\ (h1,h2 \triangleright h0) \wedge (A\ h1) \wedge (B\ h2))$
$\langle proof \rangle$

lemma *lspasl-starl-eq*:
$((A ** B)\ h0) = (\exists\, h1\ h2.\ (h1,h2 \triangleright h0) \wedge (A\ h1) \wedge (B\ h2))$
$\langle proof \rangle$

lemma *lspasl-starr*:
$[\![Gamma \wedge (h1,h2 \triangleright h0) \longrightarrow (A\ h1) \vee ((A ** B)\ h0) \vee Delta;$
$Gamma \wedge (h1,h2 \triangleright h0) \longrightarrow (B\ h2) \vee ((A ** B)\ h0) \vee Delta]\!] \Longrightarrow$
$Gamma \wedge (h1,h2 \triangleright h0) \longrightarrow ((A ** B)\ h0) \vee Delta$
$\langle proof \rangle$

lemma *lspasl-starr-inv*:
$Gamma \wedge (h1,h2 \triangleright h0) \longrightarrow ((A ** B)\ h0) \vee Delta \Longrightarrow$
$(Gamma \wedge (h1,h2 \triangleright h0) \longrightarrow (A\ h1) \vee ((A ** B)\ h0) \vee Delta) \wedge$
$(Gamma \wedge (h1,h2 \triangleright h0) \longrightarrow (B\ h2) \vee ((A ** B)\ h0) \vee Delta)$
$\langle proof \rangle$

For efficiency we only apply *R on a pair of a ternary relational atom and a formula ONCE. To achieve this, we create a special predicate to indicate that a pair of a ternary relational atom and a formula has already been used in a *R application. Note that the predicate is true even if the *R rule hasn't been applied. We will not infer the truth of this predicate in proof search, but only check its syntactical appearance, which is only generated by the lemma lspasl_starr_der. We need to ensure that this predicate is not generated elsewhere in the proof search.

definition *starr-applied*:: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow ('a \Rightarrow bool) \Rightarrow bool$ **where**
$starr\text{-}applied\ h1\ h2\ h0\ F \equiv (h1,h2 \triangleright h0) \wedge \neg(F\ h0)$

lemma *lspasl-starr-der*:

$(h1,h2 \triangleright h0) \Longrightarrow \neg ((A ** B)\ h0) \Longrightarrow$
$((h1,h2 \triangleright h0) \wedge \neg ((A\ h1) \vee ((A ** B)\ h0)) \wedge (starr\text{-}applied\ h1\ h2\ h0\ (A ** B)))$
$\vee$
$((h1,h2 \triangleright h0) \wedge \neg ((B\ h2) \vee ((A ** B)\ h0)) \wedge (starr\text{-}applied\ h1\ h2\ h0\ (A ** B)))$
$\langle proof \rangle$

lemma *lspasl-starr-eq*:
$((h1,h2 \triangleright h0) \wedge \neg ((A ** B)\ h0)) =$
$(((h1,h2 \triangleright h0) \wedge \neg ((A\ h1) \vee ((A ** B)\ h0))) \vee ((h1,h2 \triangleright h0) \wedge \neg ((B\ h2) \vee ((A ** B)\ h0))))$
$\langle proof \rangle$

lemma *lspasl-magicl*:
$[\![Gamma \wedge (h1,h2 \triangleright h0) \wedge ((A \longrightarrow* B)\ h2) \longrightarrow (A\ h1) \vee Delta;$
$Gamma \wedge (h1,h2 \triangleright h0) \wedge ((A \longrightarrow* B)\ h2) \wedge (B\ h0) \longrightarrow Delta]\!] \Longrightarrow$
$Gamma \wedge (h1,h2 \triangleright h0) \wedge ((A \longrightarrow* B)\ h2) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-magicl-inv*:
$Gamma \wedge (h1,h2 \triangleright h0) \wedge ((A \longrightarrow* B)\ h2) \longrightarrow Delta \Longrightarrow$
$(Gamma \wedge (h1,h2 \triangleright h0) \wedge ((A \longrightarrow* B)\ h2) \longrightarrow (A\ h1) \vee Delta) \wedge$
$(Gamma \wedge (h1,h2 \triangleright h0) \wedge ((A \longrightarrow* B)\ h2) \wedge (B\ h0) \longrightarrow Delta)$
$\langle proof \rangle$

For efficiency we only apply -*L on a pair of a ternary relational atom and a formula ONCE. To achieve this, we create a special predicate to indicate that a pair of a ternary relational atom and a formula has already been used in a *R application. Note that the predicate is true even if the *R rule hasn't been applied. We will not infer the truth of this predicate in proof search, but only check its syntactical appearance, which is only generated by the lemma lspasl_magicl_der. We need to ensure that in the proof search of Separata, this predicate is not generated elsewhere.

definition *magicl-applied*:: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow ('a \Rightarrow bool) \Rightarrow bool$ **where**
$magicl\text{-}applied\ h1\ h2\ h0\ F \equiv (h1,h2 \triangleright h0) \wedge (F\ h2)$

lemma *lspasl-magicl-der*:
$(h1,h2 \triangleright h0) \Longrightarrow ((A \longrightarrow* B)\ h2) \Longrightarrow$
$((h1,h2 \triangleright h0) \wedge \neg(A\ h1) \wedge ((A \longrightarrow* B)\ h2) \wedge (magicl\text{-}applied\ h1\ h2\ h0\ (A \longrightarrow* B))) \vee$
$((h1,h2 \triangleright h0) \wedge (B\ h0) \wedge ((A \longrightarrow* B)\ h2) \wedge (magicl\text{-}applied\ h1\ h2\ h0\ (A \longrightarrow* B)))$
$\langle proof \rangle$

lemma *lspasl-magicl-eq*:
$((h1,h2 \triangleright h0) \wedge ((A \longrightarrow* B)\ h2)) =$
$(((h1,h2 \triangleright h0) \wedge \neg(A\ h1) \wedge ((A \longrightarrow* B)\ h2)) \vee ((h1,h2 \triangleright h0) \wedge (B\ h0) \wedge ((A \longrightarrow* B)\ h2)))$

$\langle proof \rangle$

lemma *lspasl-magicr*:
$(\exists h1\ h0.\ Gamma \wedge (h1,h2 \triangleright h0) \wedge (A\ h1) \wedge ((not\ B)\ h0)) \longrightarrow Delta \Longrightarrow$
$Gamma \longrightarrow ((A \longrightarrow\!\!* B)\ h2) \vee Delta$
$\langle proof \rangle$

lemma *lspasl-magicr-inv*:
$Gamma \longrightarrow ((A \longrightarrow\!\!* B)\ h2) \vee Delta \Longrightarrow$
$(\exists h1\ h0.\ Gamma \wedge (h1,h2 \triangleright h0) \wedge (A\ h1) \wedge ((not\ B)\ h0)) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-magicr-der*:
$\neg ((A \longrightarrow\!\!* B)\ h2) \Longrightarrow$
$(\exists h1\ h0.\ (h1,h2 \triangleright h0) \wedge (A\ h1) \wedge ((not\ B)\ h0))$
$\langle proof \rangle$

lemma *lspasl-magicr-eq*:
$(\neg ((A \longrightarrow\!\!* B)\ h2)) =$
$((\exists h1\ h0.\ (h1,h2 \triangleright h0) \wedge (A\ h1) \wedge ((not\ B)\ h0)))$
$\langle proof \rangle$

lemma *lspasl-eq*:
$Gamma \wedge (0,h2 \triangleright h2) \wedge h1 = h2 \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (0,h1 \triangleright h2) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-eq-inv*:
$Gamma \wedge (0,h1 \triangleright h2) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (0,h2 \triangleright h2) \wedge h1 = h2 \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-eq-der*: $(0,h1 \triangleright h2) \Longrightarrow ((0,h1 \triangleright h1) \wedge h1 = h2)$
$\langle proof \rangle$

lemma *lspasl-eq-eq*: $(0,h1 \triangleright h2) = ((0,h1 \triangleright h1) \wedge (h1 = h2))$
$\langle proof \rangle$

lemma *lspasl-u*:
$Gamma \wedge (h,0 \triangleright h) \longrightarrow Delta \Longrightarrow$
$Gamma \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-u-inv*:
$Gamma \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (h,0 \triangleright h) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-u-der*: $(h,0 \triangleright h)$

$\langle proof \rangle$

lemma *lspasl-e*:
$Gamma \wedge (h1,h2 \triangleright h0) \wedge (h2,h1 \triangleright h0) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (h1,h2 \triangleright h0) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-e-inv*:
$Gamma \wedge (h1,h2 \triangleright h0) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (h1,h2 \triangleright h0) \wedge (h2,h1 \triangleright h0) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-e-der*: $(h1,h2 \triangleright h0) \Longrightarrow (h1,h2 \triangleright h0) \wedge (h2,h1 \triangleright h0)$
$\langle proof \rangle$

lemma *lspasl-e-eq*: $(h1,h2 \triangleright h0) = ((h1,h2 \triangleright h0) \wedge (h2,h1 \triangleright h0))$
$\langle proof \rangle$

lemma *lspasl-a-der*:
assumes *a1*: $(h1,h2 \triangleright h0)$
and *a2*: $(h3,h4 \triangleright h1)$
shows $(\exists h5.\ (h3,h5 \triangleright h0) \wedge (h2,h4 \triangleright h5) \wedge (h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h1))$
$\langle proof \rangle$

lemma *lspasl-a*:
$(\exists h5.\ Gamma \wedge (h3,h5 \triangleright h0) \wedge (h2,h4 \triangleright h5) \wedge (h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h1)) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h1) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-a-inv*:
$Gamma \wedge (h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h1) \longrightarrow Delta \Longrightarrow$
$(\exists h5.\ Gamma \wedge (h3,h5 \triangleright h0) \wedge (h2,h4 \triangleright h5) \wedge (h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h1)) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-a-eq*:
$((h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h1)) =$
$(\exists h5.\ (h3,h5 \triangleright h0) \wedge (h2,h4 \triangleright h5) \wedge (h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h1))$
$\langle proof \rangle$

lemma *lspasl-p*:
$Gamma \wedge (h1,h2 \triangleright h0) \wedge h0 = h3 \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (h1,h2 \triangleright h0) \wedge (h1,h2 \triangleright h3) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-p-inv*:
$Gamma \wedge (h1,h2 \triangleright h0) \wedge (h1,h2 \triangleright h3) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (h1,h2 \triangleright h0) \wedge h0 = h3 \longrightarrow Delta$

$\langle proof \rangle$

lemma *lspasl-p-der*:
$(h1,h2\triangleright h0) \Longrightarrow (h1,h2\triangleright h3) \Longrightarrow (h1,h2\triangleright h0) \wedge h0 = h3$
$\langle proof \rangle$

lemma *lspasl-p-eq*:
$((h1,h2\triangleright h0) \wedge (h1,h2\triangleright h3)) = ((h1,h2\triangleright h0) \wedge h0 = h3)$
$\langle proof \rangle$

lemma *lspasl-c*:
$Gamma \wedge (h1,h2\triangleright h0) \wedge h2 = h3 \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (h1,h2\triangleright h0) \wedge (h1,h3\triangleright h0) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-c-inv*:
$Gamma \wedge (h1,h2\triangleright h0) \wedge (h1,h3\triangleright h0) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (h1,h2\triangleright h0) \wedge h2 = h3 \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-c-der*:
$(h1,h2\triangleright h0) \Longrightarrow (h1,h3\triangleright h0) \Longrightarrow (h1,h2\triangleright h0) \wedge h2 = h3$
$\langle proof \rangle$

lemma *lspasl-c-eq*:
$((h1,h2\triangleright h0) \wedge (h1,h3\triangleright h0)) = ((h1,h2\triangleright h0) \wedge h2 = h3)$
$\langle proof \rangle$

lemma *lspasl-iu*:
$Gamma \wedge (0,h2\triangleright 0) \wedge h1 = 0 \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (h1,h2\triangleright 0) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-iu-inv*:
$Gamma \wedge (h1,h2\triangleright 0) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (0,h2\triangleright 0) \wedge h1 = 0 \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-iu-der*:
$(h1,h2\triangleright 0) \Longrightarrow ((0,0\triangleright 0) \wedge h1 = 0 \wedge h2 = 0)$
$\langle proof \rangle$

lemma *lspasl-iu-eq*:
$(h1,h2\triangleright 0) = ((0,0\triangleright 0) \wedge h1 = 0 \wedge h2 = 0)$
$\langle proof \rangle$

lemma *lspasl-d*:
$Gamma \wedge (0,0\triangleright h2) \wedge h1 = 0 \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (h1,h1\triangleright h2) \longrightarrow Delta$

$\langle proof \rangle$

lemma *lspasl-d-inv*:
$Gamma \wedge (h1,h1 \triangleright h2) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (0,0 \triangleright h2) \wedge h1 = 0 \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-d-der*:
$(h1,h1 \triangleright h2) \Longrightarrow (0,0 \triangleright 0) \wedge h1 = 0 \wedge h2 = 0$
$\langle proof \rangle$

lemma *lspasl-d-eq*:
$(h1,h1 \triangleright h2) = ((0,0 \triangleright 0) \wedge h1 = 0 \wedge h2 = 0)$
$\langle proof \rangle$

lemma *lspasl-cs-der*:
assumes *a1*: $(h1,h2 \triangleright h0)$
and *a2*: $(h3,h4 \triangleright h0)$
shows $(\exists h5\ h6\ h7\ h8.\ (h5,h6 \triangleright h1) \wedge (h7,h8 \triangleright h2) \wedge (h5,h7 \triangleright h3) \wedge (h6,h8 \triangleright h4)$
$\wedge (h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h0))$
$\langle proof \rangle$

lemma *lspasl-cs*:
$(\exists h5\ h6\ h7\ h8.\ Gamma \wedge (h5,h6 \triangleright h1) \wedge (h7,h8 \triangleright h2) \wedge (h5,h7 \triangleright h3) \wedge (h6,h8 \triangleright h4)$
$\wedge (h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h0)) \longrightarrow Delta \Longrightarrow$
$Gamma \wedge (h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h0) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-cs-inv*:
$Gamma \wedge (h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h0) \longrightarrow Delta \Longrightarrow$
$(\exists h5\ h6\ h7\ h8.\ Gamma \wedge (h5,h6 \triangleright h1) \wedge (h7,h8 \triangleright h2) \wedge (h5,h7 \triangleright h3) \wedge (h6,h8 \triangleright h4)$
$\wedge (h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h0)) \longrightarrow Delta$
$\langle proof \rangle$

lemma *lspasl-cs-eq*:
$((h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h0)) =$
$(\exists h5\ h6\ h7\ h8.\ (h5,h6 \triangleright h1) \wedge (h7,h8 \triangleright h2) \wedge (h5,h7 \triangleright h3) \wedge (h6,h8 \triangleright h4) \wedge$
$(h1,h2 \triangleright h0) \wedge (h3,h4 \triangleright h0))$
$\langle proof \rangle$

end

The above proves the soundness and invertibility of LS_PASL.

2 Lemmas David proved for separation algebra.

lemma *sep-substate-tran*:
$x \preceq y \wedge y \preceq z \Longrightarrow x \preceq z$
$\langle proof \rangle$

lemma *precise-sep-conj*:
  assumes *a1*:*precise* I **and**
    a2:*precise* I'
  shows *precise* $(I \wedge\!* I')$
⟨*proof*⟩

lemma *unique-subheap*:
  $(\sigma 1,\sigma 2 \triangleright \sigma) \Longrightarrow \exists!\sigma 2'.(\sigma 1,\sigma 2' \triangleright \sigma)$
  ⟨*proof*⟩

lemma *sep-split-substate*:
  $(\sigma 1,\ \sigma 2 \triangleright\ \sigma) \Longrightarrow$
  $(\sigma 1\ \preceq \sigma) \wedge (\sigma 2\ \preceq \sigma)$
⟨*proof*⟩

abbreviation *sep-septraction* :: $(('a::sep\text{-}algebra) \Rightarrow bool) \Rightarrow ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow bool)$ (**infixr** $\longrightarrow\oplus$ *25*)
  where
    $P \longrightarrow\oplus\ Q \equiv\ not\ (P \longrightarrow\!*\ not\ Q)$

3 Below we integrate the inference rules in proof search.

method *try-lspasl-empl* = (
    match **premises in** *P*[*thin*]:*sep-empty ?h* ⇒
    ‹*insert lspasl-empl-der*[*OF P*]›,
    simp?
)

method *try-lspasl-starl* = (
    match **premises in** *P*[*thin*]:(*?A* ∗∗ *?B*) *?h* ⇒
    ‹*insert lspasl-starl-der*[*OF P*], *auto*›,
    simp?
)

method *try-lspasl-magicr* = (
    match **premises in** *P*[*thin*]:¬(*?A* ⟶∗ *?B*) *?h* ⇒
    ‹*insert lspasl-magicr-der*[*OF P*], *auto*›,
    simp?
)

Only apply the rule Eq on (0,h1,h2) where h1 and h2 are not syntactically the same.

method *try-lspasl-eq* = (
    match **premises in** *P*[*thin*]:(*0*,*?h1*▷*?h2*) ⇒
    ‹*match P in*
    (*0*,*h*▷*h*) *for h* ⇒ ‹*fail*›

|- ⇒ ‹*insert lspasl-eq-der*[*OF P*], *auto*››,
simp?
)

We restrict that the rule IU can't be applied on (0,0,0).

method *try-lspasl-iu* = (
match **premises in** *P*[*thin*]:(*?h1*,*?h2*▷*0*) ⇒
‹*match P in*
(*0*,*0*▷*0*) ⇒ ‹*fail*›
|- ⇒ ‹*insert lspasl-iu-der*[*OF P*], *auto*››,
simp?
)

We restrict that the rule D can't be applied on (0,0,0).

method *try-lspasl-d* = (
match **premises in** *P*[*thin*]:(*h1*,*h1*▷*h2*) **for** *h1 h2* ⇒
‹*match P in*
(*0*,*0*▷*0*) ⇒ ‹*fail*›
|- ⇒ ‹*insert lspasl-d-der*[*OF P*], *auto*››,
simp?
)

We restrict that the rule P can't be applied to two syntactically identical ternary relational atoms.

method *try-lspasl-p* = (
match **premises in** *P*[*thin*]:(*h1*,*h2*▷*h0*) **for** *h0 h1 h2* ⇒
‹*match premises in* (*h1*,*h2*▷*h0*) ⇒ ‹*fail*›
|*P'*[*thin*]:(*h1*,*h2*▷*?h3*) ⇒ ‹*insert lspasl-p-der*[*OF P P'*], *auto*››,
simp?
)

We restrict that the rule C can't be applied to two syntactically identical ternary relational atoms.

method *try-lspasl-c* = (
match **premises in** *P*[*thin*]:(*h1*,*h2*▷*h0*) **for** *h0 h1 h2* ⇒
‹*match premises in* (*h1*,*h2*▷*h0*) ⇒ ‹*fail*›
|*P'*[*thin*]:(*h1*,*?h3*▷*h0*) ⇒ ‹*insert lspasl-c-der*[*OF P P'*], *auto*››,
simp?
)

We restrict that *R only applies to a pair of a ternary relational and a formula once. Here, we need to first try simp to unify heaps. In the end, we try simp_all to simplify all branches. A similar strategy is used in -*L.

method *try-lspasl-starr* = (
simp?,
match **premises in** *P*:(*h1*,*h2*▷*h*) **and** *P'*:¬(*A* ∗∗ *B*) (*h*::'*a*::*heap-sep-algebra*)
for *h1 h2 h A B* ⇒
‹*match premises in starr-applied h1 h2 h* (*A* ∗∗ *B*) ⇒ ‹*fail*›

```
    |- ⇒ ‹insert lspasl-starr-der[OF P P′], auto››,
    simp-all?
    )
```

We restrict that -*L only applies to a pair of a ternary relational and a formula once.

```
method try-lspasl-magicl = (
    simp?,
    match premises in P: (h1,h▷h2) and P′:(A ⟶* B) (h::′a::heap-sep-algebra)
for h1 h2 h A B ⇒
    ‹match premises in magicl-applied h1 h h2 (A ⟶* B) ⇒ ‹fail›
    |- ⇒ ‹insert lspasl-magicl-der[OF P P′], auto››,
    simp-all?
    )
```

We restrict that the U rule is only applicable to a world h when (h,0,h) is not in the premises. There are two cases: (1) We pick a ternary relational atom (h1,h2,h0), and check if (h1,0,h1) occurs in the premises, if not, apply U on h1. Otherwise, check other ternary relational atoms. (2) We pick a labelled formula (A h), and check if (h,0,h) occurs in the premises, if not, apply U on h. Otherwise, check other labelled formulae.

```
method try-lspasl-u-tern = (
    match premises in
    P:(h1,h2▷(h0::′a::heap-sep-algebra)) for h1 h2 h0 ⇒
    ‹match premises in
    (h1,0▷h1) ⇒ ‹match premises in
    (h2,0▷h2) ⇒ ‹match premises in
    I1:(h0,0▷h0) ⇒ ‹fail›
    |- ⇒ ‹insert lspasl-u-der[of h0]››
    |- ⇒ ‹insert lspasl-u-der[of h2]››
    |- ⇒ ‹insert lspasl-u-der[of h1]››,
    simp?
    )
```

```
method try-lspasl-u-form = (
    match premises in
    P′:- (h::′a::heap-sep-algebra) for h ⇒
    ‹match premises in (h,0▷h) ⇒ ‹fail›
    |(0,0▷0) and h = 0 ⇒ ‹fail›
    |(0,0▷0) and 0 = h ⇒ ‹fail›
    |- ⇒ ‹insert lspasl-u-der[of h]››,
    simp?
    )
```

We restrict that the E rule is only applicable to (h1,h2,h0) when (h2,h1,h0) is not in the premises.

```
method try-lspasl-e = (
    match premises in P:(h1,h2▷h0) for h1 h2 h0 ⇒
```

‹*match premises in* $(h2,h1 \triangleright h0) \Rightarrow$ ‹*fail*›
|- $\Rightarrow$ ‹*insert lspasl-e-der*[*OF P*], *auto*››,
simp?
)

We restrict that the A rule is only applicable to (h1,h2,h0) and (h3,h4,h1) when (h3,h,h0) and (h2,h4,h) or any commutative variants of the two do not occur in the premises, for some h. Additionally, we do not allow A to be applied to two identical ternary relational atoms. We further restrict that the leaves must not be 0, because otherwise this application does not gain anything.

method *try-lspasl-a* = (
match **premises in** $(h1,h2 \triangleright h0)$ **for** *h0 h1 h2* $\Rightarrow$
‹*match premises in*
$(0,h2 \triangleright h0) \Rightarrow$ ‹*fail*›
$|(h1,0 \triangleright h0) \Rightarrow$ ‹*fail*›
$|(h1,h2 \triangleright 0) \Rightarrow$ ‹*fail*›
$|P[thin]{:}(h1,h2 \triangleright h0) \Rightarrow$
‹*match premises in*
$P'{:}(h3,h4 \triangleright h1)$ *for h3 h4* $\Rightarrow$ ‹*match premises in*
$(0,h4 \triangleright h1) \Rightarrow$ ‹*fail*›
$|(h3,0 \triangleright h1) \Rightarrow$ ‹*fail*›
$|(\text{-},h3 \triangleright h0) \Rightarrow$ ‹*fail*›
$|(h3,\text{-} \triangleright h0) \Rightarrow$ ‹*fail*›
$|(h2,h4 \triangleright \text{-}) \Rightarrow$ ‹*fail*›
$|(h4,h2 \triangleright \text{-}) \Rightarrow$ ‹*fail*›
|- $\Rightarrow$ ‹*insert P P′*, *drule lspasl-a-der*, *auto*››››,
simp?
)

I don't have a good heuristics for CS right now. I simply forbid CS to be applied on the same pair twice.

method *try-lspasl-cs* = (
match **premises in** $P[thin]{:}(h1,h2 \triangleright h0)$ **for** *h0 h1 h2* $\Rightarrow$
‹*match premises in* $(h1,h2 \triangleright h0) \Rightarrow$ ‹*fail*›
$|(h2,h1 \triangleright h0) \Rightarrow$ ‹*fail*›
$|P'{:}(h3,h4 \triangleright h0)$ *for h3 h4* $\Rightarrow$ ‹*match premises in*
$(h5,h6 \triangleright h1)$ *and* $(h7,h8 \triangleright h2)$ *and* $(h5,h7 \triangleright h3)$ *and* $(h6,h8 \triangleright h4)$ *for h5 h6 h7 h8* $\Rightarrow$ ‹*fail*›
$|(i5,i6 \triangleright h2)$ *and* $(i7,i8 \triangleright h1)$ *and* $(i5,i7 \triangleright h3)$ *and* $(i6,i8 \triangleright h4)$ *for i5 i6 i7 i8* $\Rightarrow$ ‹*fail*›
$|(j5,j6 \triangleright h1)$ *and* $(j7,j8 \triangleright h2)$ *and* $(j5,j7 \triangleright h4)$ *and* $(j6,j8 \triangleright h3)$ *for j5 j6 j7 j8* $\Rightarrow$ ‹*fail*›
$|(k5,k6 \triangleright h2)$ *and* $(k7,k8 \triangleright h1)$ *and* $(k5,k7 \triangleright h4)$ *and* $(k6,k8 \triangleright h3)$ *for k5 k6 k7 k8* $\Rightarrow$ ‹*fail*›
|- $\Rightarrow$ ‹*insert lspasl-cs-der*[*OF P P′*], *auto*›››,
simp
)

method *try-lspasl-starr-guided* = (
simp?,
match **premises in** P:($h1$,$h2$▷h) **and** P':¬(A ∗∗ B) (h::$'a$::*heap-sep-algebra*)
for *h1 h2 h A B* ⇒
‹*match premises in starr-applied h1 h2 h* (A ∗∗ B) ⇒ ‹*fail*›
|*A h1* ⇒ ‹*insert lspasl-starr-der*[*OF P P'*], *auto*›
|*B h2* ⇒ ‹*insert lspasl-starr-der*[*OF P P'*], *auto*››,
simp-all?
)

method *try-lspasl-magicl-guided* = (
simp?,
match **premises in** P: ($h1$,h▷$h2$) **and** P':(A ⟶∗ B) (h::$'a$::*heap-sep-algebra*)
for *h1 h2 h A B* ⇒
‹*match premises in magicl-applied h1 h h2* (A ⟶∗ B) ⇒ ‹*fail*›
|*A h1* ⇒ ‹*insert lspasl-magicl-der*[*OF P P'*], *auto*›
|¬(*B h2*) ⇒ ‹*insert lspasl-magicl-der*[*OF P P'*], *auto*››,
simp-all?
)

In case the conclusion is not False, we normalise the goal as below.

method *norm-goal* = (
match **conclusion in** *False* ⇒ ‹*fail*›
|- ⇒ ‹*rule ccontr*›,
simp?
)

The tactic for separata. We first try to simplify the problem with auto simp add: sep_conj_ac, which ought to solve many problems. Then we apply the "true" invertible rules and structural rules which unify worlds as much as possible, followed by auto to simplify the goals. Then we apply *R and -*L and other structural rules. The rule CS is only applied when nothing else is applicable. We try not to use it.

***** Note, (try_lspasl_u |try_lspasl_e) |try_lspasl_a)+ may cause infinite loops. *****

method *separata* =
((*auto simp add*: *sep-conj-ac*)
|(*norm-goal?*,
((*try-lspasl-empl*
|*try-lspasl-starl*
|*try-lspasl-magicr*
|*try-lspasl-iu*
|*try-lspasl-d*
|*try-lspasl-eq*
|*try-lspasl-p*
|*try-lspasl-c*

|*try-lspasl-starr-guided*
|*try-lspasl-magicl-guided*)+,
auto?)
|(*try-lspasl-u-tern*
|*try-lspasl-e*
|*try-lspasl-a*)+
|(*try-lspasl-starr*
|*try-lspasl-magicl*)
)+
|*try-lspasl-u-form*+
|*try-lspasl-cs*
)+

4 Some examples.

Let's prove something that abstract separation logic provers struggle to prove. This can be proved easily in Isabelle, proof found by Sledgehammer.

lemma *fm-hard*: ((*sep-empty imp* (*p0* ⟶∗ (((*p0* ∗∗ (*p0* ⟶∗ *p1*)) ∗∗ (*not p1*)) ⟶∗
(*p0* ∗∗ (*p0* ∗∗ ((*p0* ⟶∗ *p1*) ∗∗ (*not p1*))))))) *imp* ((((*sep-empty* ∗∗ *p0*) ∗∗
(*p0* ∗∗ ((*p0* ⟶∗ *p1*) ∗∗ (*not p1*)))) *imp* (((*p0* ∗∗ *p0*) ∗∗ (*p0* ⟶∗ *p1*)) ∗∗
(*not p1*))) ∗∗ *sep-empty*)) *h*
⟨*proof*⟩

The following formula can only be proved in partial-deterministic separation algebras. Sledgehammer took a rather long time to find a proof.

lemma *fm-partial*: (((*not* (*sep-true* ⟶∗ (*not sep-empty*))) ∗∗
(*not* (*sep-true* ⟶∗ (*not sep-empty*)))) *imp*
(*not* (*sep-true* ⟶∗ (*not sep-empty*))))
(*h*::'*a*::*heap-sep-algebra*)
⟨*proof*⟩

The following is the axiom of indivisible unit. Sledgehammer finds a proof easily.

lemma *ax-iu*: ((*sep-empty and* (*A* ∗∗ *B*)) *imp A*)
(*h*::'*a*::*heap-sep-algebra*)
⟨*proof*⟩

Sledgehammer fails to find a proof in 300s for this one.

lemma (*not* (((*A* ∗∗ (*C* ⟶∗ (*not* ((*not* (*A* ⟶∗ *B*)) ∗∗ *C*)))) *and* (*not B*)) ∗∗ *C*))
(*h*::'*a*::*heap-sep-algebra*)
⟨*proof*⟩

Sledgehammer finds a proof easily.

lemma ((*sep-empty* ⟶∗ (*not* ((*not A*) ∗∗ *sep-empty*))) *imp A*)

$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer finds a proof in 46 seconds.

lemma $(A\ imp\ (not\ ((not\ (A ** B))\ and\ (not\ (A ** (not\ B))))))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer easily finds a proof.

lemma $((sep\text{-}empty\ and\ A)\ imp\ (A ** A))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer fails to find a proof in 300s.

lemma $(not\ (((A ** (C \longrightarrow* (not\ ((not\ (A \longrightarrow* B)) ** C))))\ and\ (not\ B)) ** C))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer finds a proof easily.

lemma $((sep\text{-}empty \longrightarrow* (not\ ((not\ A) ** sep\text{-}empty)))\ imp\ A)$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer finds a proof easily.

lemma $(sep\text{-}empty\ imp\ ((A ** B) \longrightarrow* (B ** A)))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer takes a while to find a proof, although the proof is by smt and is fast.

lemma $(sep\text{-}empty\ imp\ ((A ** (B\ and\ C)) \longrightarrow* ((A ** B)\ and\ (A ** C))))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer takes a long time to find a smt proof, but the smt proves it quickly.

lemma $(sep\text{-}empty\ imp\ ((A \longrightarrow* (B\ imp\ C)) \longrightarrow* ((A \longrightarrow* B)\ imp\ (A \longrightarrow* C))))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer finds a proof quickly.

lemma $(sep\text{-}empty\ imp\ (((A\ imp\ B) \longrightarrow* ((A \longrightarrow* A)\ imp\ A))\ imp\ (A \longrightarrow* A)))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer finds proofs in a while.

lemma $((A \longrightarrow* B)$ *and* $(sep\text{-}true ** (sep\text{-}empty$ *and* $A))$ *imp* $B)$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer finds proofs easily.

lemma $((sep\text{-}empty \longrightarrow* (not\ ((not\ A) ** sep\text{-}true)))$ *imp* $A)$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer takes a while to find a proof.

lemma $(not\ ((A \longrightarrow* (not\ (A ** B)))$ *and* $(((not\ A) \longrightarrow* (not\ B))$ *and* $B)))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer takes a long time to find a smt proof, although smt proves it quickly.

lemma $(sep\text{-}empty$ *imp* $((A \longrightarrow* (B \longrightarrow* C)) \longrightarrow* ((A ** B) \longrightarrow* C)))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer finds proofs easily.

lemma $(sep\text{-}empty$ *imp* $((A\ **\ (B ** C)) \longrightarrow* ((A ** B) ** C)))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer finds proofs in a few seconds.

lemma $(sep\text{-}empty$ *imp* $((A ** ((B \longrightarrow* D) ** C)) \longrightarrow* ((A ** (B \longrightarrow* D)) ** C)))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer fails to find a proof in 300s.

lemma $(not\ (((A \longrightarrow* (not\ ((not\ (D \longrightarrow* (not\ (A ** (C ** B))))) ** A)))$ *and* $C) ** (D$ *and* $(A ** B))))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer takes a while to find a proof.

lemma $(not\ ((C ** (D ** E))$ *and* $((A \longrightarrow* (not\ (not\ (B \longrightarrow* not\ (D ** (E ** C))) ** A))) **$
$(B$ *and* $(A ** sep\text{-}true)))))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
$\langle proof \rangle$

Sledgehammer fails to find a proof in 300s.

lemma $(not\ (((A \longrightarrow* (not\ ((not\ (D \longrightarrow* (not\ ((C ** E) ** (B ** A))))) ** A))) \ and\ C) ** (D\ and\ (A ** (B ** E)))))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
⟨*proof*⟩

Sledgehammer finds a proof easily.

lemma $((A ** (B ** (C ** (D ** E))))\ imp\ (E ** (B ** (A ** (C ** D)))))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
⟨*proof*⟩

lemma $((A ** (B ** (C ** (D ** (E ** (F ** G))))))\ imp\ (G ** (E ** (B ** (A ** (C ** (D ** F)))))))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
⟨*proof*⟩

Sledgehammer finds a proof in a few seconds.

lemma $(sep\text{-}empty\ imp\ ((A ** ((B \longrightarrow* E) ** (C ** D))) \longrightarrow* ((A ** D) ** (C ** (B \longrightarrow* E)))))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
⟨*proof*⟩

This is the odd BBI formula that I personally can't prove using any other methods. I only know of a derivation in my labelled sequent calculus for BBI. Sledgehammer takes a while to find a proof.

lemma $(not\ (sep\text{-}empty\ and\ A\ and\ (B ** (not\ (C \longrightarrow* (sep\text{-}empty\ imp\ A))))))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
⟨*proof*⟩

Sledgehammer finds a proof easily.

lemma $((((sep\text{-}true\ imp\ p0)\ imp\ ((p0 ** p0) \longrightarrow* ((sep\text{-}true\ imp\ p0) ** (p0 ** p0))))\ imp$
$(p1 \longrightarrow* (((sep\text{-}true\ imp\ p0)\ imp\ ((p0 ** p0) \longrightarrow* (((sep\text{-}true\ imp\ p0) ** p0) ** p0))) ** p1))))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
⟨*proof*⟩

The following are some randomly generated BBI formulae.

Sledgehammer finds a proof easily.

lemma $((((p1 \longrightarrow* p3) \longrightarrow* (p5 \longrightarrow* p2))\ imp\ ((((p7 ** p4)\ and\ (p3 \longrightarrow* p2))\ imp$
$((p7 ** p4)\ and\ (p3 \longrightarrow* p2))) \longrightarrow* (((p1 \longrightarrow* p3) \longrightarrow* (p5 \longrightarrow* p2)) **$
$(((p4 ** p7)\ and\ (p3 \longrightarrow* p2))\ imp\ ((p4 ** p7)\ and\ (p3 \longrightarrow* p2)))))))$
$(h::'a::heap\text{-}sep\text{-}algebra)$
⟨*proof*⟩

Sledgehammer finds a proof easily.

lemma (((((*p1* ⟶∗ (*p0 imp* *sep-false*)) *imp* *sep-false*) *imp* (((*p1 imp* *sep-false*) *imp*
((*p0* ∗∗ ((*p1 imp* *sep-false*) ⟶∗ (*p4* ⟶∗ *p1*))) ⟶∗ ((*p1 imp* *sep-false*) ∗∗
(*p0* ∗∗ ((*p1 imp* *sep-false*) ⟶∗ (*p4* ⟶∗ *p1*)))))) *imp* *sep-false*)) *imp*

(((*p1 imp* *sep-false*) *imp* ((*p0* ∗∗ ((*p1 imp* *sep-false*) ⟶∗ (*p4* ⟶∗ *p1*))) ⟶∗
((*p0* ∗∗ (*p1 imp* *sep-false*)) ∗∗ ((*p1 imp* *sep-false*) ⟶∗ (*p4* ⟶∗ *p1*))))) *imp*
(*p1* ⟶∗ (*p0 imp* *sep-false*)))))
(*h*::′*a*::*heap-sep-algebra*)
⟨*proof*⟩

Sledgehammer finds a proof easily.

lemma (((*p0 imp* *sep-false*) *imp* ((*p1* ∗∗ *p0*) ⟶∗ (*p1* ∗∗ ((*p0 imp* *sep-false*) ∗∗
p0)))) *imp* ((*p0 imp* *sep-false*) *imp* ((*p1* ∗∗ *p0*) ⟶∗ ((*p1* ∗∗ *p0*) ∗∗ (*p0 imp*
sep-false)))))
(*h*::′*a*::*heap-sep-algebra*)
⟨*proof*⟩

Sledgehammer finds a proof in a while.

lemma (*sep-empty* *imp* ((((*p4* ∗∗ *p1*) ⟶∗ ((*p8* ∗∗ *sep-empty*) ⟶∗ *p0*)) *imp*
(*p1* ⟶∗ (*p1* ∗∗ ((*p4* ∗∗ *p1*) ⟶∗ ((*p8* ∗∗ *sep-empty*) ⟶∗ *p0*))))) ⟶∗
(((*p4* ∗∗ *p1*) ⟶∗ ((*p8* ∗∗ *sep-empty*) ⟶∗ *p0*)) *imp* (*p1* ⟶∗ (((*p1* ∗∗ *p4*) ⟶∗
((*p8* ∗∗ *sep-empty*) ⟶∗ *p0*)) ∗∗ *p1*)))))
(*h*::′*a*::*heap-sep-algebra*)
⟨*proof*⟩

Sledgehammer finds a proof easily.

lemma ((((*p3 imp* (*p0* ⟶∗ (*p3* ∗∗ *p0*))) *imp* *sep-false*) *imp* (*p1 imp* *sep-false*)) *imp*
(*p1 imp* (*p3 imp* (*p0* ⟶∗ (*p0* ∗∗ *p3*)))))
(*h*::′*a*::*heap-sep-algebra*)
⟨*proof*⟩

Sledgehammer finds a proof in a few seconds.

lemma ((*p7* ⟶∗ (*p4* ∗∗ (*p6* ⟶∗ *p1*))) *imp* ((*p4 imp* (*p1* ⟶∗ ((*sep-empty* ∗∗
p1) ∗∗ *p4*))) ⟶∗ ((*p1 imp* (*p4* ⟶∗ (*p4* ∗∗ (*sep-empty* ∗∗ *p1*)))) ∗∗ (*p7* ⟶∗
((*p6* ⟶∗ *p1*) ∗∗ *p4*)))))
(*h*::′*a*::*heap-sep-algebra*)

⟨*proof*⟩

Sledgehammer finds a proof easily.

lemma (((*p2 imp p0*) *imp* ((*p0* ∗∗ *sep-true*) ⟶∗ (*p0* ∗∗ (*sep-true* ∗∗ (*p2 imp p0*))))) *imp* ((*p2 imp p0*) *imp* ((*sep-true* ∗∗ *p0*) ⟶∗ (*p0* ∗∗ ((*p2 imp p0*) ∗∗ *sep-true*)))))
(*h*::′*a*::*heap-sep-algebra*)
⟨*proof*⟩

Sledgehammer finds a proof easily.

lemma ((*sep-empty imp* ((*p1* ⟶∗ (((*p2 imp sep-false*) ∗∗ *p0*) ∗∗ *p8*)) ⟶∗ (*p1* ⟶∗ ((*p2 imp sep-false*) ∗∗ (*p0* ∗∗ *p8*)))))) *imp* ((*p0* ∗∗ *sep-empty*) ⟶∗ ((*sep-empty imp* ((*p1* ⟶∗ ((*p0* ∗∗ (*p2 imp sep-false*)) ∗∗ *p8*)) ⟶∗ (*p1* ⟶∗ ((*p2 imp sep-false*) ∗∗ (*p0* ∗∗ *p8*))))) ∗∗ (*p0* ∗∗ *sep-empty*))))
(*h*::′*a*::*heap-sep-algebra*)
⟨*proof*⟩

Sledgehammer finds a proof in a while.

lemma ((*p0* ⟶∗ *sep-empty*) *imp* ((*sep-empty imp* ((*sep-empty* ∗∗ ((((*p8* ∗∗ *p7*) ∗∗ (*p8 imp p4*)) ⟶∗ *p8*) ∗∗ (*p2* ∗∗ *p1*))) ⟶∗ (*p2* ∗∗ (((*p7* ∗∗ ((*p8 imp p4*) ∗∗ *p8*)) ⟶∗ *p8*) ∗∗ *p1*)))) ⟶∗ ((*sep-empty imp* (((((*p7* ∗∗ (*p8* ∗∗ (*p8 imp p4*))) ⟶∗ *p8*) ∗∗ *sep-empty*) ∗∗ (*p1* ∗∗ *p2*)) ⟶∗ (((*p7* ∗∗ ((*p8 imp p4*) ∗∗ *p8*)) ⟶∗ *p8*) ∗∗ (*p1* ∗∗ *p2*)))) ∗∗ (*p0* ⟶∗ *sep-empty*))))
(*h*::′*a*::*heap-sep-algebra*)
⟨*proof*⟩

end