

Automated Cyclic Entailment Proofs in Separation Logic

James Brotherston² Dino Distefano¹
Rasmus L. Petersen¹

¹School of Electronic Engineering and Computer Science
Queen Mary University of London

²Department of Computing
Imperial College London

August 2, 2011

Motivation

Goal I: Program Verification

Real world programs are big

Motivation

Goal I: Program Verification

Real world programs are big
Real world programs use the heap

Motivation

Goal I: Program Verification

Real world programs are big

Real world programs use the heap

Real world programs use inductive datatypes

Motivation

Goal I: Program Verification

Real world programs are big

Real world programs use the heap

Real world programs use inductive datatypes

Solution

Separation Logic handles big heap programs

Motivation

Goal I: Program Verification

Real world programs are big

Real world programs use the heap

Real world programs use inductive datatypes

Solution

Separation Logic handles big heap programs

Cyclic proofs handles inductive predicates

Motivation

Goal 2: Spread cyclic proofs

Cyclic proofs circumvent induction hypotheses (automatable)

Motivation

Goal 2: Spread cyclic proofs

Cyclic proofs circumvent induction hypotheses (automatable)
Cyclic proofs is a generic methodology

Motivation

Goal 2: Spread cyclic proofs

Cyclic proofs circumvent induction hypotheses (automatable)

Cyclic proofs is a generic methodology

Cyclic proofs are sometimes more natural

Motivation

Goal 2: Spread cyclic proofs

Cyclic proofs circumvent induction hypotheses (automatable)
Cyclic proofs is a generic methodology
Cyclic proofs are sometimes more natural

Plan

Demonstrate that turning an ordinary proof system into a cyclic one is relatively painless

Outline

- 1 Introduction
 - Separation Logic
 - Cyclic Proofs

Outline

- 1 Introduction
 - Separation Logic
 - Cyclic Proofs
- 2 Theory
 - Formal System
 - Soundness

Outline

- 1 Introduction
 - Separation Logic
 - Cyclic Proofs
- 2 Theory
 - Formal System
 - Soundness
- 3 Implementation
 - Augmented Proof Rules
 - Soundness
 - Proof Search

Outline

- 1 Introduction
 - Separation Logic
 - Cyclic Proofs
- 2 Theory
 - Formal System
 - Soundness
- 3 Implementation
 - Augmented Proof Rules
 - Soundness
 - Proof Search

Separation Logic

Separation logic is ordinary logic with extra spatial connectives

$*$, \multimap

Separation Logic

Separation logic is ordinary logic with extra spatial connectives

$*$, \multimap

$h \in P * Q$ if h can be split (disjointly) into h_1, h_2 such that
 $h_1 \in P$ and $h_2 \in Q$

Separation Logic

Separation logic is ordinary logic with extra spatial connectives

$*$, \multimap

$h \in P * Q$ if h can be split (disjointly) into h_1, h_2 such that
 $h_1 \in P$ and $h_2 \in Q$

It also has a basic heap predicate, $x \mapsto y$

Separation Logic

Separation logic is ordinary logic with extra spatial connectives

$*$, \multimap

$h \in P * Q$ if h can be split (disjointly) into h_1, h_2 such that
 $h_1 \in P$ and $h_2 \in Q$

It also has a basic heap predicate, $x \mapsto y$

$h \in x \mapsto y$ if h has a single cell x with content y

Outline

- 1 Introduction
 - Separation Logic
 - Cyclic Proofs
- 2 Theory
 - Formal System
 - Soundness
- 3 Implementation
 - Augmented Proof Rules
 - Soundness
 - Proof Search

In One Slide

Assume a logic with inductive predicates

In One Slide

Assume a logic with inductive predicates

No induction rules for inductive predicates

In One Slide

Assume a logic with inductive predicates

No induction rules for inductive predicates

Instead have simple case splitting rules

In One Slide

Assume a logic with inductive predicates

No induction rules for inductive predicates

Instead have simple case splitting rules

Allow proofs to contain loops

In One Slide

Assume a logic with inductive predicates

No induction rules for inductive predicates

Instead have simple case splitting rules

Allow proofs to contain loops

Only sound for certain proofs

Inductive Predicates

An inductive predicate is defined by giving a set of implications

Example

Non-empty lists:

$$\begin{aligned}x \mapsto y &\Rightarrow \text{ls } x \ y \\x \mapsto y * \text{ls } y \ z &\Rightarrow \text{ls } x \ z\end{aligned}$$

Inductive Predicates

An inductive predicate is defined by giving a set of implications

Example

Non-empty lists:

$$\begin{aligned}x \mapsto y &\Rightarrow \text{ls } x \ y \\x \mapsto y * \text{ls } y \ z &\Rightarrow \text{ls } x \ z\end{aligned}$$

This describes when one can conclude the predicate to hold

Inductive Predicates

An inductive predicate is defined by giving a set of implications

Example

Non-empty lists:

$$\begin{aligned}x \mapsto y &\Rightarrow \text{ls } x \ y \\x \mapsto y * \text{ls } y \ z &\Rightarrow \text{ls } x \ z\end{aligned}$$

This describes when one can conclude the predicate to hold
We think of each implication as an introduction rule

List Rules

From the implications for non-empty lists, we get the rules

$$\frac{\Phi_1 \vdash \Phi_2 * x \mapsto y}{\Phi_1 \vdash \Phi_2 * \text{ls } x \ y} \quad \frac{\Phi_1 \vdash \Phi_2 * x \mapsto y * \text{ls } y \ z}{\Phi_1 \vdash \Phi_2 * \text{ls } x \ z}$$
$$\frac{\Phi_1 * x \mapsto z \vdash \Phi_2 \quad \Phi_1 * x \mapsto y * \text{ls } y \ z \vdash \Phi_2}{\Phi_1 * \text{ls } x \ z \vdash \Phi_2}$$

Example Cyclic Proof

$$x \mapsto y * \text{ls } yz \vdash \text{ls } xz$$

$$x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz$$

$$\text{ls } xy * \text{ls } yz \vdash \text{ls } xz$$

Example Cyclic Proof

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}$$

Example Cyclic Proof

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz} \quad \frac{\frac{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}$$

Example Cyclic Proof

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}}
 \frac{\frac{x \mapsto w \vdash x \mapsto w \quad \overline{\text{ls } wy * \text{ls } yz \vdash \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}}$$

Example Cyclic Proof

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz} \quad \frac{\frac{x \mapsto w \vdash x \mapsto w \quad \text{ls } wy * \text{ls } yz \vdash \text{ls } wz}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz} \text{ cycle}$$

A shorter proof?

$$\frac{}{\frac{}{lsxy * lsyZ \vdash lsxz}}$$

A shorter proof?

$$\frac{\overline{ls\ x\ y * ls\ y\ z \vdash ls\ x\ z}}{ls\ x\ y * ls\ y\ z \vdash ls\ x\ z}$$

A shorter proof?

$$\frac{\textit{cycle}}{\frac{ls\ x\ y * ls\ y\ z \vdash ls\ x\ z}{ls\ x\ y * ls\ y\ z \vdash ls\ x\ z}}$$

A shorter proof?

$$\frac{\textit{cycle}}{\frac{l s x y * l s y z \vdash l s x z}{l s x y * l s y z \vdash l s x z}}$$

Clearly too easy...

A shorter proof?

$$\frac{\textit{cycle}}{\frac{\text{ls } x y * \text{ls } y z \vdash \text{ls } x z}{\text{ls } x y * \text{ls } y z \vdash \text{ls } x z}}$$

Clearly too easy...

Intuitive soundness condition

Every cycle must contain an unfolding step

Outline

- 1 Introduction
 - Separation Logic
 - Cyclic Proofs
- 2 Theory
 - **Formal System**
 - Soundness
- 3 Implementation
 - Augmented Proof Rules
 - Soundness
 - Proof Search

Formulas

Formulas are given inductively by the grammar:

$$\begin{aligned} F & ::= \top \mid \perp \mid x = y \mid x \neq y \\ & \mid \mathbf{emp} \mid x \mapsto y \mid x \xrightarrow{2} y, z \\ & \mid F \vee F \mid F * F \mid P\mathbf{x} \end{aligned}$$

Formulas

Formulas are given inductively by the grammar:

$$\begin{aligned} F ::= & \top \mid \perp \mid x = y \mid x \neq y \\ & \mid \mathbf{emp} \mid x \mapsto y \mid x \xrightarrow{2} y, z \\ & \mid F \vee F \mid F * F \mid \mathbf{Px} \end{aligned}$$

Formulas

Formulas are given inductively by the grammar:

$$\begin{aligned} F ::= & \top \mid \perp \mid x = y \mid x \neq y \\ & \mid \mathbf{emp} \mid x \mapsto y \mid x \xrightarrow{2} y, z \\ & \mid F \vee F \mid F * F \mid P\mathbf{x} \end{aligned}$$

No ordinary conjunction (\wedge) or spatial implication (\multimap)

Basic Proof Rules

$$\overline{F \vdash F} \text{ (Id)} \quad \overline{\perp * F \vdash G} \text{ (\perp L)} \quad \overline{F \vdash \top} \text{ (\top R)} \quad \overline{F \vdash x = x} \text{ (=R)}$$

$$\overline{x = y * x \neq y * F \vdash G} \text{ (=L)} \quad \overline{x \mapsto y * x \mapsto z * F \vdash G} \text{ (\mapsto)}$$

$$\overline{x \xrightarrow{2} y_1, y_2 * x \xrightarrow{2} z_1, z_2 * F \vdash G} \text{ ($\xrightarrow{2}$)} \quad \frac{F \vdash H \quad H \vdash G}{F \vdash G} \text{ (Cut)}$$

$$\frac{F \vdash G}{\mathbf{emp} * F \vdash G} \text{ (empL)} \quad \frac{F \vdash G}{F \vdash G * \mathbf{emp}} \text{ (empR)} \quad \frac{F_1 \vdash G_1 \quad F_2 \vdash G_2}{F_1 * F_2 \vdash G_1 * G_2} (*)$$

$$\frac{F_1 * F \vdash G \quad F_2 * F \vdash G}{(F_1 \vee F_2) * F \vdash G} \text{ (\vee L)} \quad \frac{F \vdash G_i * G}{F \vdash (G_1 \vee G_2) * G} \quad i \in \{1, 2\} \text{ (\vee R)}$$

Inductive Rule Sets

An *inductive rule set* is a finite set of *inductive rules* each of the form

$$F \stackrel{\mathbf{z}}{\Rightarrow} P\mathbf{x}$$

where F and $P\mathbf{x}$ are formulas with P a predicate symbol

and \mathbf{z} is a tuple of distinct variables, such that

$$FV(F) \cup \{\mathbf{x}\} = \{\mathbf{z}\}$$

Inductive Rule Sets

An *inductive rule set* is a finite set of *inductive rules* each of the form

$$F \stackrel{\mathbf{z}}{\Rightarrow} P\mathbf{x}$$

where F and $P\mathbf{x}$ are formulas with P a predicate symbol

and \mathbf{z} is a tuple of distinct variables, such that

$$FV(F) \cup \{\mathbf{x}\} = \{\mathbf{z}\}$$

From now on we assume a fixed inductive rule set Φ .

Inductive Proof Rules

For each inductive rule $F \xRightarrow{\mathbf{z}} P\mathbf{x}$ there is a right-unfolding rule for P :

$$\frac{G \vdash H * F[\mathbf{y}/\mathbf{z}]}{G \vdash H * P\mathbf{x}[\mathbf{y}/\mathbf{z}]} \text{ (PR)}$$

where \mathbf{y} is any tuple of variables of the same length as \mathbf{z} .

(Note that $\{\mathbf{x}\} \subseteq \{\mathbf{z}\}$ by definition, so that in $P\mathbf{x}[\mathbf{y}/\mathbf{z}]$ all of the variables in \mathbf{x} are uniformly replaced by arbitrary variables from \mathbf{y} .)

Inductive Proof Rules

The left-unfolding, or *case-split* rule for P has the following general schema:

$$\frac{\text{case premises}}{G * P\mathbf{v} \vdash H} \text{ (Case } P\text{)}$$

where, for each inductive rule of the form $F \stackrel{z}{\Rightarrow} P\mathbf{x}$, there is a *case premise*:

$$G[(\mathbf{x}[\mathbf{y}/\mathbf{z}])/\mathbf{v}] * F[\mathbf{y}/\mathbf{z}] \vdash H[(\mathbf{x}[\mathbf{y}/\mathbf{z}])/\mathbf{v}]$$

where the variables \mathbf{y} are *fresh*, i.e. $y \notin FV(G * P\mathbf{v}) \cup FV(H)$ for all $y \in \{\mathbf{y}\}$.

Pre-proofs I

Definition (Buds and Companions)

A *bud* in a derivation tree \mathcal{D} is a sequent occurrence in \mathcal{D} to which no proof rule has been applied.

A *companion* for a bud B is a sequent occurrence C in \mathcal{D} of which B is a substitution instance, i.e. $B = C[\theta]$ for some substitution θ .

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \text{ls } wy * \text{ls } yz \vdash \text{ls } wz}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz} \quad \text{ls } xy * \text{ls } yz \vdash \text{ls } xz$$

Pre-proofs I

Definition (Buds and Companions)

A *bud* in a derivation tree \mathcal{D} is a sequent occurrence in \mathcal{D} to which no proof rule has been applied.

A *companion* for a bud B is a sequent occurrence C in \mathcal{D} of which B is a substitution instance, i.e. $B = C[\theta]$ for some substitution θ .

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \overline{\text{ls } wy * \text{ls } yz \vdash \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz} \quad \text{ls } xy * \text{ls } yz \vdash \text{ls } xz$$

Pre-proofs I

Definition (Buds and Companions)

A *bud* in a derivation tree \mathcal{D} is a sequent occurrence in \mathcal{D} to which no proof rule has been applied.

A *companion* for a bud B is a sequent occurrence C in \mathcal{D} of which B is a substitution instance, i.e. $B = C[\theta]$ for some substitution θ .

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \text{ls } wy * \text{ls } yz \vdash \text{ls } wz}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz} \quad \text{ls } xy * \text{ls } yz \vdash \text{ls } xz$$

Pre-proofs I

Definition (Buds and Companions)

A *bud* in a derivation tree \mathcal{D} is a sequent occurrence in \mathcal{D} to which no proof rule has been applied.

A *companion* for a bud B is a sequent occurrence C in \mathcal{D} of which B is a substitution instance, i.e. $B = C[\theta]$ for some substitution θ .

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \overline{\text{ls } wy * \text{ls } yz \vdash \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz} \quad \text{ls } xy * \text{ls } yz \vdash \text{ls } xz$$

$$\theta = [w/x]$$

Pre-proofs II

Definition (Pre-proof)

A *pre-proof* of a sequent S is given by $(\mathcal{D}, \mathcal{R})$, where \mathcal{D} is a derivation tree whose root is S and \mathcal{R} is a function assigning a companion to every bud of \mathcal{D} .

Pre-proofs II

Definition (Pre-proof)

A *pre-proof* of a sequent S is given by $(\mathcal{D}, \mathcal{R})$, where \mathcal{D} is a derivation tree whose root is S and \mathcal{R} is a function assigning a companion to every bud of \mathcal{D} .

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{\frac{x \mapsto w \vdash x \mapsto w \quad \text{ls } wy * \text{ls } yz \vdash \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}$$

Pre-proofs II

Definition (Pre-proof)

A *pre-proof* of a sequent S is given by $(\mathcal{D}, \mathcal{R})$, where \mathcal{D} is a derivation tree whose root is S and \mathcal{R} is a function assigning a companion to every bud of \mathcal{D} .

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \overline{\text{ls } wy * \text{ls } yz \vdash \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz} \quad \overline{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}$$

Pre-proofs II

Definition (Pre-proof)

A *pre-proof* of a sequent S is given by $(\mathcal{D}, \mathcal{R})$, where \mathcal{D} is a derivation tree whose root is S and \mathcal{R} is a function assigning a companion to every bud of \mathcal{D} .

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \text{ls } wy * \text{ls } yz \vdash \text{ls } wz}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz} \quad \text{ls } xy * \text{ls } yz \vdash \text{ls } xz$$

Soundness Condition

Definition (Path)

A *path* in a pre-proof is a sequence of sequent occurrences $(F_i \vdash G_i)_{i \geq 0}$ such that, for all $i \geq 0$, it holds that either $F_{i+1} \vdash G_{i+1}$ is a premise of the rule instance in \mathcal{D} with conclusion $F_i \vdash G_i$, or $F_{i+1} \vdash G_{i+1} = \mathcal{R}(F_i \vdash G_i)$.

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \overline{\text{ls } wy * \text{ls } yz \vdash \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz \quad x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}$$

Soundness Condition

Definition (Path)

A *path* in a pre-proof is a sequence of sequent occurrences $(F_i \vdash G_i)_{i \geq 0}$ such that, for all $i \geq 0$, it holds that either $F_{i+1} \vdash G_{i+1}$ is a premise of the rule instance in \mathcal{D} with conclusion $F_i \vdash G_i$, or $F_{i+1} \vdash G_{i+1} = \mathcal{R}(F_i \vdash G_i)$.

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \text{ls } wy * \text{ls } yz \vdash \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz \quad x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}$$

Soundness Condition

Definition (Path)

A *path* in a pre-proof is a sequence of sequent occurrences $(F_i \vdash G_i)_{i \geq 0}$ such that, for all $i \geq 0$, it holds that either $F_{i+1} \vdash G_{i+1}$ is a premise of the rule instance in \mathcal{D} with conclusion $F_i \vdash G_i$, or $F_{i+1} \vdash G_{i+1} = \mathcal{R}(F_i \vdash G_i)$.

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz}}{\frac{x \mapsto w \vdash x \mapsto w \quad \text{ls } wy * \text{ls } yz \vdash \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}$$

$$\frac{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz} 0$$

Soundness Condition

Definition (Path)

A *path* in a pre-proof is a sequence of sequent occurrences $(F_i \vdash G_i)_{i \geq 0}$ such that, for all $i \geq 0$, it holds that either $F_{i+1} \vdash G_{i+1}$ is a premise of the rule instance in \mathcal{D} with conclusion $F_i \vdash G_i$, or $F_{i+1} \vdash G_{i+1} = \mathcal{R}(F_i \vdash G_i)$.

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \text{ls } wy * \text{ls } yz \vdash \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz \quad 1}$$

$$\text{ls } xy * \text{ls } yz \vdash \text{ls } xz \quad 0$$

Soundness Condition

Definition (Path)

A *path* in a pre-proof is a sequence of sequent occurrences $(F_i \vdash G_i)_{i \geq 0}$ such that, for all $i \geq 0$, it holds that either $F_{i+1} \vdash G_{i+1}$ is a premise of the rule instance in \mathcal{D} with conclusion $F_i \vdash G_i$, or $F_{i+1} \vdash G_{i+1} = \mathcal{R}(F_i \vdash G_i)$.

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \text{ls } wy * \text{ls } yz \vdash \text{ls } wz}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz} \quad 2}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz} \quad 1}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz} \quad 0$$

Soundness Condition

Definition (Path)

A *path* in a pre-proof is a sequence of sequent occurrences $(F_i \vdash G_i)_{i \geq 0}$ such that, for all $i \geq 0$, it holds that either $F_{i+1} \vdash G_{i+1}$ is a premise of the rule instance in \mathcal{D} with conclusion $F_i \vdash G_i$, or $F_{i+1} \vdash G_{i+1} = \mathcal{R}(F_i \vdash G_i)$.

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \text{ls } wy * \text{ls } yz \vdash \text{ls } wz \ 3}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz \ 2}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz \ 1}$$

$$\frac{\text{ls } x * \text{ls } yz \vdash \text{ls } xz \ 0}{\text{ls } x * \text{ls } yz \vdash \text{ls } xz \ 0}$$

Soundness Condition

Definition (Path)

A *path* in a pre-proof is a sequence of sequent occurrences $(F_i \vdash G_i)_{i \geq 0}$ such that, for all $i \geq 0$, it holds that either $F_{i+1} \vdash G_{i+1}$ is a premise of the rule instance in \mathcal{D} with conclusion $F_i \vdash G_i$, or $F_{i+1} \vdash G_{i+1} = \mathcal{R}(F_i \vdash G_i)$.

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \text{ls } wy * \text{ls } yz \vdash \text{ls } wz \text{ 3}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz \text{ 2}}}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz \quad x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz \text{ 1}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz \text{ 0 4}}$$

Soundness Condition

Definition (Path)

A *path* in a pre-proof is a sequence of sequent occurrences $(F_i \vdash G_i)_{i \geq 0}$ such that, for all $i \geq 0$, it holds that either $F_{i+1} \vdash G_{i+1}$ is a premise of the rule instance in \mathcal{D} with conclusion $F_i \vdash G_i$, or $F_{i+1} \vdash G_{i+1} = \mathcal{R}(F_i \vdash G_i)$.

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \text{ls } wy * \text{ls } yz \vdash \text{ls } wz \text{ 3}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz \text{ 2}}}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz \quad x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz \text{ 1 5}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz \text{ 0 4}}$$

Soundness Condition

Definition (Trace)

Let $(F_i \vdash G_i)_{i \geq 0}$ be a path in a pre-proof \mathcal{P} . A *trace following* $(F_i \vdash G_i)_{i \geq 0}$ is a sequence $(A_i)_{i \geq 0}$ such that, for all $i \geq 0$, A_i is a subformula occurrence of the form $P\mathbf{x}$ in the formula F_i , and either:

- (i) A_{i+1} is the subformula occurrence in F_{i+1} corresponding to A_i in F_i , or
- (ii) $F_i \vdash G_i$ is the conclusion of an instance of a case-split rule (Case P), A_i is the formula $P\mathbf{v}$ unfolded by the rule and A_{i+1} is a subformula of the formula $F[\mathbf{y}/\mathbf{z}]$ obtained by the unfolding, in which case i is said to be a *progress point* of the trace.

Soundness Condition

Let $(F_i \vdash G_i)_{i \geq 0}$ be a path in a pre-proof \mathcal{P} .

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{\frac{x \mapsto w \vdash x \mapsto w \quad \text{ls } wy * \text{ls } yz \vdash \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}$$

Soundness Condition

Let $(F_i \vdash G_i)_{i \geq 0}$ be a path in a pre-proof \mathcal{P} .

A *trace following* $(F_i \vdash G_i)_{i \geq 0}$ is a sequence $(A_i)_{i \geq 0}$ such that, for all $i \geq 0$, A_i is a subformula occurrence of the form $P\mathbf{x}$ in the formula F_i , and...

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \frac{\text{ls } wy * \text{ls } yz \vdash \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}$$

Soundness Condition

... either:

- (i) A_{i+1} is the subformula occurrence in F_{i+1} corresponding to A_i in F_i , or
- (ii) $F_i \vdash G_i$ is the conclusion of an instance of a case-split rule (Case P), A_i is the formula $P\mathbf{v}$ unfolded by the rule and A_{i+1} is a subformula of the formula $F[\mathbf{y}/\mathbf{z}]$ obtained by the unfolding, in which case i is said to be a *progress point* of the trace.

$$\frac{\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz}}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}}
 \frac{\frac{\frac{x \mapsto w \vdash x \mapsto w \quad \overline{\text{ls } wy * \text{ls } yz \vdash \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}}$$

Soundness Condition

... either:

- (i) A_{i+1} is the subformula occurrence in F_{i+1} corresponding to A_i in F_i , or
- (ii) $F_i \vdash G_i$ is the conclusion of an instance of a case-split rule (Case P), A_i is the formula $P\mathbf{v}$ unfolded by the rule and A_{i+1} is a subformula of the formula $F[\mathbf{y}/\mathbf{z}]$ obtained by the unfolding, in which case i is said to be a *progress point* of the trace.

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}}{\frac{x \mapsto w \vdash x \mapsto w \quad \frac{\text{ls } wy * \text{ls } yz \vdash \text{ls } wz}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz}}{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}}$$

Soundness Condition

... either:

- (i) A_{i+1} is the subformula occurrence in F_{i+1} corresponding to A_i in F_i , or
- (ii) $F_i \vdash G_i$ is the conclusion of an instance of a case-split rule (Case P), A_i is the formula $P\mathbf{v}$ unfolded by the rule and A_{i+1} is a subformula of the formula $F[\mathbf{y}/\mathbf{z}]$ obtained by the unfolding, in which case i is said to be a *progress point* of the trace.

$$\frac{\frac{x \mapsto y * \text{ls } yz \vdash x \mapsto y * \text{ls } yz}{x \mapsto y * \text{ls } yz \vdash \text{ls } xz} \quad \frac{x \mapsto w \vdash x \mapsto w \quad \overline{\text{ls } wy * \text{ls } yz \vdash \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash x \mapsto w * \text{ls } wz}}{x \mapsto w * \text{ls } wy * \text{ls } yz \vdash \text{ls } xz}$$

$$\frac{\text{ls } xy * \text{ls } yz \vdash \text{ls } xz}{}$$

Soundness Condition

Definition (Infinite Progress)

An *infinitely progressing trace* is a trace having infinitely many progress points.

Soundness Condition

Definition (Infinite Progress)

An *infinitely progressing trace* is a trace having infinitely many progress points.

Definition (Cyclic proof)

A pre-proof \mathcal{P} is a *cyclic proof* if it satisfies the *global trace condition*: for every infinite path $(F_i \vdash G_i)_{i \geq 0}$ in \mathcal{P} , there is an infinitely progressing trace following some tail $(F_i \vdash G_i)_{i \geq n}$ of the path.

Outline

- 1 Introduction
 - Separation Logic
 - Cyclic Proofs
- 2 Theory
 - Formal System
 - **Soundness**
- 3 Implementation
 - Augmented Proof Rules
 - Soundness
 - Proof Search

Soundness of Cyclic Proof

Theorem (Soundness)

If there is a cyclic proof of $F \vdash G$, then $F \models G$.

Soundness of Cyclic Proof

Theorem (Soundness)

If there is a cyclic proof of $F \vdash G$, then $F \models G$.

Proof (Intuitive).

Suppose for contradiction that $F \not\models G$

Soundness of Cyclic Proof

Theorem (Soundness)

If there is a cyclic proof of $F \vdash G$, then $F \models G$.

Proof (Intuitive).

Suppose for contradiction that $F \not\models G$

By local soundness, there is an infinite path with $F_i \not\models G_i, i \geq 0$

Soundness of Cyclic Proof

Theorem (Soundness)

If there is a cyclic proof of $F \vdash G$, then $F \models G$.

Proof (Intuitive).

Suppose for contradiction that $F \not\models G$

By local soundness, there is an infinite path with $F_i \not\models G_i, i \geq 0$

So for all $i \geq 0$ there is a $h_i \in F_i$

Soundness of Cyclic Proof

Theorem (Soundness)

If there is a cyclic proof of $F \vdash G$, then $F \models G$.

Proof (Intuitive).

Suppose for contradiction that $F \not\models G$

By local soundness, there is an infinite path with $F_i \not\models G_i, i \geq 0$

So for all $i \geq 0$ there is a $h_i \in F_i$

And there is an infinitely progressing trace from some $n \geq 0$

Soundness of Cyclic Proof

Theorem (Soundness)

If there is a cyclic proof of $F \vdash G$, then $F \models G$.

Proof (Intuitive).

Suppose for contradiction that $F \not\models G$

By local soundness, there is an infinite path with $F_i \not\models G_i$, $i \geq 0$

So for all $i \geq 0$ there is a $h_i \in F_i$

And there is an infinitely progressing trace from some $n \geq 0$

So for $i \geq n$, h_{i+1} contains an unfolding of h_i infinitely often

Soundness of Cyclic Proof

Theorem (Soundness)

If there is a cyclic proof of $F \vdash G$, then $F \models G$.

Proof (Intuitive).

Suppose for contradiction that $F \not\models G$

By local soundness, there is an infinite path with $F_i \not\models G_i, i \geq 0$

So for all $i \geq 0$ there is a $h_i \in F_i$

And there is an infinitely progressing trace from some $n \geq 0$

So for $i \geq n$, h_{i+1} contains an unfolding of h_i infinitely often

So h_n cannot be a finite structure □

Soundness of Cyclic Proof

Theorem (Soundness)

If there is a cyclic proof of $F \vdash G$, then $F \models G$.

Proof (Intuitive).

Suppose for contradiction that $F \not\models G$

By local soundness, there is an infinite path with $F_i \not\models G_i, i \geq 0$

So for all $i \geq 0$ there is a $h_i \in F_i$

And there is an infinitely progressing trace from some $n \geq 0$

So for $i \geq n$, h_{i+1} contains an unfolding of h_i infinitely often

So h_n cannot be a finite structure □

Full details in James Brotherston's theses...

Outline

- 1 Introduction
 - Separation Logic
 - Cyclic Proofs
- 2 Theory
 - Formal System
 - Soundness
- 3 **Implementation**
 - **Augmented Proof Rules**
 - Soundness
 - Proof Search

Implemented Proof System

The implemented proof system uses augmented sequents

$$(\alpha, \pi) : F \vdash G$$

Implemented Proof System

The implemented proof system uses augmented sequents

$$(\alpha, \pi) : F \vdash G$$

α is called the *ancestry* of the current node. It records the entire branch from the root of the proof tree to the node, in the form of a (finite) list of entailments $F_1 \vdash G_1, \dots, F_n \vdash G_n$, with $F_n \vdash G_n$ being the root of the tree.

Implemented Proof System

The implemented proof system uses augmented sequents

$$(\alpha, \pi) : F \vdash G$$

α is called the *ancestry* of the current node. It records the entire branch from the root of the proof tree to the node, in the form of a (finite) list of entailments $F_1 \vdash G_1, \dots, F_n \vdash G_n$, with $F_n \vdash G_n$ being the root of the tree.

$\pi \in \mathbb{N}$, called the *progress pointer*, is the smallest natural number n such that α_n is the conclusion of a case-split rule.

Implemented Proof System

The implemented proof system uses augmented sequents

$$(\alpha, \pi) : F \vdash G$$

α is called the *ancestry* of the current node. It records the entire branch from the root of the proof tree to the node, in the form of a (finite) list of entailments $F_1 \vdash G_1, \dots, F_n \vdash G_n$, with $F_n \vdash G_n$ being the root of the tree.

$\pi \in \mathbb{N}$, called the *progress pointer*, is the smallest natural number n such that α_n is the conclusion of a case-split rule. If no such n exists, we set $\pi = |\alpha| + 1$, so that π points past the end of the ancestry.

Tagged Predicates

Each occurrence of an inductive predicate on the left of the turnstile is given a tag (an integer)

Tagged Predicates

Each occurrence of an inductive predicate on the left of the turnstile is given a tag (an integer)

The tags are preserved by rule applications, so they correspond to traces

Translating Proof Rules I

The general transformation is:

$$\frac{S_1 \dots S_n}{S} \implies \frac{(S :: \alpha, \pi + 1) : S_1 \dots (S :: \alpha, \pi + 1) : S_n}{(\alpha, \pi) : S}$$

Translating Proof Rules I

The general transformation is:

$$\frac{S_1 \dots S_n}{S} \implies \frac{(S :: \alpha, \pi + 1) : S_1 \dots (S :: \alpha, \pi + 1) : S_n}{(\alpha, \pi) : S}$$

I.e., when applying a rule backwards, the sequent in the conclusion of the rule is added to the ancestry of each of its premises.

Translating Proof Rules I

The general transformation is:

$$\frac{S_1 \dots S_n}{S} \implies \frac{(S :: \alpha, \pi + 1) : S_1 \dots (S :: \alpha, \pi + 1) : S_n}{(\alpha, \pi) : S}$$

I.e., when applying a rule backwards, the sequent in the conclusion of the rule is added to the ancestry of each of its premises.

The progress pointer is incremented, because the distance from the current node to the nearest conclusion of a case-split rule has increased by one (reading the rule from conclusion to premise).

Translating Proof Rules II

The case-split rules are exceptions.

Translating Proof Rules II

The case-split rules are exceptions.

When a case-split rule is applied, the progress pointer is set to 1 in each of its premises.

Translating Proof Rules II

The case-split rules are exceptions.

When a case-split rule is applied, the progress pointer is set to 1 in each of its premises.

So, for example, the implemented version of the case-split rule (Case 1_S) looks like this:

$$\frac{\begin{array}{l} ((G * 1_{S_j} \mathbf{v} \mathbf{v}' \vdash H) :: \alpha, 1) : G[y/v, y/v'] * \mathbf{emp} \vdash H[y/v, y/v'] \\ ((G * 1_{S_j} \mathbf{v} \mathbf{v}' \vdash H) :: \alpha, 1) : G[y/v, y'/v'] * y \mapsto y'' * 1_{S_j} y'' y' \vdash H[y/v, y'/v'] \end{array}}{(\alpha, \pi) : G * 1_{S_j} \mathbf{v} \mathbf{v}' \vdash H}$$

Translating Proof Rules II

The case-split rules are exceptions.

When a case-split rule is applied, the progress pointer is set to 1 in each of its premises.

So, for example, the implemented version of the case-split rule (Case ls) looks like this:

$$\frac{\begin{array}{l} ((G * \text{ls}_i v v' \vdash H) :: \alpha, 1) : G[y/v, y/v'] * \mathbf{emp} \vdash H[y/v, y/v'] \\ ((G * \text{ls}_i v v' \vdash H) :: \alpha, 1) : G[y/v, y'/v'] * y \mapsto y'' * \text{ls}_i y'' y' \vdash H[y/v, y'/v'] \end{array}}{(\alpha, \pi) : G * \text{ls}_i v v' \vdash H}$$

The subscript i on ls is the tag assigned to the atomic formula occurrence.

Translating Proof Rules II

The case-split rules are exceptions.

When a case-split rule is applied, the progress pointer is set to 1 in each of its premises.

So, for example, the implemented version of the case-split rule (Case $\mathbb{1}_S$) looks like this:

$$\frac{\begin{array}{l} ((G * \mathbb{1}_{S_i} v v' \vdash H) :: \alpha, 1) : G[y/v, y/v'] * \mathbf{emp} \vdash H[y/v, y/v'] \\ ((G * \mathbb{1}_{S_i} v v' \vdash H) :: \alpha, 1) : G[y/v, y'/v'] * y \mapsto y'' * \mathbb{1}_{S_i} y'' y' \vdash H[y/v, y'/v'] \end{array}}{(\alpha, \pi) : G * \mathbb{1}_{S_i} v v' \vdash H}$$

The subscript i on $\mathbb{1}_S$ is the tag assigned to the atomic formula occurrence.

The subformula $\mathbb{1}_S y'' y'$ in the second premise, obtained by unfolding $\mathbb{1}_S v v'$ in the conclusion, inherits the tag i , in keeping with the rules for forming traces.

Translating Proof Rules II

The axiom rule (Id) is the other exception

Translating Proof Rules II

The axiom rule (Id) is the other exception

Tags should be ignored when applying (Id).

Translating Proof Rules II

The axiom rule (Id) is the other exception

Tags should be ignored when applying (Id).

We define a binary predicate `matches` on formulas to implement equality up to change of tags.

Translating Proof Rules II

The axiom rule (Id) is the other exception

Tags should be ignored when applying (Id).

We define a binary predicate `matches` on formulas to implement equality up to change of tags.

The implemented form of (Id) is then as follows:

$$\frac{F \text{ matches } F'}{(\alpha, \pi) : F \vdash F'} \text{ (c_Id)}$$

Translating Proof Rules IV

Finally, we need to add a rule that allows us to form cycles.

Translating Proof Rules IV

Finally, we need to add a rule that allows us to form cycles.

The ancestry information is enough to form cycles, but the progress pointer allows us to form cycles with at least one progress point:

Translating Proof Rules IV

Finally, we need to add a rule that allows us to form cycles.

The ancestry information is enough to form cycles, but the progress pointer allows us to form cycles with at least one progress point:

To find a companion for $(\alpha, \pi) : F \vdash G$, it suffices to find a substitution θ and an n such that $n > \pi$, α_n is defined and $\alpha_n = (F \vdash G)[\theta]$.

Translating Proof Rules IV

Finally, we need to add a rule that allows us to form cycles.

The ancestry information is enough to form cycles, but the progress pointer allows us to form cycles with at least one progress point:

To find a companion for $(\alpha, \pi) : F \vdash G$, it suffices to find a substitution θ and an n such that $n > \pi$, α_n is defined and $\alpha_n = (F \vdash G)[\theta]$.

However, G and the right hand side of α_n can differ on tags.

Translating Proof Rules IV

Thus, the proof rule for link formation in the implemented system is

$$\frac{|\alpha| > n > \pi \quad \exists \theta. \alpha_n = (F \vdash G')[\theta] \quad G \text{ matches } G'[\theta]}{(\alpha, \pi) : F \vdash G}$$

Translating Proof Rules IV

Thus, the proof rule for link formation in the implemented system is

$$\frac{|\alpha| > n > \pi \quad \exists \theta. \alpha_n = (F \vdash G')[\theta] \quad G \text{ matches } G'[\theta]}{(\alpha, \pi) : F \vdash G}$$

This rule ensures that if we can form a downlink from B to C then there is a progressing trace on the finite path $C \dots B$ in the proof tree.

Outline

- 1 Introduction
 - Separation Logic
 - Cyclic Proofs
- 2 Theory
 - Formal System
 - Soundness
- 3 **Implementation**
 - Augmented Proof Rules
 - **Soundness**
 - Proof Search

Soundness of the Implementation

There is a map \mathcal{E} from proofs in the implemented system to pre-proofs in the formal system.

Soundness of the Implementation

There is a map \mathcal{E} from proofs in the implemented system to pre-proofs in the formal system.

For a proof tree T in the implemented system, $\mathcal{E}(T) = (\mathcal{D}, \mathcal{R})$, where:

- \mathcal{D} is the derivation tree obtained by erasing ancestries, progress pointers and predicate tags and turning every conclusion of `(c_downlink)` into a bud of \mathcal{D} ;
- \mathcal{R} is a function from the buds of \mathcal{D} to suitable companions, built from the applications of `(c_downlink)`.

Soundness of the Implementation

There is a map \mathcal{E} from proofs in the implemented system to pre-proofs in the formal system.

For a proof tree T in the implemented system, $\mathcal{E}(T) = (\mathcal{D}, \mathcal{R})$, where:

- \mathcal{D} is the derivation tree obtained by erasing ancestries, progress pointers and predicate tags and turning every conclusion of `(c_downlink)` into a bud of \mathcal{D} ;
- \mathcal{R} is a function from the buds of \mathcal{D} to suitable companions, built from the applications of `(c_downlink)`.

Soundness holds, since for every proof P in the implemented system, the pre-proof $\mathcal{E}(P)$ is actually a cyclic proof.

Soundness of the Implementation

Theorem (Soundness of the implementation)

If there is a proof of $([], 1) : F \vdash G$ in the implemented system, then $F \models G$.

Soundness of the Implementation

Proof (Sketch).

Given a proof P of $([], 1) : F \vdash G$, $\mathcal{E}(P)$ is clearly a pre-proof by construction.

Soundness of the Implementation

Proof (Sketch).

Given a proof P of $([], 1) : F \vdash G$, $\mathcal{E}(P)$ is clearly a pre-proof by construction.

Let $(S_i)_{i \geq 0}$ be an infinite path in $\mathcal{E}(P)$.

Soundness of the Implementation

Proof (Sketch).

Given a proof P of $([], 1) : F \vdash G$, $\mathcal{E}(P)$ is clearly a pre-proof by construction.

Let $(S_i)_{i \geq 0}$ be an infinite path in $\mathcal{E}(P)$.

There is a non-empty (finite) set \mathbf{B} of buds which are visited infinitely often on $(S_i)_{i \geq n}$.

Soundness of the Implementation

Proof (Sketch).

Given a proof P of $([], 1) : F \vdash G$, $\mathcal{E}(P)$ is clearly a pre-proof by construction.

Let $(S_i)_{i \geq 0}$ be an infinite path in $\mathcal{E}(P)$.

There is a non-empty (finite) set \mathbf{B} of buds which are visited infinitely often on $(S_i)_{i \geq n}$.

Choose $B \in \mathbf{B}$ such that $\mathcal{R}(B)$ is as close as possible to the root of \mathcal{D} .

Soundness of the Implementation

Proof (Sketch).

Given a proof P of $([], 1) : F \vdash G$, $\mathcal{E}(P)$ is clearly a pre-proof by construction.

Let $(S_i)_{i \geq 0}$ be an infinite path in $\mathcal{E}(P)$.

There is a non-empty (finite) set \mathbf{B} of buds which are visited infinitely often on $(S_i)_{i \geq n}$.

Choose $B \in \mathbf{B}$ such that $\mathcal{R}(B)$ is as close as possible to the root of \mathcal{D} .

By inspection of the (`c_downlink`) rule, there is some tagged atomic formula $P_i\mathbf{x}$ occurring in both $\mathcal{R}(B)$ and B whose case-split rule is applied on the path $\mathcal{R}(B) \dots B$.

Soundness of the Implementation

Proof (Sketch).

Given a proof P of $([], 1) : F \vdash G$, $\mathcal{E}(P)$ is clearly a pre-proof by construction.

Let $(S_i)_{i \geq 0}$ be an infinite path in $\mathcal{E}(P)$.

There is a non-empty (finite) set \mathbf{B} of buds which are visited infinitely often on $(S_i)_{i \geq n}$.

Choose $B \in \mathbf{B}$ such that $\mathcal{R}(B)$ is as close as possible to the root of \mathcal{D} .

By inspection of the (`c_downlink`) rule, there is some tagged atomic formula $P_i\mathbf{x}$ occurring in both $\mathcal{R}(B)$ and B whose case-split rule is applied on the path $\mathcal{R}(B) \dots B$.

There must be an infinitely progressing trace following $(S_i)_{i \geq n}$, with all predicates tagged by i . □

Soundness of the Implementation

Proof (Continued).

It exists because (`c_downlink`) requires identical tags on the left and rules preserve tags.

Soundness of the Implementation

Proof (Continued).

It exists because $(c_downlink)$ requires identical tags on the left and rules preserve tags.

It is infinitely progressing because $(c_downlink)$ ensures a progress point. □

Outline

- 1 Introduction
 - Separation Logic
 - Cyclic Proofs
- 2 Theory
 - Formal System
 - Soundness
- 3 **Implementation**
 - Augmented Proof Rules
 - Soundness
 - **Proof Search**

Split Entailments

To better manage the sizes of the generated proofs, the entailment relation has been split into two: the augmented entailment $(\alpha, \pi) : P \vdash Q$ and a basic one $P \vdash_{basic} Q$ which is not augmented (and so \vdash_{basic} is actually a subset of \vdash).

Split Entailments

To better manage the sizes of the generated proofs, the entailment relation has been split into two: the augmented entailment $(\alpha, \pi) : P \vdash Q$ and a basic one $P \vdash_{basic} Q$ which is not augmented (and so \vdash_{basic} is actually a subset of \vdash).

The idea is to relay all reasoning using the associativity, commutativity and unit of $*$ to \vdash_{basic} . Such rules as (**empR**) are then found in this lightweight entailment rather than in the augmented one.

Split Entailments

To better manage the sizes of the generated proofs, the entailment relation has been split into two: the augmented entailment $(\alpha, \pi) : P \vdash Q$ and a basic one $P \vdash_{basic} Q$ which is not augmented (and so \vdash_{basic} is actually a subset of \vdash).

The idea is to relay all reasoning using the associativity, commutativity and unit of $*$ to \vdash_{basic} . Such rules as (**empR**) are then found in this lightweight entailment rather than in the augmented one.

It is important that \vdash_{basic} does not interfere with the predicate tags, and so it is limited to reorganizing terms. Its id-rule, for instance, does not use the `matches`-predicate, and there is no cut rule.

Split Entailments

For the augmented entailment to make use of lightweight entailment rules such as (**empR**), we provide cut rules to inject \vdash_{basic} -reasoning into our proofs:

$$\frac{P \vdash_{basic} R \quad (\alpha, \pi) : R \vdash Q}{(\alpha, \pi) : P \vdash Q} \text{ (basicL)}$$

$$\frac{(\alpha, \pi) : P \vdash R \quad R \vdash_{basic} Q}{(\alpha, \pi) : P \vdash Q} \text{ (basicR)}$$

Split Entailments

For the augmented entailment to make use of lightweight entailment rules such as (**empR**), we provide cut rules to inject \vdash_{basic} -reasoning into our proofs:

$$\frac{P \vdash_{basic} R \quad (\alpha, \pi) : R \vdash Q}{(\alpha, \pi) : P \vdash Q} \text{ (basicL)}$$

$$\frac{(\alpha, \pi) : P \vdash R \quad R \vdash_{basic} Q}{(\alpha, \pi) : P \vdash Q} \text{ (basicR)}$$

Tactics

Our prover is a collection of HOL Light tactics arranged into layers:

- 1 A tactic for each rule in the implemented system

Tactics

Our prover is a collection of HOL Light tactics arranged into layers:

- 1 A tactic for each rule in the implemented system
- 2 Tactics using \vdash_{basic} -reasoning to prepare rule applications

Tactics

Our prover is a collection of HOL Light tactics arranged into layers:

- 1 A tactic for each rule in the implemented system
- 2 Tactics using \vdash_{basic} -reasoning to prepare rule applications
- 3 Tactics which only succeed if they “advance”

Tactics

Our prover is a collection of HOL Light tactics arranged into layers:

- 1 A tactic for each rule in the implemented system
- 2 Tactics using \vdash_{basic} -reasoning to prepare rule applications
- 3 Tactics which only succeed if they “advance”

This facilitates interactive use.

We implemented a backtracking proof search which applies any rule it can, from a prioritized list of rule sets:

1. $\{(c_Id), \text{link formation}\}$
2. advancing right rules
3. case-split rules

Tactics

Our prover is a collection of HOL Light tactics arranged into layers:

- 1 A tactic for each rule in the implemented system
- 2 Tactics using \vdash_{basic} -reasoning to prepare rule applications
- 3 Tactics which only succeed if they “advance”

This facilitates interactive use.

We implemented a backtracking proof search which applies any rule it can, from a prioritized list of rule sets:

1. $\{(c_Id), \text{link formation}\}$
2. advancing right rules
3. case-split rules

The other rules are only invoked as part of auxiliary reasoning for the rules in these groups.

Tactics: Layer 1

There is a tactic for each rule of the implemented proof system, and tactics are generated for the unfolding rules given by the inductive definitions.

The left rules introduce fresh variables and perform the (potentially unifying) substitutions, while the right rules introduce existential metavariables for any extra exposed variables.

Tactics: Layer 3

Right-unfolding rules, for instance, typically expose new state on the right side.

Tactics: Layer 3

Right-unfolding rules, for instance, typically expose new state on the right side.

An *advancing* version of such a rule will try to match this on the left hand side (resolving existential metavariables if necessary) and invoke a tactic to eliminate common state; the entire rule application fails if no state can be disposed of.

Tactics: Layer 3

Right-unfolding rules, for instance, typically expose new state on the right side.

An *advancing* version of such a rule will try to match this on the left hand side (resolving existential metavariables if necessary) and invoke a tactic to eliminate common state; the entire rule application fails if no state can be disposed of.

Elimination of common state is implemented using \vdash_{basic} -reasoning to bring both sides to similar forms and then using the rules (*) and a version of (c_Id) which resolves existential metavariables.

Cyclic Proofs More Natural?

I find the case splitting reasoning quite natural :)

Cyclic Proofs More Natural?

I find the case splitting reasoning quite natural :)

But consider the alternative, inductive proof of

$$\text{ls } x \ x' * \text{ls } x' \ y \vdash \text{ls } x \ y$$

Cyclic Proofs More Natural?

I find the case splitting reasoning quite natural :)

But consider the alternative, inductive proof of

$$\text{ls } x \ x' * \text{ls } x' \ y \vdash \text{ls } x \ y$$

It is by induction on $\text{ls } x \ x'$ using the induction hypothesis

$$\text{ls } x' \ y \multimap \text{ls } x \ y$$

Cyclic Proofs More Natural?

I find the case splitting reasoning quite natural :)

But consider the alternative, inductive proof of

$$\text{ls } x \ x' * \text{ls } x' \ y \vdash \text{ls } x \ y$$

It is by induction on $\text{ls } x \ x'$ using the induction hypothesis

$$\text{ls } x' \ y \multimap \text{ls } x \ y$$

Not only is this induction hypothesis not a subformula of the goal sequent, but it is not even expressible in our formula language (or that of most available verification tools).

Summary

We have presented a method for automatically answering Separation Logic entailment questions using Cyclic Proofs.

Summary

We have presented a method for automatically answering Separation Logic entailment questions using Cyclic Proofs.

(Hoping to demonstrate that Cyclic Proof is a viable method for dealing with inductive predicates.)