Automated Reasoning in Separation Logic with Inductive Definitions

Cristina Serban
Supervised by Radu Iosif

Université de Grenoble Alpes, VERIMAG

Thesis committee

Nikos Gorogiannis, Middlesex University London
Etienne Lozes, Université Nice Sophia Antipolis
Wei-Ngan Chin, National University of Singapore
Didier Galmiche, Université de Lorraine
Mihaela Sighireanu, Université Paris Diderot
Mnacho Echenim, Grenoble INP, LIG

May 31, 2018
Outline

1 Motivation and contributions

2 Inductive systems of predicates
   - The entailment problem

3 Proof systems for inductive entailments
   - Downwards inclusion check for tree automata
   - Inference rules
   - Restricting the set of constraints for soundness and completeness

4 Decision procedures for Separation Logic

5 Implementation and experimental evaluation

6 Conclusions and future work
Given a program \( P \) and a set of initial program states \( I \)
is there an execution of \( P \) starting from a state in \( I \)
and ending in a state from a *bad* set \( B \)?

States in \( B \) may be reached by performing an *anomalous action*
(e.g. dereferencing an unallocated pointer or memory leak)
or are incorrect w.r.t. to the intended functionality of \( P \)

Proving program correctness: annotate \( P \) with *assertions* such that
- Whenever an assertion is reached, it holds for the current state of \( P \)
- The bad set \( B \) is excluded from the states reachable by \( P \)
Program verification

This process has been formalized by Hoare logic, reasoning about triples

\[ \{ \phi \} \ c \ \{ \psi \} \]

precondition \( \phi \) holds \( \rightarrow \) \( c \) is executed \( \rightarrow \) postcondition \( \psi \) holds
Program verification

This process has been formalized by Hoare logic, reasoning about triples

\[
\{ \phi \} \ c \ \{ \psi \}
\]

precondition \( \phi \) holds \(\rightarrow\) \( c \) is executed \(\rightarrow\) postcondition \( \psi \) holds

Using deduction rules, we can automatically compute

- Strongest postcondition: \( \text{sp}(c, \phi) \)
- Weakest precondition: \( \text{wp}(c, \psi) \)

Given a specification \( \{ \phi \} \ P \ \{ \psi \} \), we obtain verification conditions

\[
\text{sp}(P, \phi) \models \psi \quad \phi \models \text{wp}(P, \psi)
\]
Program verification

\( \{ x = x_0 \land y = y_0 \} \text{ swap}(x, y) \{ x = y_0 \land y = x_0 \} \)
Program verification

\[
\{ x = x_0 \land y = y_0 \} \text{swap}(x, y) \{ x = y_0 \land y = x_0 \}
\]

\text{swap}(x, y)

\[
\begin{align*}
x &:= x + y \\
y &:= x - y \\
x &:= x - y
\end{align*}
\]
Program verification

\{ x = x_0 \land y = y_0 \} \text{swap}(x, y) \{ x = y_0 \land y = x_0 \}

\text{swap}(x, y)
\{ x = x_0 \land y = y_0 \} \quad \text{Precondition}
\begin{align*}
x & := x + y \\
y & := x - y \\
x & := x - y
\end{align*}
Program verification

\[ \{ x = x_0 \land y = y_0 \} \text{ swap}(x, y) \{ x = y_0 \land y = x_0 \} \]

\text{swap}(x, y)
\[\begin{align*}
\{ x = x_0 \land y = y_0 \} & \quad \text{Precondition} \\
x & := x + y \\
\{ x = x_0 + y_0 \land y = y_0 \} & \\
y & := x - y \\
x & := x - y
\end{align*}\]
Program verification

\{x = x_0 \land y = y_0\} \ \text{swap}(x, y) \ \{x = y_0 \land y = x_0\}

\text{swap}(x, y)
\begin{align*}
\{x = x_0 \land y = y_0\} \\
& \quad \text{Precondition} \\
& x := x + y \\
\{x = x_0 + y_0 \land y = y_0\} \\
& y := x - y \\
\{x = x_0 + y_0 \land y = x_0 + y_0 - y_0\} \\
& x := x - y
\end{align*}
Program verification

\{x = x_0 \land y = y_0\} \text{ swap}(x, y) \{x = y_0 \land y = x_0\}

\text{swap}(x, y)
\begin{align*}
\{x = x_0 \land y = y_0\} & \\
x & := x + y & \text{Precondition} \\
\{x = x_0 + y_0 \land y = y_0\} & \\
y & := x - y \\
\{x = x_0 + y_0 \land y = x_0 + y_0 - y_0\} & \\
x & := x - y & \text{Strongest postcondition} \\
\{x = x_0 + y_0 - (x_0 + y_0 - y_0) \land y = x_0 + y_0 - y_0\} & \\
\end{align*}
Program verification

\[
\{ x = x_0 \land y = y_0 \} \text{ swap}(x, y) \{ x = y_0 \land y = x_0 \}
\]

\text{swap}(x, y)

\[
\{ x = x_0 \land y = y_0 \}
\]

\text{Precondition}

\[
x := x + y
\]

\[
\{ x = x_0 + y_0 \land y = y_0 \}
\]

\[
y := x - y
\]

\[
\{ x = x_0 + y_0 \land y = x_0 + y_0 - y_0 \}
\]

\text{Strongest postcondition}

\[
x := x - y
\]

\[
\{ x = x_0 + y_0 - (x_0 + y_0 - y_0) \land y = x_0 + y_0 - y_0 \}
\]

\[
\text{sp}(\text{swap}(x, y), x = x_0 \land y = y_0) \models (x = y_0 \land y = x_0) \text{ holds}
\]
Reasoning about dynamically allocated memory

Most programs use dynamic memory and mutable data structures

Coarse memory abstractions can lead to complex verification conditions, e.g. properties of untouched memory carried through a function body
Reasoning about dynamically allocated memory

Most programs use dynamic memory and mutable data structures

Coarse memory abstractions can lead to complex verification conditions, e.g. properties of untouched memory carried through a function body

Compositional methods required to ensure scalability

- **Local reasoning** - specifications and proofs for a program component refer only to the memory that component accesses (footprint)

- **Modularity** - analysing components in isolation and combining the results to obtain a global verification condition
Separation Logic (SL)

Two types of connectives:
- Additive: $\land$, $\rightarrow$ (classic conjunction and implication)
- Multiplicative: $\ast$, $\ast$ (separating conjunction and implication)

Memory (heap) – a resource of finite volume which can be split

Resource interpretation of the separating connectives:
- $\ast$ decomposes current resources
- $\ast$ talks about new or fresh resources
Separation Logic (SL)

Two types of connectives:

- **Additive**: $\land$, $\to$ (classic conjunction and implication)
- **Multiplicative**: $\ast$, $\rightarrow$ (separating conjunction and implication)

\[
\phi_1 \land \phi_2 \models \psi \iff \phi_1 \models \phi_2 \to \psi \\
\phi_1 \ast \phi_2 \models \psi \iff \phi_1 \models \phi_2 \ast \psi
\]

**but**

\[
\phi \models \phi \land \phi \\
\phi \not\models \phi \ast \phi \\
\phi \ast \psi \not\models \phi
\]
SL features – Frame rule for compositional reasoning

The ability to extend local specifications regained at a deeper level with *

\[
\begin{array}{c}
\{\phi\} \textbf{c} \{\psi\} \\
\hline
\{\phi \ast \varphi\} \textbf{c} \{\psi \ast \varphi\}
\end{array}
\]

\(\phi\) and \(\psi\) refer to the footprint of \(\textbf{c}\), while \(\varphi\) involves only variables and heap cells not modified or mutated by \(\textbf{c}\)
Recursive data structures are ubiquitous in real-life software.

Described by inductively defined predicates, which can also specify the shape of the memory where they are dynamically allocated.

Simplest example of an inductive predicate: natural numbers, using the successor function $\text{succ}$ from Peano arithmetic.

\[
\text{Natural}(x) ::= x = 0 \lor \exists y. x = \text{succ}(y) \land \text{Natural}(y)
\]
Building a list segment $ls(x, y)$

- $empty$

\[ emp \land x = y \]
Building a list segment $ls(x, y)$

$\text{empty}$

$emp \land x = y$

$x \mapsto u$

$x \mapsto u$
Building a list segment $ls(x, y)$

\[ \text{empty} \]

\[ \text{emp} \land x = y \]

\[ x \mapsto u \quad ls(u, y) \]

\[ x \rightarrow u \rightarrow \ldots \rightarrow y \]
Building a list segment \( ls(x, y) \)

\[ \text{empty} \]

\[ \text{emp} \land x = y \]

\[ x \mapsto u \ast ls(u, y) \]
Building a list segment $ls(x, y)$

\[\begin{align*}
\text{empty} & \quad \text{emp} \land x = y \\
x & \mapsto u \ast ls(u, y)
\end{align*}\]

\[ls(x, y) ::= \text{emp} \land x = y \lor \exists u . x \mapsto u \ast ls(u, y)\]
Building a list segment \( ls(x, y) \)

\[
\begin{align*}
\text{empty} & \quad \text{emp} \land x = y \\
x & \mapsto u \ast ls(u, y)
\end{align*}
\]

\[
ls(x, y) ::= \text{emp} \land x = y \lor \exists u. x \mapsto u \ast ls(u, y)
\]

\( \text{list segment} \)
Building a list segment \( ls(x, y) \)

**Empty**

\[
\text{emp} \land x = y
\]

\[
x \mapsto u \ast ls(u, y)
\]

**List Segment**

\[
ls(x, y) ::= \text{emp} \land x = y \lor \exists u . x \mapsto u \ast ls(u, y)
\]

**Acyclic List Segment**

\[
ls^a(x, y) ::= \text{emp} \land x = y \lor x \neq y \land \exists u . x \mapsto u \ast ls^a(u, y)
\]
Building a list segment \( ls(x, y) \)

\[
\begin{align*}
\text{empty} & \quad \text{emp} \land x = y \\
\text{ls}(x, y) & := \text{emp} \land x = y \lor \exists u . x \mapsto u \ast \text{ls}(u, y) \\
\text{ls}^a(x, y) & := \text{emp} \land x = y \lor x \neq y \land \exists u . x \mapsto u \ast \text{ls}^a(u, y)
\end{align*}
\]
Program verification with Separation Logic

\[
\{ x \neq y \land ls(x, y) \} \text{deleteStart}(x, y) \{ ls(x, y) \}
\]

deleteStart(x, y)

\[
z := x.next
\]

dispose(x)

\[
x := z
\]

*Bi as an Assertion Language for Mutable Data Structures*

[S. Ishtiaq, P. O'Hearn – POPL 2001]
Program verification with Separation Logic

\[ \{ x \neq y \land ls(x, y) \} \text{ deleteStart}(x, y) \{ ls(x, y) \} \]

deleteStart(x, y)

\[
z := x.\text{next}
\]

dispose(x)

\[
x := z
\]
\[
\{ ls(x, y) \}
\]

Postcondition

*BI as an Assertion Language for Mutable Data Structures*

[S. Ishtiaq, P. O'Hearn – POPL 2001]
Program verification with Separation Logic

\[ \{ x \neq y \land ls(x, y) \} \text{ deleteStart}(x, y) \{ ls(x, y) \} \]

deleteStart(x, y)

\[ z := x.\text{next} \]

dispose(x)

\[ \{ ls(z, y) \} \]

\[ x := z \]

\[ \{ ls(x, y) \} \]

Postcondition

*BI as an Assertion Language for Mutable Data Structures*
[S. Ishtiaq, P. O'Hearn – POPL 2001]*
Program verification with Separation Logic

\{x \neq y \land ls(x, y)\} \text{deleteStart}(x, y) \{ls(x, y)\}

deleteStart(x, y)

z := x.next
\{\exists x_0. x \mapsto x_0 * ls(z, y)\}

dispose(x)
\{ls(z, y)\}
x := z
\{ls(x, y)\}

Postcondition

B\i as an Assertion Language for Mutable Data Structures
[S. Ishtiaq, P. O'Hearn – POPL 2001]
Program verification with Separation Logic

\[ \{ x \neq y \land ls(x, y) \} \text{ deleteStart}(x, y) \{ ls(x, y) \} \]

deleteStart(x, y)
\[ \{ \exists z_0 . (\exists x_0 . x \mapsto x_0 \ast ls(z_0, y) \land x \mapsto z_0 \ast \top) \} \] Weakest precondition
\[ z := x.\text{next} \]
\[ \{ \exists x_0 . x \mapsto x_0 \ast ls(z, y) \} \]
dispose(x)
\[ \{ ls(z, y) \} \]
x := z
\[ \{ ls(x, y) \} \] Postcondition

Bl as an Assertion Language for Mutable Data Structures
[S. Ishtiaq, P. O'Hearn – POPL 2001]
Program verification with Separation Logic

\{x \neq y \land ls(x, y)\} \text{deleteStart}(x, y) \{ls(x, y)\}

deleteStart(x, y)
\{\exists z_0. (\exists x_0. x \mapsto x_0 \ast ls(z_0, y) \land x \mapsto z_0 \ast \top)\} \quad \text{Weakest precondition}

z := x.next
\{\exists x_0. x \mapsto x_0 \ast ls(z, y)\}

dispose(x)
\{ls(z, y)\}

x := z
\{ls(x, y)\} \quad \text{Postcondition}

\exists z_0. (\exists x_0. x \mapsto x_0 \ast ls(z_0, y) \land x \mapsto z_0 \ast \top) \equiv \exists z_0. x \mapsto z_0 \ast ls(z_0, y)

x \neq y \land ls(x, y) \models \text{wp(deleteStart}(x, y), ls(x, y)) \text{ holds}

\textit{BI as an Assertion Language for Mutable Data Structures}
[S. Ishtiaq, P. O'Hearn – POPL 2001]
Contributions

Non-inductive entailments in Separation Logic

- Checking the validity of an entailment $\phi \models \psi$ can be reduced to checking the unsatisfiability of $\phi \land \neg \psi$
- Decision procedures for the satisfiability of the quantifier free and the $\exists^* \forall^*$-quantified fragments of Separation Logic
- Implemented in the DPLL($T$)-based SMT solver CVC4


Contributions

Complete proof systems for inductive entailments

- Generic cyclic proof system for First-Order Logic, inspired by an antichain-based method for finite tree automata inclusion
- Reliant on Infinite Descent to close recurring proof branches
- Semantic boundaries for the inductive definitions within which the proof system remains sound and complete
- Proof system and semantic restrictions adapted to Separation Logic
Contributions

Inductor – an inductive entailment checker tool

- Implements our cyclic proof system for Separation Logic and queries our decision procedures from CVC4 for non-inductive entailments
- Provides proofs for valid inputs and counterexamples for invalid ones

- R. Iosif, C. Serban. An Entailment Checker for Separation Logic with Inductive Definitions – AVOCS 2018
Outline

1 Motivation and contributions

2 Inductive systems of predicates
   - The entailment problem

3 Proof systems for inductive entailments
   - Downwards inclusion check for tree automata
   - Inference rules
   - Restricting the set of constraints for soundness and completeness

4 Decision procedures for Separation Logic

5 Implementation and experimental evaluation

6 Conclusions and future work
Syntax of multisorted FOL

Let $\Sigma = (\Sigma^s, \Sigma^f)$ be a signature where:
- $\Sigma^s$ is a set of sort symbols $\sigma$, $\text{Bool} \in \Sigma^s$
- $\Sigma^f$ is a set of function symbols $f^{\sigma_1\ldots\sigma_n}$

Let $\text{Var} = \{x, y, z, \ldots\}$ be a countable set of first-order variables
Syntax of multisorted FOL

Let $\Sigma = (\Sigma^s, \Sigma^f)$ be a signature where:
- $\Sigma^s$ is a set of sort symbols $\sigma$, $\text{Bool} \in \Sigma^s$
- $\Sigma^f$ is a set of function symbols $f^{\sigma_1 \ldots \sigma_n \sigma}$

Let $\text{Var} = \{x, y, z, \ldots\}$ be a countable set of first-order variables

Terms: $t^\sigma ::= x^\sigma \in \text{Var} \mid c^\sigma \in \Sigma^f \mid f(t_1^{\sigma_1}, \ldots, t_n^{\sigma_n}), f^{\sigma_1 \ldots \sigma_n} \in \Sigma^f$

$\mid \langle t_1^{\sigma_1}, \ldots, t_n^{\sigma_n} \rangle, \sigma = \sigma_1 \times \ldots \times \sigma_n$

Formulae: $\phi_{\text{FOL}} ::= \top \mid \bot \mid t^{\text{Bool}} \mid t^\sigma \approx u^\sigma \mid \neg \psi_{\text{FOL}}$

$\mid \psi_1_{\text{FOL}} \land \psi_2_{\text{FOL}} \mid \psi_1_{\text{FOL}} \lor \psi_2_{\text{FOL}} \mid \exists x^\sigma. \psi_{\text{FOL}} \mid \forall x^\sigma. \psi_{\text{FOL}}$
Semantics of multisorted FOL

An interpretation $\mathcal{I}$ maps
- a sort $\sigma \in \Sigma^s$ to a set $\sigma^\mathcal{I}$
- a symbol $f^{\sigma_1 \ldots \sigma_n} \in \Sigma^f$ to a function $f^\mathcal{I} : \sigma_1^\mathcal{I} \times \ldots \times \sigma_n^\mathcal{I} \to \sigma^\mathcal{I}$

A valuation $\nu$ under $\mathcal{I}$ assigns a value $\nu(x^\sigma) \in \sigma^\mathcal{I}$ to each $x^\sigma \in \text{Var}$

We write $\mathcal{I}, \nu \models \phi$ whenever the result of replacing each $f$ with $f^\mathcal{I}$ and each $x$ with $\nu(x)$ in $\phi$ is true
Syntax of SL

\[ \Sigma^s = \{ \text{Loc}, \text{Data}, \text{Bool} \}, \quad \Sigma^f = \{ \text{nil}^{\text{Loc}} \} \text{ and } \text{Data} = \text{Loc}^k \text{ with } k \geq 1 \text{ fixed.} \]

We fix \( \mathcal{I} \) such that \( \mathcal{I}(\text{Loc}) = L \) is countably infinite and \( \text{nil}^\mathcal{I} = \ell_{\text{nil}} \in L \).
Syntax of SL

\[ \Sigma^s = \{ \text{Loc}, \text{Data}, \text{Bool} \}, \: \Sigma^f = \{ \text{nil}^{\text{Loc}} \} \text{ and } \text{Data} = \text{Loc}^k \text{ with } k \geq 1 \text{ fixed.} \]

We fix \( \mathcal{I} \) such that \( \mathcal{I} (\text{Loc}) = L \) is countably infinite and \( \text{nil}^\mathcal{I} = \ell_{\text{nil}} \in L \).

Formulae: \[ \phi^{\text{SL}} ::= \top \mid \bot \mid t^{\text{Bool}} \mid t^\sigma \approx u^\sigma \mid \neg \psi^{\text{SL}} \]
\[ \mid \psi_1^{\text{SL}} \land \psi_2^{\text{SL}} \mid \psi_1^{\text{SL}} \lor \psi_2^{\text{SL}} \mid \exists x^\sigma . \psi^{\text{SL}} \mid \forall x^\sigma . \psi^{\text{SL}} \]
\[ \mid \text{emp} \mid t^{\text{Loc}} \mapsto u^{\text{Data}} \mid \psi_1^{\text{SL}} \ast \psi_2^{\text{SL}} \mid \psi_1^{\text{SL}} \rightarrow \psi_2^{\text{SL}} \]
Semantics of SL

A **heap** is a mapping \( h : L \rightarrow \text{fin} L^k \) and Heaps is the set of heaps

\[
h_1 \not= h_2 \iff \text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset \quad h = h_1 \uplus h_2 \iff h_1 \not= h_2 \land h = h_1 \cup h_2
\]

We write \( \nu, h \models_{SL} \phi \) if \( \phi \) is interpreted to true under \( \nu \) and \( h \).

- \( \nu, h \models_{SL} \text{emp} \) iff \( h = \emptyset \)
- \( \nu, h \models_{SL} t \mapsto u \) iff \( h = \{(t^{\mathcal{I}}_\nu, u^{\mathcal{I}}_\nu)\} \) and \( t^{\mathcal{I}}_\nu \neq \ell_{nil} \)
- \( \nu, h \models_{SL} \phi_1 \ast \phi_2 \) iff \( \exists h_1, h_2. h = h_1 \uplus h_2 \) and \( \mathcal{I}, h_i \models_{SL} \phi_i, \forall i \in \{1, 2\} \)
- \( \nu, h \models_{SL} \phi_1 \rightarrow \phi_2 \) iff \( \forall h'. \text{ if } h' \not= h \text{ and } \nu, h' \models_{SL} \phi_1 \text{ then } \nu, h' \uplus h \models_{SL} \phi_2 \)
Systems of inductive definitions

Let Pred be a countable set of predicates $p^{\sigma_1 \ldots \sigma_n}$.

A **predicate rule** is a pair

\[
\langle p(\bar{x}), \{\phi(\bar{x}, \bar{x}_1, \ldots, \bar{x}_m), q_1(\bar{x}_1), \ldots, q_n(\bar{x}_m)\} \rangle
\]

- **goal**
- **body**
  - **constraint**
  - **subgoals**
Systems of inductive definitions

Let Pred be a countable set of predicates $p^{\sigma_1 \ldots \sigma_n}$.

A **predicate rule** is a pair

$$\langle p(\overline{x}), \{ \phi(\overline{x}, \overline{x}_1, \ldots, \overline{x}_m), q_1(\overline{x}_1), \ldots, q_n(\overline{x}_m) \} \rangle$$

A **constraint** and a **subgoals**

An **inductive system** $S$ is a finite set of rules and we write

$$p(\overline{x}) \leftarrow S R_1 \mid \ldots \mid R_\ell$$

when $\{ \langle p(\overline{x}), R_1 \rangle, \ldots, \langle p(\overline{x}), R_\ell \rangle \}$ are all the predicate rules for $p$ in $S$
The entailment problem

\[ F_S \text{ monotone and continuous function on assignments mapping each } p^{\sigma_1 \ldots \sigma_n} \in \text{Pred to a set of solutions } \mathcal{X}(p) \]

\[ \mathcal{X}(p) \subseteq \sigma_1^I \times \ldots \times \sigma_n^I \text{ (FOL)} \quad \mathcal{X}(p) \subseteq L^n \times \text{Heaps (SL)} \]

\[ \mu S = \bigcap \{ \mathcal{X} \mid F_S(\mathcal{X}) \subseteq \mathcal{X} \} \text{ least fixed point of } F_S \text{ and least solution of } S \]
The entailment problem

$\mathbb{F}_S$ monotone and continuous function on assignments mapping each $p^{\sigma_1\cdots\sigma_n} \in \text{Pred}$ to a set of solutions $\mathcal{X}(p)$

$$\mathcal{X}(p) \subseteq \sigma_1^I \times \cdots \times \sigma_n^I \quad \text{(FOL)} \quad \mathcal{X}(p) \subseteq \mathbb{L}^n \times \text{Heaps} \quad \text{(SL)}$$

$$\mu S = \bigcap\{\mathcal{X} \mid \mathbb{F}_S(\mathcal{X}) \subseteq \mathcal{X}\}$$ least fixed point of $\mathbb{F}_S$ and least solution of $S$

Given an inductive system $S$

and predicates $p^{\sigma_1\cdots\sigma_n}$ and $q_1^{\sigma_1\cdots\sigma_n}, \ldots, q_m^{\sigma_1\cdots\sigma_n}$,

does $\mu S(p) \subseteq \bigcup_{i=1}^m \mu S(q_i)$?

We denote entailment problems as

$$p \models_S^T q_1, \ldots, q_m \quad \text{(FOL)} \quad p \models_S^{SL} q_1, \ldots, q_m \quad \text{(SL)}$$
Entailments under the canonical interpretation

The **Herbrand (canonical) interpretation** $\mathcal{H}$ maps terms to trees

$$
c^\mathcal{H} \rightarrow \text{c} \quad f^\mathcal{H}(t_1^{\sigma_1}, \ldots, t_n^{\sigma_n}) \rightarrow f \quad (t_1)^\mathcal{H}_\nu \cdots (t_n)^\mathcal{H}_\nu
$$

**Theorem**

*The entailment problem is undecidable for inductive systems under the Herbrand interpretation.*
Representing nondeterministic finite tree automata

An NFTA can be naturally viewed as an inductive system

- states are represented by predicates
- predicate rules are obtained by translation from transition rules

Consider a ranked alphabet $\mathcal{F} = \{ f(, ), g(), a, b \}$

$A = \{ \{ p, p_1, p_2 \}, \mathcal{F}, \{ p \}, \Delta_A \}$ and $B = \{ \{ q, q_1, q_2 \}, \mathcal{F}, \{ q \}, \Delta_B \}$
Representing nondeterministic finite tree automata

An NFTA can be naturally viewed as an inductive system

- states are represented by predicates
- predicate rules are obtained by translation from transition rules

Consider a ranked alphabet $\mathcal{F} = \{ f(,), g(), a, b \}$

$A = \{ \{ p, p_1, p_2 \}, \mathcal{F}, \{ p \}, \Delta_A \}$ and $B = \{ \{ q, q_1, q_2 \}, \mathcal{F}, \{ q \}, \Delta_B \}$

$\Delta_A = \{ p \xrightarrow{f} (p_1, p_2) \}$
Representing nondeterministic finite tree automata

An NFTA can be naturally viewed as an inductive system
- states are represented by predicates
- predicate rules are obtained by translation from transition rules

Consider a ranked alphabet $\mathcal{F} = \{f(,), g(), a, b\}$

$$A = \{\{p, p_1, p_2\}, \mathcal{F}, \{p\}, \Delta_A\} \text{ and } B = \{\{q, q_1, q_2\}, \mathcal{F}, \{q\}, \Delta_B\}$$

$$\Delta_A = \{p \xrightarrow{f} (p_1, p_2), \quad p_1 \xrightarrow{g} p_1, \quad p_2 \xrightarrow{g} p_2\}$$
Representing nondeterministic finite tree automata

An NFTA can be naturally viewed as an inductive system

- states are represented by predicates
- predicate rules are obtained by translation from transition rules

Consider a ranked alphabet $\mathcal{F} = \{ f(,), g(), a, b \}$

$A = \{ \{ p, p_1, p_2 \}, \mathcal{F}, \{ p \}, \Delta_A \}$ and $B = \{ \{ q, q_1, q_2 \}, \mathcal{F}, \{ q \}, \Delta_B \}$

$\Delta_A = \{ p \xrightarrow{f} (p_1, p_2), \ p_1 \xrightarrow{g} p_1, \ p_1 \xrightarrow{a} (), \ p_2 \xrightarrow{g} p_2, \ p_2 \xrightarrow{b} () \}$

$\Delta_B = \{ q \xrightarrow{f} (q_1, q_2), \ q_1 \xrightarrow{g} q_1, \ q_1 \xrightarrow{a} (), \ q_2 \xrightarrow{g} q_2, \ q_2 \xrightarrow{b} () \}$
Representing nondeterministic finite tree automata

An NFTA can be naturally viewed as an inductive system
- states are represented by predicates
- predicate rules are obtained by translation from transition rules

Consider a ranked alphabet $\mathcal{F} = \{ f(,), g(), a, b \}$

$A = \{ \{ p, p_1, p_2 \}, \mathcal{F}, \{ p \}, \Delta_A \}$ and $B = \{ \{ q, q_1, q_2 \}, \mathcal{F}, \{ q \}, \Delta_B \}$

$$\begin{align*}
\Delta_A &= \{ p \xrightarrow{f} (p_1, p_2), \quad p_1 \xrightarrow{g} p_1, \quad p_1 \xrightarrow{a} () , \\
&\quad p_2 \xrightarrow{g} p_2, \quad p_2 \xrightarrow{b} () \} \\
\Delta_B &= \{ q \xrightarrow{f} (q_1, q_2), \quad q_1 \xrightarrow{g} q_1, \quad q_1 \xrightarrow{a} () , \\
&\quad q \xrightarrow{f} (q_2, q_1), \quad q_2 \xrightarrow{g} q_2, \quad q_2 \xrightarrow{b} () \}
\end{align*}$$

Starting from both $p$ and $q$  \hspace{3cm} Only starting from $q$
Example of inductive entailments in FOL

\[ p \overset{f}{\rightarrow} (p_1, p_2) \rightarrow \langle p(x), \{x \approx f(x_1, x_2), p_1(x_1), p_2(x_2)\} \rangle \]

\[ p(x) \leftarrow_S x \approx f(x_1, x_2), p_1(x_1), p_2(x_2) \quad q(x) \leftarrow_S x \approx f(x_1, x_2), q_1(x_1), q_2(x_2) \]

\[ p_1(x) \leftarrow_S x \approx g(x_1), p_1(x_1) \mid x \approx a \quad q_1(x) \leftarrow_S x \approx g(x_1), q_1(x_1) \mid x \approx a \]

\[ p_2(x) \leftarrow_S x \approx g(x_1), p_2(x_1) \mid x \approx b \quad q_2(x) \leftarrow_S x \approx g(x_1), q_2(x_1) \mid x \approx b \]

\[ p \models^H_S q \quad q \not\models^H_S p \]
Entailments with symbolic heap constraints

Symbolic heaps $\Pi \land \Theta$

\[
\begin{align*}
\Pi &::= x \approx y \mid \neg(x \approx y) \mid \Pi_1 \land \Pi_2 \\
\Theta &::= \text{emp} \mid x \mapsto (y_1, \ldots, y_k) \mid \Theta_1 \ast \Theta_2
\end{align*}
\]

Theorem

The entailment problem is undecidable for inductive systems with symbolic heap constraints.
Examples of inductive entailments in SL

\[ \text{ls}(x, y) \leftarrow_{S_l} \text{emp} \land x = y \mid x \mapsto u, \text{ls}(u, y) \]
\[ \text{ls}^a(x, y) \leftarrow_{S_l} \text{emp} \land x = y \mid x \neq y \land x \mapsto u, \text{ls}^a(u, y) \]

\[ \text{ls}^a \models_{S_l} \text{ls} \quad \text{ls} \not\models_{S_l} \text{ls}^a \]
Examples of inductive entailments in SL

\[ tree(x) \leftarrow_{S_t} \text{emp} \land x = \text{nil} \mid x \mapsto (l, r), \, tree(l), \, tree(r) \]

\[ tree_1^+(x) \leftarrow_{S_t} x \mapsto (\text{nil, nil}) \mid x \mapsto (l, r), \, tree_1^+(l), \, tree(r) \]
\[ \quad \mid x \mapsto (l, r), \, tree(l), \, tree_1^+(r) \]

\[ tree_2^+(x) \leftarrow_{S_t} x \mapsto (\text{nil, nil}) \mid x \mapsto (l, r), \, tree_2^+(l), \, tree_2^+(r) \]

\[ tree_1^+ \models_{S_t} \text{tree} \quad tree_2^+ \models_{S_t} \text{tree} \quad tree_2^+ \models_{S_t} tree_1^+ \]

\[ tree \not\models_{S_t} tree_1^+ \quad tree \not\models_{S_t} tree_2^+ \quad tree_1^+ \not\models_{S_t} tree_2^+ \]
Outline

1. Motivation and contributions

2. Inductive systems of predicates
   - The entailment problem

3. Proof systems for inductive entailments
   - Downwards inclusion check for tree automata
   - Inference rules
   - Restricting the set of constraints for soundness and completeness

4. Decision procedures for Separation Logic

5. Implementation and experimental evaluation

6. Conclusions and future work
Language inclusion for tree automata

\[ A = \{\{p, p_1, p_2\}, \emptyset, \{p\}, \Delta_A\} \text{ and } B = \{\{q, q_1, q_2\}, \emptyset, \{q\}, \Delta_B\} \]

\[ \Delta_A = \{p \xrightarrow{f} (p_1, p_2), \quad p_1 \xrightarrow{g} p_1, \quad p_1 \xrightarrow{a} (), \quad p_2 \xrightarrow{g} p_2, \quad p_2 \xrightarrow{b} ()\} \]

\[ \Delta_B = \{q \xrightarrow{f} (q_1, q_2), \quad q_1 \xrightarrow{g} q_1, \quad q_1 \xrightarrow{a} (), \quad q \xrightarrow{f} (q_2, q_1), \quad q_2 \xrightarrow{g} q_2, \quad q_2 \xrightarrow{b} ()\} \]

**Language of** \(p\) **in** \(A\): \(L(A, p)\) – set of all trees accepted starting in \(p\)

**Language of** \(A\): \(L(A) = \bigcup_{s \in I_A} L(A, s) = L(A, p)\)

**Language inclusion**: Does \(L(A) \subseteq L(B)\) hold?

\[ L(A) \subseteq L(B) \iff L(A, p) \subseteq L(B, q) \iff p \models^H q \]
Downwards inclusion check for tree automata

\[ \Delta_A = \{ p \xrightarrow{f} (p_1, p_2), \quad p_1 \xrightarrow{g} p_1, \quad p_1 \xrightarrow{a} (), \quad p_2 \xrightarrow{g} p_2, \quad p_2 \xrightarrow{b} () \} \]

\[ \Delta_B = \{ q \xrightarrow{f} (q_1, q_2), \quad q_1 \xrightarrow{g} q_1, \quad q_1 \xrightarrow{a} (), \quad q \xrightarrow{f} (q_2, q_1), \quad q_2 \xrightarrow{g} q_2, \quad q_2 \xrightarrow{b} () \} \]
Downwards inclusion check for tree automata

\[ \Delta_A = \{ p \xrightarrow{f} (p_1, p_2), \quad p_1 \xrightarrow{g} p_1, \quad p_1 \xrightarrow{a} (), \quad p_2 \xrightarrow{g} p_2, \quad p_2 \xrightarrow{b} () \}\]

\[ \Delta_B = \{ q \xrightarrow{f} (q_1, q_2), \quad q_1 \xrightarrow{g} q_1, \quad q_1 \xrightarrow{a} (), \quad q \xrightarrow{f} (q_2, q_1), \quad q_2 \xrightarrow{g} q_2, \quad q_2 \xrightarrow{b} () \}\]

Efficient Inclusion Checking on Explicit and Semi-symbolic Tree Automata
Downwards inclusion check for tree automata

\[
\Delta_A = \{ p \xrightarrow{f} (p_1, p_2), \quad p_1 \xrightarrow{g} p_1, \quad p_1 \xrightarrow{a} (),
\quad p_2 \xrightarrow{g} p_2, \quad p_2 \xrightarrow{b} () \}
\]

\[
\Delta_B = \{ q \xrightarrow{f} (q_1, q_2), \quad q_1 \xrightarrow{g} q_1, \quad q_1 \xrightarrow{a} (),
\quad q \xrightarrow{f} (q_2, q_1), \quad q_2 \xrightarrow{g} q_2, \quad q_2 \xrightarrow{b} () \}
\]

\[
L(A, p_1) \times L(A, p_2) \subseteq L(B, q_1) \times L(B, q_2) \cup L(B, q_2) \times L(B, q_1)
\]

Efficient Inclusion Checking on Explicit and Semi-symbolic Tree Automata
Downwards inclusion check for tree automata

\[ R_1 \times R_2 \subseteq S_1 \times S_2 \cup T_1 \times T_2 \]
Downwards inclusion check for tree automata

\[ R_1 \times R_2 \subseteq S_1 \times S_2 \cup T_1 \times T_2 \]

\[ \iff \]

\[ ( R_1 \subseteq S_1 \cup T_1 ) \land ( R_1 \subseteq S_1 \lor R_2 \subseteq T_2 ) \land ( R_1 \subseteq T_1 \lor R_2 \subseteq S_2 ) \land ( R_2 \subseteq S_2 \cup T_2 ) \]

Efficient Inclusion Checking on Explicit and Semi-symbolic Tree Automata
Downwards inclusion check for tree automata

\[ R_1 \times R_2 \subseteq S_1 \times S_2 \cup T_1 \times T_2 \]

\[ \iff \]

\[ (R_1 \subseteq S_1 \cup T_1) \land (R_1 \subseteq S_1 \lor R_2 \subseteq T_2) \land (R_1 \subseteq T_1 \lor R_2 \subseteq S_2) \land (R_2 \subseteq S_2 \cup T_2) \]

\[ \iff \]

\[ (R_1 \subseteq S_1 \cup T_1) \land (R_1 \subseteq S_1) \land (R_1 \subseteq T_1) \land (R_2 \subseteq S_2 \cup T_2) \]

---

*Efficient Inclusion Checking on Explicit and Semi-symbolic Tree Automata*

Downwards inclusion check for tree automata

\[ R_1 \times R_2 \subseteq S_1 \times S_2 \cup T_1 \times T_2 \]

\[ \iff \]

\[ (R_1 \subseteq S_1 \cup T_1) \land (R_1 \subseteq S_1 \lor R_2 \subseteq T_2) \land (R_1 \subseteq T_1 \lor R_2 \subseteq S_2) \land (R_2 \subseteq S_2 \cup T_2) \]

\[ \iff \]

\[ (R_1 \subseteq S_1 \cup T_1) \land (R_1 \subseteq S_1) \land (R_1 \subseteq T_1) \land (R_2 \subseteq S_2 \cup T_2) \lor \]

\[ (R_1 \subseteq S_1 \cup T_1) \land (R_1 \subseteq S_1) \land (R_2 \subseteq S_2) \land (R_2 \subseteq S_2 \cup T_2) \]

---

*Efficient Inclusion Checking on Explicit and Semi-symbolic Tree Automata*

Downwards inclusion check for tree automata

\[ R_1 \times R_2 \subseteq S_1 \times S_2 \cup T_1 \times T_2 \]

\[ \iff \]

\[ (R_1 \subseteq S_1 \cup T_1) \land (R_1 \subseteq S_1) \land (R_1 \subseteq T_1) \land (R_2 \subseteq S_2 \cup T_2) \lor (R_1 \subseteq S_1 \cup T_1) \land (R_1 \subseteq S_1) \land (R_2 \subseteq S_2) \land (R_2 \subseteq S_2 \cup T_2) \lor (R_1 \subseteq S_1 \cup T_1) \land (R_2 \subseteq T_2) \land (R_1 \subseteq T_1) \land (R_2 \subseteq S_2 \cup T_2) \]

Efficient Inclusion Checking on Explicit and Semi-symbolic Tree Automata
Downwards inclusion check for tree automata

\[ R_1 \times R_2 \subseteq S_1 \times S_2 \cup T_1 \times T_2 \]

\[ \iff \]

\[ (R_1 \subseteq S_1 \cup T_1) \land (R_1 \subseteq S_1) \lor (R_2 \subseteq T_2) \land (R_1 \subseteq T_1) \lor (R_2 \subseteq S_2) \land (R_2 \subseteq S_2 \cup T_2) \]

\[ \iff \]

\[ (R_1 \subseteq S_1 \cup T_1) \land (R_1 \subseteq S_1) \land (R_1 \subseteq T_1) \land (R_2 \subseteq S_2 \cup T_2) \lor \\
(R_1 \subseteq S_1 \cup T_1) \land (R_1 \subseteq S_1) \land (R_2 \subseteq S_2) \land (R_2 \subseteq S_2 \cup T_2) \lor \\
(R_1 \subseteq S_1 \cup T_1) \land (R_2 \subseteq T_2) \land (R_1 \subseteq T_1) \land (R_2 \subseteq S_2 \cup T_2) \lor \\
(R_1 \subseteq S_1 \cup T_1) \land (R_2 \subseteq T_2) \land (R_2 \subseteq S_2) \land (R_2 \subseteq S_2 \cup T_2) \]

Efficient Inclusion Checking on Explicit and Semi-symbolic Tree Automata
Downwards inclusion check for tree automata

\[ \Delta_A = \{ p \xrightarrow{f} (p_1, p_2), \ p_1 \xrightarrow{g} p_1, \ p_1 \xrightarrow{a} (), \ p_2 \xrightarrow{g} p_2, \ p_2 \xrightarrow{b} () \} \]

\[ \Delta_B = \{ q \xrightarrow{f} (q_1, q_2), \ q_1 \xrightarrow{g} q_1, \ q_1 \xrightarrow{a} (), \ q \xrightarrow{f} (q_2, q_1), \ q_2 \xrightarrow{g} q_2, \ q_2 \xrightarrow{b} () \} \]

Efficient Inclusion Checking on Explicit and Semi-symbolic Tree Automata
Downwards inclusion check for tree automata

\[ \Delta_A = \{ p \xrightarrow{f} (p_1, p_2), \; p_1 \xrightarrow{g} p_1, \; p_1 \xrightarrow{a} (), \; p_2 \xrightarrow{g} p_2, \; p_2 \xrightarrow{b} () \} \]

\[ \Delta_B = \{ q \xrightarrow{f} (q_1, q_2), \; q_1 \xrightarrow{g} q_1, \; q_1 \xrightarrow{a} (), \; q \xrightarrow{f} (q_2, q_1), \; q_2 \xrightarrow{g} q_2, \; q_2 \xrightarrow{b} () \} \]

Check successful when reaching \(((), S)\) with \((()) \in S\)

Efficient Inclusion Checking on Explicit and Semi-symbolic Tree Automata
Downwards inclusion check for tree automata

\[ \Delta_A = \{ \ p \overset{f}{\rightarrow} (p_1, p_2), \ p_1 \overset{g}{\rightarrow} p_1, \ p_1 \overset{a}{\rightarrow} (), \ p_2 \overset{g}{\rightarrow} p_2, \ p_2 \overset{b}{\rightarrow} () \} \]

\[ \Delta_B = \{ \ q \overset{f}{\rightarrow} (q_1, q_2), \ q_1 \overset{g}{\rightarrow} q_1, \ q_1 \overset{a}{\rightarrow} (), \ q \overset{f}{\rightarrow} (q_2, q_1), \ q_2 \overset{g}{\rightarrow} q_2, \ q_2 \overset{b}{\rightarrow} () \} \]

Check successful when reaching an already processed pair \((p, S)\)

Efficient Inclusion Checking on Explicit and Semi-symbolic Tree Automata

Downwards inclusion check for tree automata

\[ \Delta_A = \{ \ p \xmapsto{f} (p_1, p_2), \ p_1 \xmapsto{g} p_1, \ p_1 \xmapsto{a} (), \ p_2 \xmapsto{g} p_2, \ p_2 \xmapsto{b} () \} \]

\[ \Delta_B = \{ \ q \xmapsto{f} (q_1, q_2), \ q_1 \xmapsto{g} q_1, \ q_1 \xmapsto{a} (), \ q \xmapsto{f} (q_2, q_1), \ q_2 \xmapsto{g} q_2, \ q_2 \xmapsto{b} () \} \]

Efficient Inclusion Checking on Explicit and Semi-symbolic Tree Automata
Sequents and inference rules

A pair \((p, \{q_1, \ldots, q_n\})\) \rightarrow \text{ a sequent } \Gamma \vdash \Delta

\Gamma \text{ – set of formulae and predicate atoms (conjunction)}
\Delta \text{ – set of conjunctions over formulae and predicate atoms (disjunction)}
Sequents and inference rules

a pair \((p, \{q_1, \ldots, q_n\})\) \rightarrow \text{ a sequent } \Gamma \vdash \Delta

\Gamma – set of formulae and predicate atoms
(conjunction)

\Delta – set of conjunctions over formulae
and predicate atoms (disjunction)

An inference rule is of the form:

\[
\begin{align*}
&\Gamma_1 \vdash \Delta_1 \quad \ldots \quad \Gamma_n \vdash \Delta_n \\
&\text{IR} \\
&\frac{}{\Gamma \vdash \Delta}
\end{align*}
\]

- \(\Gamma \vdash \Delta\) is the \textbf{consequent}
- \(\Gamma_i \vdash \Delta_i\) are the \textbf{antecedents}
Sequents and inference rules

A pair \((p, \{q_1, \ldots, q_n\})\) leads to a sequent \(\Gamma \vdash \Delta\)

\(\Gamma\) – set of formulae and predicate atoms (conjunction)
\(\Delta\) – set of conjunctions over formulae and predicate atoms (disjunction)

An inference rule is of the form:

\[
\frac{\vdots}{\begin{array}{c}
\top \\
\Gamma \vdash \Delta
\end{array}}
\]

- \(\Gamma \vdash \Delta\) is the **consequent**
- \(\Gamma_i \vdash \Delta_i\) are the **antecedents**, we write \(\top\) if \(n = 0\)
Sequents and inference rules

A pair \((p, \{q_1, \ldots, q_n\})\) → a sequent \(\Gamma \vdash \Delta\)

\(\Gamma\) – set of formulae and predicate atoms (conjunction)
\(\Delta\) – set of conjunctions over formulae and predicate atoms (disjunction)

An inference rule is of the form:

\[
\begin{array}{c}
\top \\
\Gamma \vdash \Delta \\
\vdots \\
\Gamma_p \vdash \Delta_p
\end{array}
\]

- \(\Gamma \vdash \Delta\) is the **consequent**
- \(\Gamma_i \vdash \Delta_i\) are the **antecedents**, we write \(\top\) if \(n = 0\)
- \(\Gamma_p \vdash \Delta_p\) is the **pivot** of the rule, ancestor of the consequent
Sequents and inference rules

A pair \((p, \{q_1, \ldots, q_n\})\) → a sequent \(\Gamma \vdash \Delta\)

\(\Gamma\) – set of formulae and predicate atoms (conjunction)
\(\Delta\) – set of conjunctions over formulae and predicate atoms (disjunction)

An inference rule is of the form:

\[
\frac{
\top
}{
\Gamma \vdash \Delta \quad \text{side conditions}
}
\]

- \(\Gamma \vdash \Delta\) is the consequent
- \(\Gamma_i \vdash \Delta_i\) are the antecedents, we write \(\top\) if \(n = 0\)
- \(\Gamma_p \vdash \Delta_p\) is the pivot of the rule, ancestor of the consequent
- \(C\) is a pivot constraint on the path from the pivot to the consequent
Derivations and proofs

\[
\begin{align*}
\text{IR}_1 & \quad \vdash \quad \text{IR}_2 \\
\Gamma_2 & \vdash \Delta_2 \\
\text{IR}_3 & \quad \vdash \\
\Gamma_4 & \vdash \Delta_4 \\
\text{IR}_4 & \quad \vdash \\
\Gamma_5 & \vdash \Delta_5 \\
\end{align*}
\]

\textbf{Derivation} (possibly infinite)

\[
\begin{align*}
\text{IR}_1 & \quad \vdash \quad \text{IR}_2 \\
\Gamma_2 & \vdash \Delta_2 \\
\text{IR}_3 & \quad \vdash \\
\Gamma_4 & \vdash \Delta_4 \\
\text{IR}_4 & \quad \vdash \\
\Gamma_5 & \vdash \Delta_5 \\
\end{align*}
\]

\textbf{Proof} (finite, all leaves are } \top \\
Derivations and proofs

\[ \Gamma \vdash \Delta \]

\[ \Lambda \Gamma \vdash^T \exists \Delta \quad \text{(FOL)} \]

\[ \star \Gamma \vdash^{SL} \exists \Delta \quad \text{(SL)} \]

- **Soundness** - If there exists a proof for \( \Gamma \vdash \Delta \), then the corresponding entailment holds.

- **Completeness** - If the entailment holds, then there exists a proof for its corresponding sequent.
Inference rules - LU (left unfold)

\[ \begin{align*}
\text{LU} & \quad \frac{x \mapsto (\text{nil}, \text{nil}) \vdash \text{tree}_1^+(x)}{\vdash \text{tree}_2^+(x)} \\
& \quad \exists l_1, \exists r_1. x \mapsto (l_1, r_1) \Rightarrow \text{tree}_2^+(x) \vdash \text{tree}_1^+(x)
\end{align*} \]

\[ \text{tree}_2^+(x) \leftarrow S_t \ x \mapsto (\text{nil}, \text{nil}) \mid x \mapsto (l, r), \text{tree}_2^+(l), \text{tree}_2^+(r) \]
Inference rules - RU (right unfold)

\[ x \mapsto (\text{nil}, \text{nil}) \vdash x \mapsto (\text{nil}, \text{nil}) , \]
\[ \exists l_1 \exists r_1 . x \mapsto (l_1, r_1) \ast \text{tree}_1^+(l_1) \ast \text{tree}(r_1) , \]
\[ \exists l_1 \exists r_1 . x \mapsto (l_1, r_1) \ast \text{tree}(l_1) \ast \text{tree}_1^+(r_1) \]

\[ \text{RU} \quad \text{LU} \]
\[ x \mapsto (\text{nil}, \text{nil}) \vdash \text{tree}_1^+(x) \]
\[ \text{tree}_2^+(x) \vdash \text{tree}_1^+(x) \]

\[ \text{tree}_1^+(x) \leftarrow_{S_t} x \mapsto (\text{nil}, \text{nil}) \mid x \mapsto (l, r), \text{tree}_1^+(l), \text{tree}(r) \]
\[ \mid x \mapsto (l, r), \text{tree}(l), \text{tree}_1^+(r) \]
Inference rules - AX (axiom)

\[ \begin{align*}
AX & \quad \vdash (\text{nil}, \text{nil}), \\
RU & \quad \vdash (\text{nil}, \text{nil}), \quad \exists l_1 \exists r_1 . x \mapsto (l_1, r_1) * \text{tree}^+_1(l_1) * \text{tree}(r_1), \\
LU & \quad \vdash \text{tree}^+_1(x), \quad \exists l_1 \exists r_1 . x \mapsto (l_1, r_1) * \text{tree}(l_1) * \text{tree}^+_1(r_1)
\end{align*} \]

Similar to reaching a pair \(((), S)\) with \((()) \in S\) in the NFTA inclusion check
Inference rules - LU (left unfold)

\[
\begin{align*}
\text{LU} & \quad x \mapsto (l_0, r_0), \ tree_2^+(l_0), \ tree_2^+(r_0) \vdash \ tree_1^+(x) \\
& \quad \text{tree}_2^+(x) \vdash \ tree_1^+(x)
\end{align*}
\]

\[
\text{tree}_2^+(x) \leftarrow S_t \ x \mapsto (\text{nil}, \text{nil}) \mid x \mapsto (l, r), \ tree_2^+(l), \ tree_2^+(r)
\]
Inference rules - RU (right unfold)

\[ x \mapsto (l_0, r_0), \; \text{tree}_2^+(l_0), \; \text{tree}_2^+(r_0) \vdash x \mapsto (\text{nil}, \text{nil}), \]
\[ \exists l_1 \exists r_1 . x \mapsto (l_1, r_1) \ast \text{tree}_1^+(l_1) \ast \text{tree}(r_1), \]
\[ \exists l_1 \exists r_1 . x \mapsto (l_1, r_1) \ast \text{tree}(l_1) \ast \text{tree}_1^+(r_1) \]

\[ \frac{\text{RU}}{\text{LU}} \]

\[ x \mapsto (l_0, r_0), \; \text{tree}_2^+(l_0), \; \text{tree}_2^+(r_0) \vdash \text{tree}_1^+(x) \]
\[ \text{tree}_2^+(x) \vdash \text{tree}_1^+(x) \]

\[ \text{tree}_1^+(x) \leftarrow_{S_t} x \mapsto (\text{nil}, \text{nil}) \mid x \mapsto (l, r), \; \text{tree}_1^+(l), \; \text{tree}(r) \]
\[ \mid x \mapsto (l, r), \; \text{tree}(l), \; \text{tree}_1^+(r) \]
Inference rules - RD (reduce)

\[
\begin{align*}
\text{RD} & \quad tree_2^+(l_0), \ tree_2^+(r_0) \vdash tree_1^+(l_0) \ast tree(r_0), \ tree(l_0) \ast tree_1^+(r_0) \\
& \quad x \mapsto (l_0, r_0), \ tree_2^+(l_0), \ tree_2^+(r_0) \vdash x \mapsto (\text{nil, nil}), \\
& \quad \exists l_1 \exists r_1. \ x \mapsto (l_1, r_1) \ast tree_1^+(l_1) \ast tree(r_1), \\
& \quad \exists l_1 \exists r_1. \ x \mapsto (l_1, r_1) \ast tree(l_1) \ast tree_1^+(r_1)
\end{align*}
\]

\[
\begin{align*}
\text{RU} & \quad x \mapsto (l_0, r_0), \ tree_2^+(l_0), \ tree_2^+(r_0) \vdash tree_1^+(x) \\
& \quad LU \quad x \mapsto (l_0, r_0), \ tree_2^+(l_0), \ tree_2^+(r_0) \vdash tree_1^+(x) \\
& \quad tree_2^+(x) \vdash tree_1^+(x)
\end{align*}
\]

\[
\begin{align*}
\quad x \mapsto (l_0, r_0) \not\models_{\text{SL}} x \mapsto (\text{nil, nil}) \\
\quad x \mapsto (l_0, r_0) \models_{\text{SL}} \exists l_1 \exists r_1. \ x \mapsto (l_1, r_1) \quad \Rightarrow \quad \theta = \{(l_1, l_0), (r_1, r_0)\}
\end{align*}
\]

LU + RU + RD encode a transition action in the NFTA inclusion check
\[ \begin{align*}
\text{tree}_2^+(l_0) & \vdash \text{tree}_1^+(l_0) \\
\text{tree}_2^+(l_0), \text{tree}_2^+(r_0) & \vdash \text{tree}_1^+(l_0) \ast \text{tree}(r_0), \text{tree}(l_0) \ast \text{tree}_1^+(r_0) \\
\exists l_1 \exists r_1 . x & \mapsto (l_1, r_1) \ast \text{tree}_1^+(l_1) \ast \text{tree}(r_1), \\
\exists l_1 \exists r_1 . x & \mapsto (l_1, r_1) \ast \text{tree}(l_1) \ast \text{tree}_1^+(r_1) \\
\text{tree}_2^+(x) & \vdash \text{tree}_1^+(x)
\end{align*} \]
Inference rules - ID (infinite descent)

\[
\begin{align*}
\text{ID} & \quad \top \\
\text{SP} & \quad \result \ (l_0) \vdash \result (l_0) \\
\text{RD} & \quad \result (l_0), \ \gamma + \gamma (r_0) \vdash \gamma \ast \gamma (r_0), \ \gamma (l_0) \ast \gamma (r_0) \\
\text{RU} & \quad \gamma \ast \gamma (l_0), \ \gamma + \gamma (r_0) \vdash \gamma (x) \\
\end{align*}
\]

\[
\begin{align*}
\gamma & \mapsto (l_0, r_0), \ \gamma + \gamma (l_0), \ \gamma + \gamma (r_0) \vdash \gamma \mapsto (\text{nil}, \text{nil}), \\
& \quad \exists l_1 \exists r_1. \ \gamma \mapsto (l_1, r_1) \ast \gamma (l_1) \ast \gamma (r_1), \\
& \quad \exists l_1 \exists r_1. \ \gamma \mapsto (l_1, r_1) \ast \gamma (l_1) \ast \gamma (r_1) \\
\end{align*}
\]

\[
\begin{align*}
\gamma & \mapsto (l_0, r_0), \ \gamma + \gamma (l_0), \ \gamma + \gamma (r_0) \vdash \gamma (x) \\
\end{align*}
\]

**Backlink** from $\Gamma \vdash \Delta' \theta$ to the pivot $\Gamma \vdash \Delta$ with $\Delta' \supseteq \Delta$

**Pivot constraint:** LU required between the pivot and the consequent

Generalizes the reaching an already seen pair in the NFTA inclusion check
Outline

1. Motivation and contributions

2. Inductive systems of predicates
   - The entailment problem

3. Proof systems for inductive entailments
   - Downwards inclusion check for tree automata
   - Inference rules
   - Restricting the set of constraints for soundness and completeness

4. Decision procedures for Separation Logic

5. Implementation and experimental evaluation

6. Conclusions and future work
Ranked

Subgoal models decrease in a well-founded domain w.r.t the goal models

If $h_1 = h \cup h_2$, then $h_2 \trianglelefteq h_1$ and $h_2 \vartriangleleft h_1$ if, moreover, $h \neq \emptyset$

$$\langle p(\overline{x}), \{ \phi(\overline{x}, \overline{x_1}, \ldots, \overline{x_m}), q_1(\overline{x_1}), \ldots, q_i(\overline{x_1}), \ldots, q_m(\overline{x_m}) \} \rangle$$

Necessary to ensure progress along a recurring branch closed by ID

Sufficient to check $\phi \models_{\text{SL}} \neg \text{emp}$ for each predicate rule with subgoals

- In PSPACE for quantifier-free SL constraints
- In P for symbolic heap constraints
Soundness

Inference rules are sound for entailments with **ranked** inductive definitions. \( \text{LU, RU, RD, SP, AX} \) are **locally sound**.

**Global soundness of proofs using ID**: a counterexample \( \text{cex}_1 \) for the root \( \Gamma_1 \vdash \Delta_1 \) of a proof can be propagated along an infinite trace:

\[
\Gamma_1 \vdash \Delta_1 \quad \Gamma_2 \vdash \Delta_2 \quad \ldots \quad \Gamma_i \vdash \Delta_i \quad \Gamma_{i+1} \vdash \Delta_{i+1} \quad \ldots \\
\text{cex}_1 \quad \triangleright \quad \text{cex}_2 \quad \triangleright \quad \ldots \quad \triangleright \quad \text{cex}_i \quad \triangleright \quad \text{cex}_{i+1} \quad \triangleright \quad \ldots 
\]

Infinitely many direct paths \( \Gamma_p \vdash \Delta_p \ldots \Gamma \vdash \Delta \) s.t. \( \Gamma_p \vdash \Delta_p \) pivot of \( \Gamma \vdash \Delta \)

Ranked + \( \text{LU} \) required \( \Rightarrow \) counterexample strictly decreases infinitely often, contradicting the well-foundedness of the chosen domain.
Non-filtering

The constraint of a predicate rule does not filter the subgoal solutions

\[ \langle p(\overline{x}), \{\phi(\overline{x}, \overline{x}_1, \ldots, \overline{x}_m), q_1(\overline{x}_1), \ldots, q_m(\overline{x}_m)\} \rangle \]

\[ h_0 \text{ vs } h_1, \ldots, h_m \text{ disjoint} \]

\{x \mapsto (l, r), \text{tree}(l), \text{tree}(r)\} \text{ vs } \{x \mapsto (l, r) \land \neg(l \approx \text{nil}), \text{tree}(l), \text{tree}(r)\}
Non-filtering

The constraint of a predicate rule does not filter the subgoal solutions

\[ \langle p(\overline{x}), \{ \phi(\overline{x}, \overline{x}_1, \ldots, \overline{x}_m), q_1(\overline{x}_1), \ldots, q_m(\overline{x}_m) \} \rangle \]

\(h_0, h_1, \ldots, h_m\) disjoint

\{\(x \mapsto (l, r), \text{tree}(l), \text{tree}(r)\}\) vs \{\(x \mapsto (l, r) \land \neg(l \approx \text{nil}), \text{tree}(l), \text{tree}(r)\)\}

- In EXPSPACE for quantifier-free and \(-^*\)-free SL constraints
Finite variable instantiation (fvi)

\[ \phi( \bar{x}, \bar{x}_1, \ldots, \bar{x}_i, \ldots, \bar{x}_n ) \models \psi( \bar{x}, \bar{y}_1, \ldots, \bar{y}_j, \ldots, \bar{y}_m ) \]

Finitely many substitutions \( \theta \) mapping each \( \bar{y}_j \) to some \( \bar{x}_i \) s.t. \( \phi \models \psi \theta \)

Moreover, these are all the substitutions for which \( \phi \models \psi \theta \) holds

\[ x \mapsto (l_0, r_0) \models_{\text{SL}} \exists l_1 \exists r_1 . x \mapsto (l_1, r_1) \quad \theta = \{(l_1, l_0), (r_1, r_0)\} \]

- In NEXPTIME under the Herbrand interpretation
- In PSPACE for quantifier-free and \( \neg \)-free SL constraints
- In \( \Sigma^p_2 \) for symbolic heap constraints
Non-overlapping

Given two constraints $\phi(x, x_1, \ldots, x_n)$ and $\psi(x, y_1, \ldots, y_m)$

$\phi \land \psi$ is satisfiable only if $\phi \models \exists y_1 \ldots \exists y_m \cdot \psi$

Two constraints cannot overlap (i.e. have at least one model in common), without one entailing the other: $x \mapsto (\text{nil}, \text{nil}) \models^{\text{SL}} \exists \exists r . x \mapsto (l, r)$

Check that $\exists x_1 \exists x_1 \ldots \exists x_n \forall y_1 \ldots \forall y_m \cdot \phi \land \neg \psi$ is unsatisfiable

- In NP under the Herbrand interpretation
- In PSPACE for quantifier-free and $\neg \ast$-free SL constraints
- In $\Pi_2^P$ for symbolic heap constraints
Completeness in FOL

Inference rules are complete for entailments with FOL inductive systems that are **ranked, non-filtering, non-overlapping**, with the **fvi property**

Proofs can be discovered by employing a search strategy

$$ S = (LU \cdot RU^* \cdot RD \cdot \land R^* \cdot SP?)^* \cdot LU? \cdot RU^* \cdot (AX \mid ID) $$

- $\land R$ is a cleanup rule after RD, eliminates conjunctions of predicates with the same arguments from the right-hand side of a sequent
Completeness in SL

Only for definitions generating matching coverage trees of the heap models

\[ ls(x, y) \leftarrow S_r \text{ emp} \land x = y \mid x \mapsto u, ls(u, y) \]

\[ ls^r(x, y) \leftarrow S_r \text{ emp} \land x = y \mid u \mapsto y, ls^r(x, u) \]

\[ ls \models_{S_r} ls^r \text{ and } ls^r \models_{S_r} ls \text{ hold, but we cannot build a proof} \]

\[ \bar{v} = \langle 1, 4 \rangle \text{ and } h = \{(1, 2), (2, 3), (3, 4)\} \text{ solution for both } ls \text{ and } ls^r, \]

but the two definitions generate different coverage trees for \( h \)

\[ t = \{(1, 2)\} \quad t^r = \{(3, 4)\} \]

\[ \{(2, 3)\} \quad \{(2, 3)\} \]

\[ \{(3, 4)\} \quad \{(1, 2)\} \]
Outline

1 Motivation and contributions

2 Inductive systems of predicates
   - The entailment problem

3 Proof systems for inductive entailments
   - Downwards inclusion check for tree automata
   - Inference rules
   - Restricting the set of constraints for soundness and completeness

4 Decision procedures for Separation Logic

5 Implementation and experimental evaluation

6 Conclusions and future work
The quantifier-free fragment of SL

- The satisfiability problem for the quantifier-free fragment of SL is PSPACE-complete if the satisfiability of the base theory is in PSPACE.
- SL formulae are transformed into second-order formulae with quantifiers ranging over sets and uninterpreted functions, whose cardinality is polynomially bound by the size of the input.
- Lazy, counterexample-driven quantifier instantiation method.
- Sound, complete and terminating.

A Decision Procedure for Separation Logic in SMT
[A. Reynolds, R. Iosif, C. Serban, T. King – ATVA 2016]
The Bernays-Schönfinkel-Ramsey fragment of SL

- Satisfiability of $\exists x_1 \ldots \exists x_m \forall y_1 \ldots \forall y_n \cdot \phi(x_1, \ldots, x_m, y_1, \ldots, y_n)$

- $\text{Loc} = U$, $\text{Data} = U^k$, $U$ sort of the theory of equality
  - Undecidable when $k \geq 2$ due to $\forall$-quantified variables in the scope of $\neg \ast$ under an even number of negations
  - PSPACE-complete for the $\neg \ast$-free fragment

- $\text{Loc} = \text{Data} = \text{Int}$, with addition and total order - undecidable

- $\text{Loc} = U$, $\text{Data} = U \times \text{Int}$ - undecidable

- Semi-decision procedure - incremental quantifier instantiation based on candidate models returned by a solver for quantifier-free inputs

*Reasoning in the Bernays-Schönfinkel-Ramsey Fragment of Separation Logic*
[A. Reynolds, R. Iosif, C. Serban – VMCAI 2017]*
Outline

1 Motivation and contributions

2 Inductive systems of predicates
   • The entailment problem

3 Proof systems for inductive entailments
   • Downwards inclusion check for tree automata
   • Inference rules
   • Restricting the set of constraints for soundness and completeness

4 Decision procedures for Separation Logic

5 Implementation and experimental evaluation

6 Conclusions and future work
Inductor – checker for inductive entailments in SL

Written in C++, queries CVC4 for noninductive entailments

Inductive definitions and entailment queries are given as SMT-LIB scripts

Proof-search strategies given as nondeterministic finite word automata

Explores all possible derivations using a tree structure with two types of nodes: SNodes (for sequents) and RNodes (for inference rules)

Node status: unknown, valid, invalid, updates propagate to ancestors

An Entailment Checker for Separation Logic with Inductive Definitions
[R. Iosif, C. Serban – AVOCS 2018]
## Experimental evaluation

<table>
<thead>
<tr>
<th>LHS</th>
<th>RHS</th>
<th>R</th>
<th>Sequents</th>
<th>Rules</th>
<th>$H_{Seq}$</th>
<th>$H_{LU}$</th>
<th>$H_{SP}$</th>
<th>T</th>
<th>CVC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$tree_1^+$</td>
<td>$tree$</td>
<td>√</td>
<td>34</td>
<td>22</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>0.096s</td>
<td>9</td>
</tr>
<tr>
<td>$tree_2^+$</td>
<td>$tree$</td>
<td>√</td>
<td>21</td>
<td>15</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>0.053s</td>
<td>7</td>
</tr>
<tr>
<td>$tree_2^+$</td>
<td>$tree_1^+$</td>
<td>√</td>
<td>1477</td>
<td>889</td>
<td>11</td>
<td>3</td>
<td>2</td>
<td>5.515s</td>
<td>37</td>
</tr>
<tr>
<td>$tree_1$</td>
<td>$tree_1^+$</td>
<td></td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0.033s</td>
<td>7</td>
</tr>
<tr>
<td>$tree_1$</td>
<td>$tree_2^+$</td>
<td></td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0.028s</td>
<td>5</td>
</tr>
<tr>
<td>$tree_1^+$</td>
<td>$tree_2^+$</td>
<td></td>
<td>38</td>
<td>25</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>0.096s</td>
<td>14</td>
</tr>
<tr>
<td>$ls^a$</td>
<td>$ls$</td>
<td>√</td>
<td>8</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0.014s</td>
<td>2</td>
</tr>
<tr>
<td>$ls$</td>
<td>$ls^a$</td>
<td></td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0.015s</td>
<td>2</td>
</tr>
</tbody>
</table>

---

An Entailment Checker for Separation Logic with Inductive Definitions

[R. Iosif, C. Serban – AVOCS 2018]
Outline

1. Motivation and contributions
2. Inductive systems of predicates
   - The entailment problem
3. Proof systems for inductive entailments
   - Downwards inclusion check for tree automata
   - Inference rules
   - Restricting the set of constraints for soundness and completeness
4. Decision procedures for Separation Logic
5. Implementation and experimental evaluation
6. Conclusions and future work
Conclusions

Cyclic proof systems for entailments between inductively defined predicates in (multisorted) First-Order Logic and Separation Logic

Infinite Descent principle, inductive invariants produced during proof search

Semantic boundaries within which soundness and completeness are ensured

Dedicated decision procedures for SL (used for checking the restrictions and noninductive entailments when applying AX or RD)

Proof of concept entailment checker implementation for SL
Prove entailments between inductive definitions that generate coverage trees related by an isomorphism resulting from a rotation ($ls \models^{SL}_{S_r} ls^r$)