An Entailment Checker for Separation Logic with Inductive Definitions

Radu Iosif, Cristina Serban

Université de Grenoble Alpes, CNRS, VERIMAG

July 18, 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 Implementation and experimental evaluation

5 Conclusions
Program verification

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 \( B \)?

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 $\longrightarrow$ $c$ is executed $\longrightarrow$ 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

\[ \text{sp}(P, \phi) \models \psi \quad \phi \models \text{wp}(P, \psi) \]
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: $\ast$, $\multimap$ (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
- $\multimap$ talks about new or fresh resources
The ability to extend local specifications regained at a deeper level with *

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

\(\phi\) and \(\psi\) refer to the footprint of \(c\), while \(\varphi\) involves only variables and heap cells not modified or mutated by \(c\)
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

Can be used in verification conditions
Building a list segment $ls(x, y)$

$empty$

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

$\text{empty}$

$\text{emp} \land x = y$

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

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

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

Cristina Serban (VERIMAG)
An Entailment Checker for SL with Inductive Definitions
July 18, 2018 9/44
Building a list segment \( ls(x, y) \)

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

$\text{empty}$

$emp \land x = y$  \hspace{1cm} $x \mapsto u \ast ls(u, y)$

$ls(x, y) ::= emp \land x = y \lor \exists u . x \mapsto u \ast ls(u, y)$

list segment
Building a list segment $ls(x, y)$

\[ \text{empty} \]

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

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

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

list segment
Building a list segment $ls(x, y)$

$ls(x, y) ::= emp \land x = y \lor \exists u . x \mapsto u \ast ls(u, y)$

$ls^a(x, y) ::= emp \land x = y \lor x \neq y \land \exists u . x \mapsto u \ast ls^a(u, y)$

List segment

Acyclic list segment
Contributions

Checker for entailments between predicates with inductive definitions in SL

Implements a cyclic proof system inspired by an antichain-based method for language inclusion of tree automata

Identify semantic restrictions for soundness and completeness

Decision procedures from the DPLL(T)-based SMT solver CVC4 are used to establish the satisfiability of ground SL formulae
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 Implementation and experimental evaluation

5 Conclusions
Syntax of SL

**Sort symbols:** Loc, Data, Bool, where Data = Loc\(^k\) with \(k \geq 1\) fixed

**Function symbols:** \(\text{nil}^{\text{Loc}}\)

**Variables:** \(\text{Var} = \{x, y, z, \ldots\}\), each having a sort \(\sigma\)

Fixed **interpretation** \(\mathcal{I}\) s.t. \(\text{Loc}^{\mathcal{I}} = \mathcal{L}\) countably infinite, \(\text{nil}^{\mathcal{I}} = \ell_{\text{nil}} \in \mathcal{L}\)
Syntax of SL

Sort symbols: Loc, Data, Bool, where Data = Loc$^k$ with $k \geq 1$ fixed

Function symbols: nil$^{\text{Loc}}$

Variables: Var = \{x, y, z, \ldots\}, each having a sort $\sigma$

Fixed interpretation $\mathcal{I}$ s.t. Loc$^\mathcal{I}$ = L countably infinite, nil$^\mathcal{I}$ = $\ell_{\text{nil}}$ ∈ L

Terms: $t^\sigma ::= x^\sigma \in \text{Var} \mid \text{nil}^{\text{Loc}} \mid \langle t^{\text{Loc}}_1, \ldots, t^{\text{Loc}}_k \rangle$

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

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

Heap is a mapping \( h : L \rightarrow_{\text{fin}} \mathcal{L}^k \) and Heaps is the set of heaps

\[
h_1 \neq h_2 \iff \text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset \quad h = h_1 \cup h_2 \iff h_1 \neq h_2 \land h = h_1 \cup h_2
\]

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

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

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

A **predicate rule** is a pair

$$\langle p(\bar{x}), \{\phi(\bar{x}, \bar{x}_1, \ldots, \bar{x}_m), q_1(\bar{x}_1), \ldots, q_m(\bar{x}_m)\}\rangle$$
Systems of inductive definitions

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

A predicate rule is a pair

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

constraint subgoals

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

$$p(\overline{x}) :=_S R_1 \mid \ldots \mid R_\ell$$

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

\( F_S \) monotone and continuous function on assignments mapping each
\( p^{\sigma_1...\sigma_n} \in \text{Pred} \) to a set of solutions \( \mathcal{X}(p) \subseteq \text{L}^n \times \text{Heaps} \)

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

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

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

**Given an inductive system $S$**

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

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

We denote entailment problems as $p \models^S_{SL} q_1, \ldots, q_m$
Examples of inductive entailments in SL

\[ \begin{align*}
ls(x, y) &:= \mathcal{S}_\ell \text{ emp} \land x = y \mid x \mapsto u, \, ls(u, y) \\
ls^a(x, y) &:= \mathcal{S}_\ell \text{ emp} \land x = y \mid x \neq y \land x \mapsto u, \, ls^a(u, y)
\end{align*} \]

\[ \begin{align*}
\models_{\mathcal{S}_\ell} \quad &ls^a \quad \models_{\mathcal{S}_\ell} \quad ls \\
\not\models_{\mathcal{S}_\ell} \quad &ls \quad \not\models_{\mathcal{S}_\ell} \quad ls^a
\end{align*} \]
Examples of inductive entailments in SL

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

\[ tree_1^+(x) := S_t x \mapsto (\text{nil, nil}) | x \mapsto (l, r), tree_1^+(l), tree(r) \]

\[ tree_2^+(x) := S_t x \mapsto (\text{nil, nil}) | x \mapsto (l, r), tree_2^+(l), tree_2^+(r) \]

\[ \text{built only by } tree_1^+ \]

\[ \text{built both by } tree_1^+ \text{ and } tree_2^+ \]

\[ tree_1^+ \models_{S_t} tree \]

\[ tree_2^+ \models_{S_t} tree \]

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

\[ tree \not\models_{S_t} tree_2^+ \]
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 Implementation and experimental evaluation

5 Conclusions
Nondeterministic finite tree automata

An NFTA can be 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 $B = \{\{q, q_1, q_2\}, \mathcal{F}, \{q\}, \Delta_B\}$
An NFTA can be 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 $B = \{ \{ q, q_1, q_2 \}, \mathcal{F}, \{ q \}, \Delta_B \}$

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

An NFTA can be 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 $B = \{ \{ q, q_1, q_2 \}, \mathcal{F}, \{ q \}, \Delta_B \}$

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

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

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

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

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

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

An NFTA can be 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 $B = \{\{q, q_1, q_2\}, \mathcal{F}, \{q\}, \Delta_B\}$

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

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

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

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

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

Language inclusion: whether \( \mathcal{L}(A) \subseteq \mathcal{L}(B) \), where \( \mathcal{L}(A) = \bigcup_{s \in l_A} \mathcal{L}(A, s) \)

\[ \mathcal{L}(A) \subseteq \mathcal{L}(B) \iff \mathcal{L}(A, p) \subseteq \mathcal{L}(B, q) \iff p \models_S q \]
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), \ 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} (\) \}
\]

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

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

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

\[ ((p_1, p_2), \{(q_1, q_2), (q_2, q_1)\}) \]

\[ \mathcal{L}(A, p_1) \times \mathcal{L}(A, p_2) \subseteq \mathcal{L}(B, q_1) \times \mathcal{L}(B, q_2) \cup \mathcal{L}(B, q_2) \times \mathcal{L}(B, q_1) \]
Downwards inclusion check for tree automata

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\Gamma \text{ – set of formulae and predicate atoms} \quad \Delta \text{ – set of } \ast \text{ conjunctions over formulae and predicate atoms (disjunction)}

\Gamma \vdash \Delta \text{ is the consequent} \quad \Gamma_i \vdash \Delta_i \text{ are the antecedents} \quad \Gamma \vdash \Delta
Sequents and inference rules

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

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

An inference rule is of the form:

\[
IR \quad \frac{\Gamma_1 \vdash \Delta_1 \ldots \Gamma_n \vdash \Delta_n}{\Gamma \vdash \Delta} \quad \text{side conditions}
\]

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

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

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

An inference rule is of the form:

\[
\begin{align*}
IR & \quad \vdash \\
\Gamma \vdash \Delta & \quad \text{side conditions}
\end{align*}
\]

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

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

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

An inference rule is of the form:

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

- \(\Gamma \vdash \Delta\) 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 \((p, \{q_1, \ldots, q_m\})\) \rightarrow 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:

\[
\begin{array}{c}
IR \\
t &
\end{array}
\]

\(\Gamma \vdash \Delta\)
\(\vdots c\)
\(\Gamma_p \vdash \Delta_p\)

- \(\Gamma \vdash \Delta\) 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

\[ \Gamma_1 \vdash \Delta_1 \]

\[ \Gamma_2 \vdash \Delta_2 \]

\[ \Gamma_3 \vdash \Delta_3 \]

\[ \Gamma_4 \vdash \Delta_4 \]

\[ \Gamma_5 \vdash \Delta_5 \]

**Derivation** (possibly infinite)

\[ \Gamma_1 \vdash \Delta_1 \]

\[ \Gamma_2 \vdash \Delta_2 \]

\[ \Gamma_3 \vdash \Delta_3 \]

\[ \Gamma_4 \vdash \Delta_4 \]

\[ \Gamma_5 \vdash \Delta_5 \]

**Proof** (finite, all leaves are \( \top \))
Derivations and proofs

\[ \Gamma \vdash \Delta \rightarrow \ast \Gamma \models_{SL} \bigvee \Delta \]

- **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)

$$
\begin{array}{c}
\text{LU} \\
\hline
x \mapsto (\text{nil}, \text{nil}) \vdash tree_1^+(x) \\
\hline
\end{array}
$$

$$
tree_2^+(x) := S_t \ x \mapsto (\text{nil}, \text{nil}) \ | \ x \mapsto (l, r), \ tree_2^+(l), \ tree_2^+(r)
$$
Inference rules - RU (right unfold)

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

RU

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

LU

\[ tree^+_2(x) \vdash tree^+_1(x) \]

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

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

\[
\begin{align*}
\text{RU} & : \quad x \mapsto (\text{nil}, \text{nil}) \vdash \text{tree}_1^+ (x), \\
\text{LU} & : \quad \text{tree}_2^+ (x) \vdash \text{tree}_1^+ (x)
\end{align*}
\]

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

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

\[
tree_2^+ (x) := S_t \ x \mapsto (\text{nil}, \text{nil}) \ | \ x \mapsto (l, r), \ tree_2^+ (l), \ tree_2^+ (r)
\]
Inference rules - RU (right unfold)

\[
x \mapsto (l_0, r_0), \ tree_2^+(l_0), \ tree_2^+(r_0) \vdash x \mapsto (\text{nil}, \text{nil}),
\]

\[
\exists l_1 \exists r_1 . x \mapsto (l_1, r_1) * tree_1^+(l_1) * tree(r_1),
\]

\[
\exists l_1 \exists r_1 . x \mapsto (l_1, r_1) * tree(l_1) * tree_1^+(r_1)
\]

RU

\[
x \mapsto (l_0, r_0), \ tree_2^+(l_0), \ tree_2^+(r_0) \vdash \text{tree}_1^+(x)
\]

LU

\[
\begin{align*}
tree_2^+(x) \vdash & \quad \text{tree}_1^+(x) \\
\end{align*}
\]

\[
tree_1^+(x) := S_t \ x \mapsto (\text{nil}, \text{nil}) \mid x \mapsto (l, r), \ tree_1^+(l), \ tree(r) \\
\mid x \mapsto (l, r), \ tree(l), \ tree_1^+(r)
\]
Inference rules - RD (reduce)

\[
\begin{align*}
tree_2^+(l_0), \ tree_2^+(r_0) \vdash & \ tree_1^+(l_0) \ast tree(r_0), \ tree(l_0) \ast tree_1^+(r_0) \\
\end{align*}
\]

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

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

\[
\begin{align*}
x \mapsto (l_0, r_0) \not\models_{\text{SL}} x \mapsto (\text{nil, nil}) \\
\end{align*}
\]

\[
\begin{align*}
x \mapsto (l_0, r_0) \models_{\text{SL}} \exists l_1 \exists r_1 . x \mapsto (l_1, r_1) \Rightarrow \ \theta = \{(l_1, l_0), (r_1, r_0)\}
\end{align*}
\]

LU + RU + RD encode a transition action in the NFTA inclusion check
Inference rules - SP (split)

\[
\begin{align*}
\text{SP} & \quad tree^+_2(l_0) \vdash tree^+_1(l_0) \\
\text{RD} & \quad tree^+_2(l_0), tree^+_2(r_0) \vdash tree^+_1(l_0) \ast tree(r_0), tree(l_0) \ast tree^+_1(r_0) \\
\end{align*}
\]

\[
\begin{align*}
x \mapsto (l_0, r_0), & \quad tree^+_2(l_0), tree^+_2(r_0) \vdash x \mapsto (\text{nil, nil}), \\
& \quad \exists l_1 \exists r_1 . x \mapsto (l_1, r_1) \ast tree^+_1(l_1) \ast tree(r_1), \\
& \quad \exists l_1 \exists r_1 . x \mapsto (l_1, r_1) \ast tree(l_1) \ast tree^+_1(r_1) \\
\text{RU} & \quad x \mapsto (l_0, r_0), tree^+_2(l_0), tree^+_2(r_0) \vdash tree^+_1(x) \\
\text{LU} & \quad tree^+_2(x) \vdash tree^+_1(x) \\
\end{align*}
\]

\[
\begin{align*}
& \quad tree^+_2(l_0) \vdash tree^+_1(l_0), tree(l_0) \\
& \quad tree^+_2(l_0) \vdash tree^+_1(l_0) \\
& \quad tree^+_2(l_0) \vdash tree(l_0) \\
& \quad tree^+_2(r_0) \vdash tree(r_0), tree^+_1(r_0) \\
\Rightarrow & \quad tree^+_2(l_0) \vdash tree^+_1(l_0) \\
& \quad tree^+_2(l_0) \vdash tree(l_0) \\
& \quad tree^+_2(r_0) \vdash tree(r_0), tree^+_1(r_0)
\end{align*}
\]
Inference rules - ID (infinite descent)

\[
\begin{align*}
\text{ID} & \quad \Gamma \\
\hline
\text{tree}_2^+(l_0) & \vdash \text{tree}_1^+(l_0)
\end{align*}
\]

\[
\begin{align*}
\text{SP} & \quad \text{tree}_2^+(l_0), \text{tree}_2^+(r_0) \vdash \text{tree}_1^+(l_0) \cdot \text{tree}(r_0), \text{tree}(l_0) \cdot \text{tree}_1^+(r_0)
\end{align*}
\]

\[
\begin{align*}
\text{RD} & \quad x \mapsto (l_0, r_0), \text{tree}_2^+(l_0), \text{tree}_2^+(r_0) \vdash x \mapsto (\text{nil}, \text{nil}), \\
& \quad \exists l_1 \exists r_1. x \mapsto (l_1, r_1) \cdot \text{tree}_1^+(l_1) \cdot \text{tree}(r_1), \\
& \quad \exists l_1 \exists r_1. x \mapsto (l_1, r_1) \cdot \text{tree}(l_1) \cdot \text{tree}_1^+(r_1)
\end{align*}
\]

\[
\begin{align*}
\text{RU} & \quad x \mapsto (l_0, r_0), \text{tree}_2^+(l_0), \text{tree}_2^+(r_0) \vdash \text{tree}_1^+(x)
\end{align*}
\]

\[
\begin{align*}
\text{LU} & \quad \text{tree}_2^+(x) \vdash \text{tree}_1^+(x)
\end{align*}
\]

**Backlink** from $\Gamma \theta \vdash \Delta' \theta$ 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 Implementation and experimental evaluation

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

If $h_1 = h \cup h_2$, then $h_2 \leq h_1$ and $h_2 \prec h_1$ if, moreover, $h \neq \emptyset$.

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

Necessary to ensure progress along a recurring branch closed by ID.
Soundness

Inference rules are sound for entailments with ranked inductive definitions

LU, RU, RD, SP, AX are locally sound

Global soundness of proofs using ID: a counterexample $cex_1$ for the root $\Gamma_1 \vdash \Delta_1$ of a proof can be propagated along an infinite trace

\[
\Gamma_1 \vdash \Delta_1 \quad \Gamma_2 \vdash \Delta_2 \quad \ldots \quad \Gamma_i \vdash \Delta_i \quad \Gamma_{i+1} \vdash \Delta_{i+1} \quad \ldots
\]

\[
cex_1 \triangleright cex_2 \triangleright \ldots \triangleright cex_i \triangleright cex_{i+1} \triangleright \ldots
\]

Ranked + LU required by ID $\Rightarrow$ counterexample strictly decreases infinitely often, contradicting the well-foundedness of the chosen domain
Completeness

Three additional semantic restrictions on the constraints of an inductive system - safe removal of constraints from sequents when using $\text{RD}$

- Reduce to checking satisfiability of $\exists^*\forall^*$-quantified SL formulae, which is PSPACE-complete for the $\neg^*$-free fragment

Proofs can be discovered by employing a search strategy

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

Only for definitions generating matching coverage trees of the heap models
Completeness

Only for definitions generating matching coverage trees of the heap models

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

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

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

\[
\overline{v} = \langle 1, 4 \rangle \text{ and } h = \{(1, 2), (2, 3), (3, 4)\} \text{ solution for both } ls \text{ and } ls^r, \text{ but the two definitions generate different coverage trees for } h
\]

\[
t = \{(1, 2)\}
\]

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

\[
\{(3, 4)\}
\]

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

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

\[
\{(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 Implementation and experimental evaluation

5 Conclusions
Inductor – checker for inductive entailments in SL

Written in C++, queries CVC4 to check restrictions and noninductive entailments when applying AX or RD

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

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

Output: **valid** + proof, **invalid** + counterexample
## Experimental evaluation

<table>
<thead>
<tr>
<th>LHS</th>
<th>RHS</th>
<th>R</th>
<th>Sequents</th>
<th>Rules</th>
<th>$H_{\text{Seq}}$</th>
<th>$H_{\text{LU}}$</th>
<th>$H_{\text{SP}}$</th>
<th>T</th>
<th>CVC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{tree}_1^+$</td>
<td>$\text{tree}$</td>
<td>$\checkmark$</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>$\text{tree}_2^+$</td>
<td>$\text{tree}$</td>
<td>$\checkmark$</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>$\text{tree}_1^+$</td>
<td>$\text{tree}_1^+$</td>
<td>$\checkmark$</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>$\text{tree}$</td>
<td>$\text{tree}_1^+$</td>
<td>$\neg$</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>$\text{tree}$</td>
<td>$\text{tree}_2^+$</td>
<td>$\neg$</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>$\text{tree}_1^+$</td>
<td>$\text{tree}_2^+$</td>
<td>$\neg$</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>$\text{ls}^a$</td>
<td>$\text{ls}$</td>
<td>$\checkmark$</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>$\text{ls}$</td>
<td>$\text{ls}^a$</td>
<td>$\neg$</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>
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 Implementation and experimental evaluation

5 Conclusions
Conclusions

Cyclic proof system for entailments between inductive predicates in SL

Infinite Descent principle, inductive invariants produced during proof search

Semantic boundaries within which soundness and completeness are ensured

Inductor uses a search strategy to explore possible derivations and outputs a proof if an entailment is shown to be valid, or counterexamples if it is not