Compositional Entailment Checking for a Fragment of Separation Logic

Constantin Enea   Mihaela Sighireanu

LIAFA, Université Paris Diderot, France

Ondřej Lengál   **Tomáš Vojnar**

Brno University of Technology, Czech Republic

Vienna UT, 2015

Introduction

- Procedure for checking entailments in separation logic.

- Separation logic (SL):
  - a formalism for reasoning about heaps,
  - allows for scalability: local reasoning,
  - used, e.g., in Space Invader, Slayer, HIP/SLEEK, Predator, S2, . . .

Introduction

- Procedure for checking entailments in separation logic.

- Separation logic (SL):
    - a formalism for reasoning about heaps,
    - allows for scalability: local reasoning,
    - used, e.g., in Space Invader, Slayer, HIP/SLEEK, Predator, S2, ...

- Reasoning about heap-manipulating programs:
    - crucial for many program analysis tasks,
    - difficult: ∞ sets of graphs,
    - still under heavy research.

Separation Logic

- Basic formulae of SL:

$$\begin{array}{lcll}
\varphi & ::= & \exists x_1, ..., x_n \,.\, \Pi \wedge \Sigma & \\
\Pi & ::= & x_1 = x_2 \mid x_1 \neq x_2 \mid x = \texttt{null} \mid \Pi_1 \wedge \Pi_2 & \text{pure part} \\
\Sigma & ::= & emp \mid x \mapsto \{(f_1, x_1), \ldots, (f_n, x_n)\} \mid \Sigma_1 * \Sigma_2 & \text{shape part}
\end{array}$$

- Example:

$$\varphi = \exists x_1 \,.\, E \mapsto \{(\texttt{next}, x_1)\} * x_1 \mapsto \{(\texttt{next}, F)\}$$

Separation Logic

- Basic formulae of SL:

$$\begin{array}{lcll} \varphi & ::= & \exists x_1, ..., x_n \,.\, \Pi \wedge \Sigma & \\ \Pi & ::= & x_1 = x_2 \mid x_1 \neq x_2 \mid x = \mathtt{null} \mid \Pi_1 \wedge \Pi_2 & \text{pure part} \\ \Sigma & ::= & emp \mid x \mapsto \{(f_1, x_1), \ldots, (f_n, x_n)\} \mid \Sigma_1 * \Sigma_2 & \text{shape part} \end{array}$$

- Example:

$$\varphi = \exists x_1 \,.\, E \mapsto \{(\mathtt{next}, x_1)\} * x_1 \mapsto \{(\mathtt{next}, F)\}$$

- Inductive predicates:
  - Abstraction:
    - a data structure of any length (size) via recursion.
  - Example (singly linked list):

$$\begin{array}{rl} sll(E, F) \overset{\text{def}}{=} & (E = F \wedge emp) \vee \\ & \big(E \neq F \wedge \exists X_{tl} \,.\, E \mapsto \{(\mathtt{next}, X_{tl})\} * sll(X_{tl}, F)\big) \end{array}$$

Entailments in Separation Logic (1/2)

$$\varphi \overset{?}{\models} \psi$$

Is φ an unfolding of ψ?

- Example:

$$\exists x_1, x_2 \,.\, E \mapsto \{(\mathtt{next}, x_1)\} * sll(x_1, x_2) * x_2 \mapsto \{(\mathtt{next}, F)\}$$

$$\overset{?}{\models} sll(E, F)$$

- where

$$\begin{aligned} sll(E, F) \overset{\text{def}}{=} \; & (E = F \wedge emp) \vee \\ & \big(E \neq F \wedge \exists X_{tl} \,.\, E \mapsto \{(\mathtt{next}, X_{tl})\} * sll(X_{tl}, F)\big) \end{aligned}$$

Entailments in Separation Logic (2/2)

- Invariant checking for heap-manipulating programs:
  - resolving verification conditions in deductive verification,
  - fixpoint checking in abstract interpretation-based approaches.
- In general undecidable.
- There exist decision procedures for various fragments.
  - In what follows, one such fragment and a decision procedure are presented.
  - Originally published at APLAS'14.

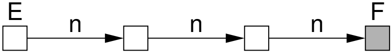
Considered Singly-Linked Fragment (1/3)

We start with various kinds of singly-linked lists expressible using the template:

$$P(E, F, \overrightarrow{B}) = (E = F \wedge emp) \vee \\ \left(E \notin \{F\} \cup \overrightarrow{B} \wedge \exists X_{tl} \exists \overrightarrow{\mathbf{Z}} \,.\, \mathbf{\Sigma}(\mathbf{E}, \mathbf{X}_{tl}, \overrightarrow{Z} \cup \overrightarrow{B}) * P(X_{tl}, F, \overrightarrow{B})\right)$$

This template allows us to express:

- Simple singly-linked lists (SLLs):



$\models sll(E, F)$

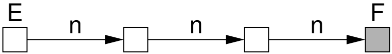
Considered Singly-Linked Fragment (1/3)

We start with various kinds of singly-linked lists expressible using the template:

$$P(E, F, \overrightarrow{B}) = (E = F \wedge emp) \vee$$
$$\left(E \notin \{F\} \cup \overrightarrow{B} \wedge \exists X_{tl} \exists \overrightarrow{\mathbf{Z}} \,.\, \mathbf{\Sigma}(\mathbf{E}, \mathbf{X}_{tl}, \overrightarrow{Z} \cup \overrightarrow{B}) * P(X_{tl}, F, \overrightarrow{B})\right)$$

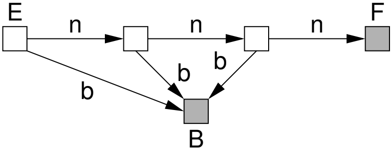
This template allows us to express:

- Simple singly-linked lists (SLLs):



$\models sll(E, F)$

- SLLs with additional (e.g. head/tail) pointers:

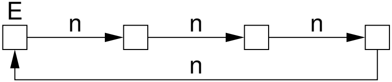


$\models sllb(E, F, B)$

Considered Singly-Linked Fragment (2/3)

$$P(E,F,\overrightarrow{B}) = (E = F \wedge emp) \vee$$
$$\left(E \notin \{F\} \cup \overrightarrow{B} \wedge \exists X_{tl} \exists \overrightarrow{\mathbf{Z}} \,.\, \mathbf{\Sigma}(\mathbf{E}, \mathbf{X}_{tl}, \overrightarrow{Z} \cup \overrightarrow{B}) * P(X_{tl}, F, \overrightarrow{B})\right)$$

- Cyclic lists:

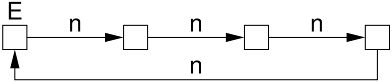


$\models sllc(E)$

Considered Singly-Linked Fragment (2/3)

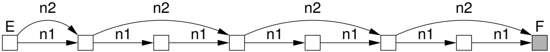
$$P(E, F, \overrightarrow{B}) = (E = F \wedge emp) \vee$$
$$\left(E \notin \{F\} \cup \overrightarrow{B} \wedge \exists X_{tl} \exists \overrightarrow{\mathbf{Z}} \,.\, \mathbf{\Sigma}(\mathbf{E}, \mathbf{X}_{tl}, \overrightarrow{Z} \cup \overrightarrow{B}) * P(X_{tl}, F, \overrightarrow{B})\right)$$

- Cyclic lists:

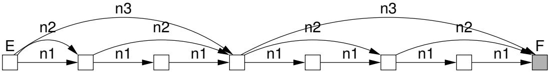


$\models sllc(E)$

- Skip lists:



$\models skl2(E, F)$

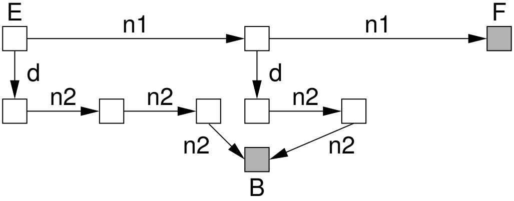


$\models skl3(E, F)$

Considered Singly-Linked Fragment (3/3)

$$P(E, F, \overrightarrow{B}) = (E = F \wedge emp) \vee$$
$$\left(E \notin \{F\} \cup \overrightarrow{B} \wedge \exists X_{tl} \exists \overrightarrow{\mathbf{Z}} \,.\, \mathbf{\Sigma}(\mathbf{E}, \mathbf{X}_{tl}, \overrightarrow{Z} \cup \overrightarrow{B}) * P(X_{tl}, F, \overrightarrow{B})\right)$$

- Nested combinations of the above:



$\models nsll(E, F, B)$

Overview

$$\underbrace{\exists \overrightarrow{X} . \Pi_{\varphi} \wedge \Sigma_{\varphi}}_{\varphi} \overset{?}{\models} \underbrace{\Pi_{\psi} \wedge \Sigma_{\psi}}_{\psi}$$

1. Normalize φ and ψ:
   - add implied (dis)equalities,
   - remove empty inductive predicates.

Overview

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_\varphi \land \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \land \Sigma_\psi}_{\psi}$$

1. Normalize φ and ψ:
    - add implied (dis)equalities,
    - remove empty inductive predicates.
2. Test entailment of normalized pure parts (is $\Pi_\varphi \Rightarrow \Pi_\psi$ SAT?).

Overview

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_\varphi \wedge \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \wedge \Sigma_\psi}_{\psi}$$

1. Normalize φ and ψ:
    - add implied (dis)equalities,
    - remove empty inductive predicates.
2. Test entailment of normalized pure parts (is $\Pi_\varphi \Rightarrow \Pi_\psi$ SAT?).
3. Match every points-to $x \mapsto \{\dots\}$ in Σ_ψ with a points-to in Σ_φ.

Overview

$$\underbrace{\exists \overrightarrow{X}\,.\,\Pi_\varphi \wedge \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \wedge \Sigma_\psi}_{\psi}$$

1. Normalize φ and ψ:
    - add implied (dis)equalities,
    - remove empty inductive predicates.
2. Test entailment of normalized pure parts (is $\Pi_\varphi \Rightarrow \Pi_\psi$ SAT?).
3. Match every points-to $x \mapsto \{\dots\}$ in Σ_ψ with a points-to in Σ_φ.
4. Reduce the rest of Σ_φ and Σ_ψ to:

$$\varphi_1 \overset{?}{\models} P_1 \quad \wedge \quad \varphi_2 \overset{?}{\models} P_2 \quad \wedge \quad \varphi_3 \overset{?}{\models} P_3 \quad \wedge \quad \dots$$

    ① Transform $\varphi_i \rightsquigarrow$ tree $\mathcal{T}_{\varphi_i}$:
        - spanning tree + routing expressions.

Overview

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_\varphi \wedge \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \wedge \Sigma_\psi}_{\psi}$$

1. Normalize φ and ψ:
   - add implied (dis)equalities,
   - remove empty inductive predicates.
2. Test entailment of normalized pure parts (is $\Pi_\varphi \Rightarrow \Pi_\psi$ SAT?).
3. Match every points-to $x \mapsto \{\dots\}$ in Σ_ψ with a points-to in Σ_φ.
4. Reduce the rest of Σ_φ and Σ_ψ to:

$$\varphi_1 \overset{?}{\models} P_1 \quad \wedge \quad \varphi_2 \overset{?}{\models} P_2 \quad \wedge \quad \varphi_3 \overset{?}{\models} P_3 \quad \wedge \quad \dots$$

   1. Transform $\varphi_i \rightsquigarrow$ tree $\mathcal{T}_{\varphi_i}$:
      - spanning tree + routing expressions.
   2. Transform $P_i \rightsquigarrow$ tree automaton $\mathcal{A}_{P_i}$:
      - all unfoldings of P_i.

Overview

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_\varphi \wedge \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \wedge \Sigma_\psi}_{\psi}$$

1. Normalize φ and ψ:
   - add implied (dis)equalities,
   - remove empty inductive predicates.
2. Test entailment of normalized pure parts (is $\Pi_\varphi \Rightarrow \Pi_\psi$ SAT?).
3. Match every points-to $x \mapsto \{\dots\}$ in Σ_ψ with a points-to in Σ_φ.
4. Reduce the rest of Σ_φ and Σ_ψ to:

$$\varphi_1 \overset{?}{\models} P_1 \quad \wedge \quad \varphi_2 \overset{?}{\models} P_2 \quad \wedge \quad \varphi_3 \overset{?}{\models} P_3 \quad \wedge \quad \dots$$

   1. Transform $\varphi_i \rightsquigarrow$ tree $\mathcal{T}_{\varphi_i}$:
      - spanning tree + routing expressions.
   2. Transform $P_i \rightsquigarrow$ tree automaton $\mathcal{A}_{P_i}$:
      - all unfoldings of P_i.
   3. Test:

$$\mathcal{T}_{\varphi_i} \overset{?}{\in} \mathcal{L}(\mathcal{A}_{P_i})$$

Normalization (1/3)

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_\varphi \wedge \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \wedge \Sigma_\psi}_{\psi}$$

Boolean abstractions of φ and ψ:

- A conjunctive formula constructed as follows:
- Start with $\emptyset$ and process the **pure** part first.

Normalization (1/3)

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_\varphi \wedge \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \wedge \Sigma_\psi}_{\psi}$$

Boolean abstractions of φ and ψ:

- A conjunctive formula constructed as follows:
- Start with $\emptyset$ and process the **pure** part first.
- Add an encoding of Π:
  - For $E = F$ in Π, add $[E = F]$ for a Boolean variable $[E = F]$.
  - For $E \neq F$ in Π, add $\neg[E = F]$.

Normalization (1/3)

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_\varphi \wedge \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \wedge \Sigma_\psi}_{\psi}$$

Boolean abstractions of φ and ψ:

- A conjunctive formula constructed as follows:
- Start with $\emptyset$ and process the **pure** part first.
- Add an encoding of Π:
  - For $E = F$ in Π, add $[E = F]$ for a Boolean variable $[E = F]$.
  - For $E \neq F$ in Π, add $\neg[E = F]$.
- Add an encoding of **equality**:
  - reflexivity: $[E = E]$ for all E,
  - symmetry: $[E = F] \Leftrightarrow [F = E]$ for all E, F,
  - transitivity: $[E = F] \wedge [F = G] \Rightarrow [E = G]$ for all E, F, G.

Normalization (2/3)

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_\varphi \wedge \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \wedge \Sigma_\psi}_{\psi}$$

Boolean abstractions of φ and ψ:

- ***Pure*** *part* Π *finished, now do the* ***shape*** Σ.
- Add an encoding of Σ:
  - For $a = E \mapsto \{f_1 \mapsto x_1, \ldots\}$ in Σ, add $[E, a]$. (E is allocated in a.)
  - For $a = P(E, F, \overrightarrow{B})$ in Σ, add $[E, a] \oplus [E = F]$.

Normalization (2/3)

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_\varphi \wedge \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \wedge \Sigma_\psi}_{\psi}$$

Boolean abstractions of φ and ψ:

- ***Pure*** *part* Π *finished, now do the* ***shape*** Σ.
- Add an encoding of Σ:
  - For $a = E \mapsto \{f_1 \mapsto x_1, \dots\}$ in Σ, add $[E, a]$. (E is allocated in a.)
  - For $a = P(E, F, \overrightarrow{B})$ in Σ, add $[E, a] \oplus [E = F]$.
- Add an encoding of **separating conjunction** $*$:
  - $\big([E = F] \wedge [E, a]\big) \Rightarrow \neg[F, a']$ for all $E, F, a \neq a'$.

Normalization (3/3)

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_\varphi \land \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \land \Sigma_\psi}_{\psi}$$

Boolean abstraction of φ (for ψ similar):

- Properties of *BoolAbs*[φ]:
  - φ and *BoolAbs*[φ] are equisatisfiable.
  - $\varphi \Rightarrow E = F$ iff *BoolAbs*[φ] $\Rightarrow [E = F]$.
  - $\varphi \Rightarrow E \neq F$ iff *BoolAbs*[φ] $\Rightarrow \neg[E = F]$.

Normalization (3/3)

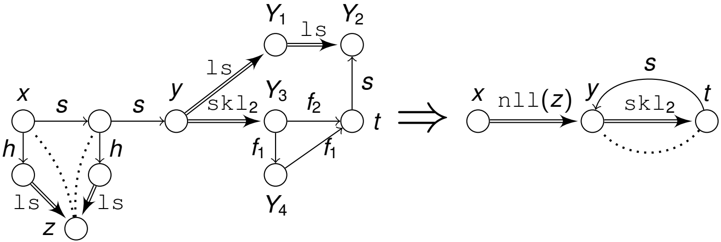
$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_\varphi \wedge \Sigma_\varphi}_{\varphi} \overset{?}{\models} \underbrace{\Pi_\psi \wedge \Sigma_\psi}_{\psi}$$

Boolean abstraction of φ (for ψ similar):

- Properties of *BoolAbs*[φ]:
  - φ and *BoolAbs*[φ] are equisatisfiable.
  - $\varphi \Rightarrow E = F$ iff *BoolAbs*[φ] $\Rightarrow [E = F]$.
  - $\varphi \Rightarrow E \neq F$ iff *BoolAbs*[φ] $\Rightarrow \neg[E = F]$.
- Normalization of φ:
  - If *BoolAbs*[φ] is UNSAT:
    - φ' `:= false; return` φ'`;`
  - If *BoolAbs*[φ] is SAT:
    - φ' `:=` φ`;`
    - φ' `:=` $\varphi' \cup$ (dis)equalities implied by *BoolAbs*[φ]`;`
    - φ' `:=` $\varphi' \setminus$ empty inductive predicates`;` // $P(E, F, \overrightarrow{B})$ *s.t.* $E = F$ *in* φ'
    - `return` φ'`;`

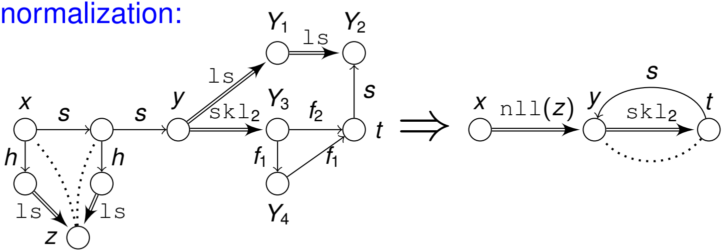
A Complex Entailment Example $\psi_1 \Rightarrow \psi_2$ (1/3)

$$\begin{aligned}
\psi_1 \quad &\equiv \quad \exists Y_1, Y_2, Y_3, Y_4, Z_1, Z_2, Z_3 .\, x \neq z \wedge Z_2 \neq z \wedge \\
&\qquad x \mapsto \{(s, Z_2), (h, Z_1)\} * Z_2 \mapsto \{(s, y), (h, Z_3)\} * \mathtt{ls}(Z_1, z) * \mathtt{ls}(Z_3, z) \\
&\qquad \mathtt{ls}(y, Y_1) * \mathtt{skl}_2(y, Y_3) * \mathtt{ls}(Y_1, Y_2) * \\
&\qquad Y_3 \mapsto \{(f_2, t), (f_1, Y_4)\} * Y_4 \mapsto \{(f_2, \mathtt{null}), (f_1, t)\} * t \mapsto \{(s, Y_2)\} \\
\psi_2 \quad &\equiv \quad y \neq t \wedge \mathtt{nll}(x, y, z) * \mathtt{skl}_2(y, t) * t \mapsto \{(s, y)\}
\end{aligned}$$

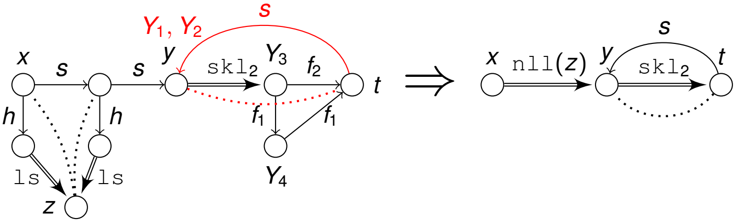


A Complex Entailment Example $\psi_1 \Rightarrow \psi_2$ (2/3)

Before normalization:



After normalization:



Entailment of Shape Parts

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_{\varphi'} \wedge \Sigma_{\varphi'}}_{\varphi'} \overset{?}{\models} \underbrace{\Pi_{\psi'} \wedge \Sigma_{\psi'}}_{\psi'}$$

For every shape atom of $\Sigma_{\psi'}$, find a subformula of $\Sigma_{\varphi'}$:

- For each points-to $E \mapsto \{(f_1, x_1), \ldots\}$ in $\Sigma_{\psi'}$:
  - find $E \mapsto \{(f_1, x_1), \ldots\}$ in Σ_{φ}.

Entailment of Shape Parts

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_{\varphi'} \wedge \Sigma_{\varphi'}}_{\varphi'} \overset{?}{\models} \underbrace{\Pi_{\psi'} \wedge \Sigma_{\psi'}}_{\psi'}$$

For every shape atom of $\Sigma_{\psi'}$, find a subformula of $\Sigma_{\varphi'}$:

- For each points-to $E \mapsto \{(f_1, x_1), \ldots\}$ in $\Sigma_{\psi'}$:
  - find $E \mapsto \{(f_1, x_1), \ldots\}$ in Σ_{φ}.
- For inductive predicates $P(E, F, \overrightarrow{B})$ in $\Sigma_{\psi'}$:
  - in the order from the most specialized to the most general,

Entailment of Shape Parts

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_{\varphi'} \wedge \Sigma_{\varphi'}}_{\varphi'} \overset{?}{\models} \underbrace{\Pi_{\psi'} \wedge \Sigma_{\psi'}}_{\psi'}$$

For every shape atom of $\Sigma_{\psi'}$, find a subformula of $\Sigma_{\varphi'}$:

- For each points-to $E \mapsto \{(f_1, x_1), \dots\}$ in $\Sigma_{\psi'}$:
  - find $E \mapsto \{(f_1, x_1), \dots\}$ in Σ_{φ}.
- For inductive predicates $P(E, F, \overrightarrow{B})$ in $\Sigma_{\psi'}$:
  - in the order from the most specialized to the most general,
  - look at $\Sigma_{\varphi'}$ as a graph,

Entailment of Shape Parts

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_{\varphi'} \wedge \Sigma_{\varphi'}}_{\varphi'} \overset{?}{\models} \underbrace{\Pi_{\psi'} \wedge \Sigma_{\psi'}}_{\psi'}$$

For every shape atom of $\Sigma_{\psi'}$, find a subformula of $\Sigma_{\varphi'}$:

- For each points-to $E \mapsto \{(f_1, x_1), \dots\}$ in $\Sigma_{\psi'}$:
  - find $E \mapsto \{(f_1, x_1), \dots\}$ in Σ_{φ}.
- For inductive predicates $P(E, F, \overrightarrow{B})$ in $\Sigma_{\psi'}$:
  - in the order from the most specialized to the most general,
  - look at $\Sigma_{\varphi'}$ as a graph,
  - select a subgraph G of $\Sigma_{\varphi'}$ corresponding to E, F, and $\overrightarrow{B}$,

Entailment of Shape Parts

$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_{\varphi'} \wedge \Sigma_{\varphi'}}_{\varphi'} \overset{?}{\models} \underbrace{\Pi_{\psi'} \wedge \Sigma_{\psi'}}_{\psi'}$$

For every shape atom of $\Sigma_{\psi'}$, find a subformula of $\Sigma_{\varphi'}$:

- For each points-to $E \mapsto \{(f_1, x_1), \dots\}$ in $\Sigma_{\psi'}$:
  - find $E \mapsto \{(f_1, x_1), \dots\}$ in Σ_{φ}.
- For inductive predicates $P(E, F, \overrightarrow{B})$ in $\Sigma_{\psi'}$:
  - in the order from the most specialized to the most general,
  - look at $\Sigma_{\varphi'}$ as a graph,
  - select a subgraph G of $\Sigma_{\varphi'}$ corresponding to E, F, and $\overrightarrow{B}$,
  - transform G into a tree $\mathcal{T}_G$ with the root E,

Entailment of Shape Parts

$$\underbrace{\exists \overrightarrow{X} . \Pi_{\varphi'} \wedge \Sigma_{\varphi'}}_{\varphi'} \overset{?}{\models} \underbrace{\Pi_{\psi'} \wedge \Sigma_{\psi'}}_{\psi'}$$

For every shape atom of $\Sigma_{\psi'}$, find a subformula of $\Sigma_{\varphi'}$:

- For each points-to $E \mapsto \{(f_1, x_1), \dots\}$ in $\Sigma_{\psi'}$:
    - find $E \mapsto \{(f_1, x_1), \dots\}$ in Σ_{φ}.
- For inductive predicates $P(E, F, \overrightarrow{B})$ in $\Sigma_{\psi'}$:
    - in the order from the most specialized to the most general,
    - look at $\Sigma_{\varphi'}$ as a graph,
    - select a subgraph G of $\Sigma_{\varphi'}$ corresponding to E, F, and $\overrightarrow{B}$,
    - transform G into a tree $\mathcal{T}_G$ with the root E,
    - transform $P(E, F, \overrightarrow{B})$ into a tree automaton $\mathcal{A}_{P(E,F,\overrightarrow{B})}$,

Entailment of Shape Parts

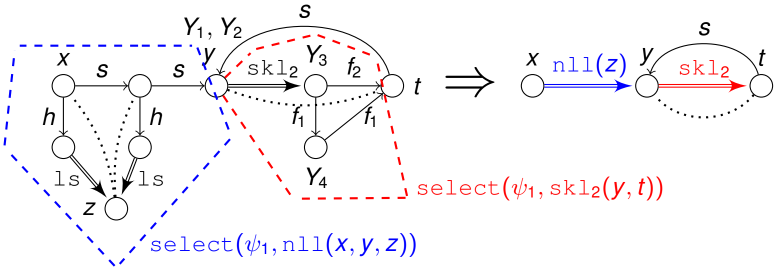
$$\underbrace{\exists \overrightarrow{X} \,.\, \Pi_{\varphi'} \wedge \Sigma_{\varphi'}}_{\varphi'} \overset{?}{\models} \underbrace{\Pi_{\psi'} \wedge \Sigma_{\psi'}}_{\psi'}$$

For every shape atom of $\Sigma_{\psi'}$, find a subformula of $\Sigma_{\varphi'}$:

- For each points-to $E \mapsto \{(f_1, x_1), \dots\}$ in $\Sigma_{\psi'}$:
  - find $E \mapsto \{(f_1, x_1), \dots\}$ in Σ_{φ}.
- For inductive predicates $P(E, F, \overrightarrow{B})$ in $\Sigma_{\psi'}$:
  - in the order from the most specialized to the most general,
  - look at $\Sigma_{\varphi'}$ as a graph,
  - select a subgraph G of $\Sigma_{\varphi'}$ corresponding to E, F, and $\overrightarrow{B}$,
  - transform G into a tree $\mathcal{T}_G$ with the root E,
  - transform $P(E, F, \overrightarrow{B})$ into a tree automaton $\mathcal{A}_{P(E,F,\overrightarrow{B})}$,
  - test: $\mathcal{T}_G \overset{?}{\in} \mathcal{L}(\mathcal{A}_{P(E,F,\overrightarrow{B})})$

A Complex Entailment Example $\psi_1 \Rightarrow \psi_2$ (3/3)

Selected subgraphs:



Entailment of Shape Parts

Transforming Graphs into Trees (1/2)

Identify a unique spanning tree of a rooted graph:

- Construct an ordering on selectors such that:
  - selectors of a predicate $\prec$ selectors of its nested predicates,
  - “forward” selectors $\prec$ selectors going “down” to nested predicates,
  - selectors going “down” $\prec$ selectors going to the border.

- Spanning tree edges: a part of minimal paths to particular nodes.

Entailment of Shape Parts

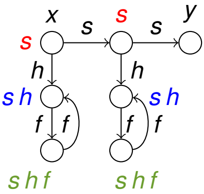
Transforming Graphs into Trees (1/2)

Identify a unique spanning tree of a rooted graph:

- Construct an ordering on selectors such that:
  - selectors of a predicate $\prec$ selectors of its nested predicates,
  - "forward" selectors $\prec$ selectors going "down" to nested predicates,
  - selectors going "down" $\prec$ selectors going to the border.
- Spanning tree edges: a part of minimal paths to particular nodes.

Label nodes with minimal compacted paths:

- each f^n in a path, $n \geq 1$, replaced by f,
- identifies repeated nodes of the same kind.



Entailment of Shape Parts

Transforming Graphs into Trees (2/2)

Split every join node ($>$ 1 incoming edges) into several copies:

- one for every incoming edge $\notin ST$,
- label every copy with an alias:
  - paths to variables: $alias[M], M \in Vars$,
  - go "up" to the closest *label* (loops): $alias{\uparrow}[label]$,
  - go "up" and then "down" (multiple joining paths)): $alias{\uparrow\downarrow}[label]$.
- If no alias available, return $\perp$ (out of fragment).

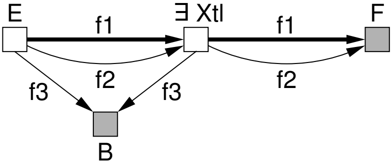
Entailment of Shape Parts

Transforming Graphs into Trees (2/2)

Split every join node ($>$ 1 incoming edges) into several copies:

- one for every incoming edge $\notin ST$,
- label every copy with an alias:
  - paths to variables: $alias[M], M \in Vars$,
  - go "up" to the closest *label* (loops): $alias{\uparrow}[label]$,
  - go "up" and then "down" (multiple joining paths)): $alias{\uparrow\downarrow}[label]$.
- If no alias available, return $\perp$ (out of fragment).

For example:



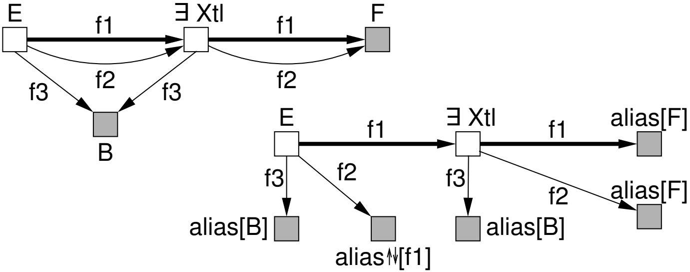
Entailment of Shape Parts

Transforming Graphs into Trees (2/2)

Split every join node (>1 incoming edges) into several copies:

- one for every incoming edge $\notin ST$,
- label every copy with an alias:
  - paths to variables: $alias[M], M \in Vars$,
  - go “up” to the closest *label* (loops): $alias{\uparrow}[label]$,
  - go “up” and then “down” (multiple joining paths)): $alias{\uparrow\downarrow}[label]$.
- If no alias available, return $\perp$ (out of fragment).

For example:



Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata (1/3)

Derive a TA encoding all possible partial as well as full unfoldings of P where folded and unfolded parts may arbitrarily interleave:

Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata (1/3)

Derive a TA encoding all possible partial as well as full unfoldings of P where folded and unfolded parts may arbitrarily interleave:

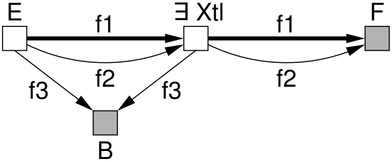
1. Unfold the predicate P twice $\rightsquigarrow P^{[2]}$:
   - necessary to capture all possible alias relations we use.

Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata (1/3)

Derive a TA encoding all possible partial as well as full unfoldings of *P* where folded and unfolded parts may arbitrarily interleave:

1. Unfold the predicate *P* twice $\leadsto P^{[2]}$:
   - necessary to capture all possible alias relations we use.
   - E.g., $\Sigma_P(E, X_{tl}, B) = E \mapsto \{(f_1, X_{tl}), (f_2, X_{tl})(f_3, B)\}$:

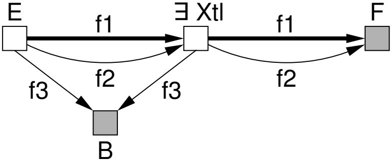


Entailment of Shape Parts

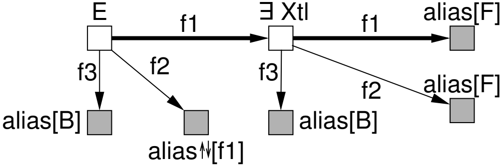
Transforming Inductive Predicates into Tree Automata (1/3)

Derive a TA encoding all possible partial as well as full unfoldings of *P* where folded and unfolded parts may arbitrarily interleave:

1. Unfold the predicate *P* twice $\rightsquigarrow P^{[2]}$:
    - necessary to capture all possible alias relations we use.
    - E.g., $\Sigma_P(E, X_{tl}, B) = E \mapsto \{(f_1, X_{tl}), (f_2, X_{tl})(f_3, B)\}$:



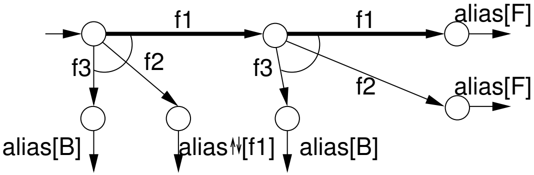
2. Get the tree encoding $\mathcal{T}_{P^{[2]}}$:



Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata (2/3)

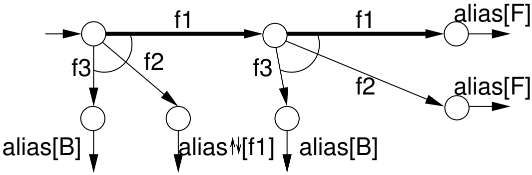
3. Transform $\mathcal{T}_{P[2]}$ into a tree automaton $\mathcal{A}_{P[2]}$ accepting $\{\mathcal{T}_{P[2]}\}$:



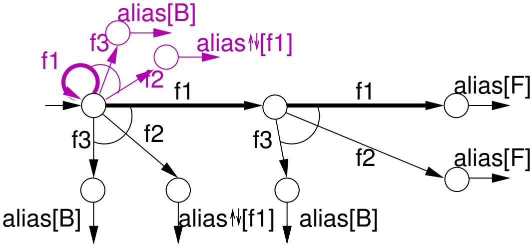
Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata (2/3)

3 Transform $\mathcal{T}_{P[2]}$ into a tree automaton $\mathcal{A}_{P[2]}$ accepting $\{\mathcal{T}_{P[2]}\}$:



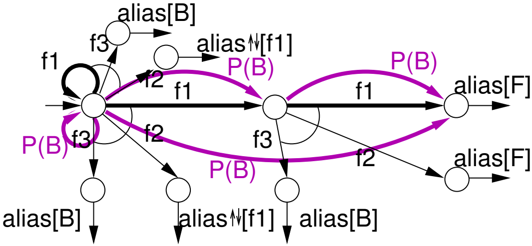
4 Add a loop enabling construction of the list backbone of size ≥ 2:



Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata (3/3)

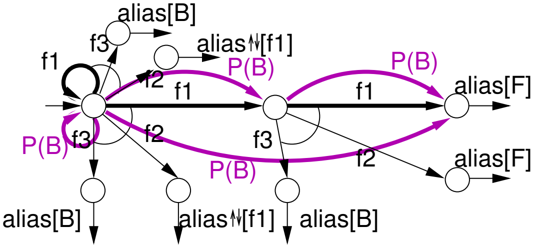
5. Duplicate backbone transitions with transitions over $P \rightsquigarrow \mathcal{A}_{P[2+]}$,
   - to enable arbitrary interleaving:



Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata (3/3)

5. Duplicate backbone transitions with transitions over $P \rightsquigarrow \mathcal{A}_{P[2+]}$,
   - to enable arbitrary interleaving:

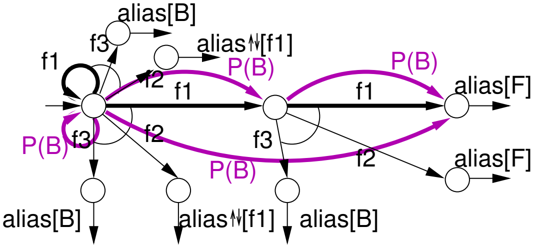


6. For every nested predicate edge over Q, insert $\mathcal{A}_Q$,
   - and modify the aliases appearing in $\mathcal{A}_Q$.

Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata (3/3)

5. Duplicate backbone transitions with transitions over $P \rightsquigarrow \mathcal{A}_{P[2+]}$,
   - to enable arbitrary interleaving:



6. For every nested predicate edge over Q, insert $\mathcal{A}_Q$,
   - and modify the aliases appearing in $\mathcal{A}_Q$.
7. Unite $\mathcal{A}_{P[2+]}$ with $\mathcal{A}_{P[1]} \rightsquigarrow \mathcal{A}_{P[1+]}$.

Soundness, Completeness & Complexity

The decision procedure is:

- sound,

Soundness, Completeness & Complexity

The decision procedure is:

- sound,
- polynomial (relative to SAT) and incomplete:
  - issues with possibly empty nested lists and consequent aliasing,

Soundness, Completeness & Complexity

The decision procedure is:

- sound,
- polynomial (relative to SAT) and incomplete:
  - issues with possibly empty nested lists and consequent aliasing,
- an easy extension – exponential and complete:
  - exponential in the maximum height of the hierarchy of nested predicates.

Doubly-Linked Lists

$$\begin{aligned}\texttt{dll}(E,F,P,S) = {} & (E = S \land F = P \land emp) \lor \\ & (E \neq S \land F \neq P \ \land \ \exists X_{\texttt{tl}}.\, E \mapsto \{(next, X_{\texttt{tl}}), (prev, P)\} * \texttt{dll}(X_{\texttt{tl}}, F, E, S)).\end{aligned}$$

Doubly-Linked Lists

$$\texttt{dll}(E, F, P, S) = (E = S \land F = P \land emp) \lor$$
$$(E \neq S \land F \neq P \ \land \ \exists X_{\texttt{tl}}.\, E \mapsto \{(next, X_{\texttt{tl}}), (prev, P)\} * \texttt{dll}(X_{\texttt{tl}}, F, E, S)).$$

A new kind of label is needed to close the next/prev loops:

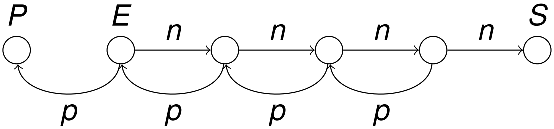
- second predecessor with a given label.

Doubly-Linked Lists

$$\mathtt{dll}(E, F, P, S) = (E = S \wedge F = P \wedge emp) \vee$$
$$(E \neq S \wedge F \neq P \ \wedge \ \exists X_{\mathtt{tl}}.\, E \mapsto \{(next, X_{\mathtt{tl}}), (prev, P)\} * \mathtt{dll}(X_{\mathtt{tl}}, F, E, S)).$$

A new kind of label is needed to close the next/prev loops:

- second predecessor with a given label.

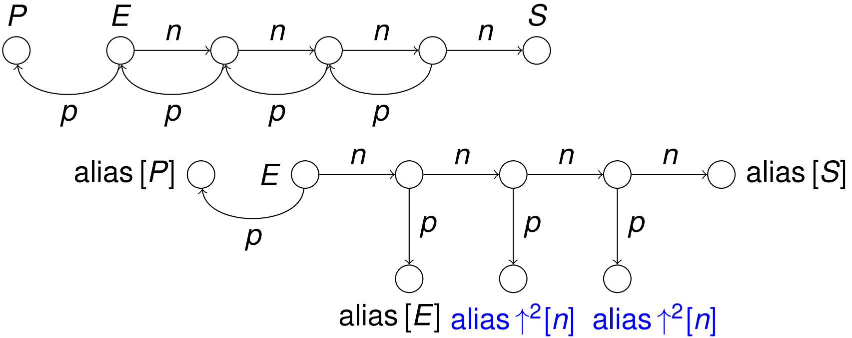


Doubly-Linked Lists

$$\texttt{dll}(E, F, P, S) = (E = S \wedge F = P \wedge emp) \vee$$
$$(E \neq S \wedge F \neq P \;\wedge\; \exists X_{\texttt{tl}}.\, E \mapsto \{(next, X_{\texttt{tl}}), (prev, P)\} * \texttt{dll}(X_{\texttt{tl}}, F, E, S)).$$

A new kind of label is needed to close the next/prev loops:

- second predecessor with a given label.

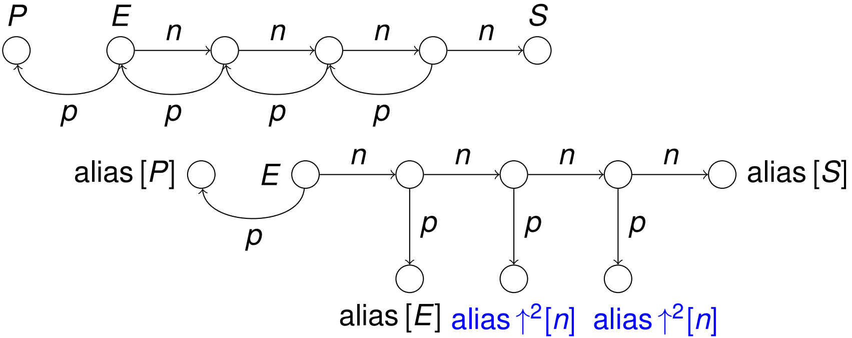


Doubly-Linked Lists

$$\mathtt{dll}(E, F, P, S) = (E = S \wedge F = P \wedge emp) \vee$$
$$(E \neq S \wedge F \neq P \;\wedge\; \exists X_{\mathtt{tl}}.\, E \mapsto \{(next, X_{\mathtt{tl}}), (prev, P)\} * \mathtt{dll}(X_{\mathtt{tl}}, F, E, S)).$$

A new kind of label is needed to close the next/prev loops:

- second predecessor with a given label.



One more kind of label needed for circular DLLs.

Experimental Results

Implemented in a solver **SPEN**.

- Input format: SMTLIB2,
  - extension for separation logic.
- Uses:
  - MINISAT,
  - VATA tree automata library.
- Benchmarks (from SL-COMP'14):
  - 292 `ls` problems: $< 8\,\mathrm{s}$ – 2nd place,
  - 43 "*fixed definitions*" problems – operations on:
    - nested singly-linked lists,
    - nested circular singly-linked lists,
    - 3-level skip lists,
    - doubly-linked lists.
    - average time: 0.35 s – 1st place.

Future work

- Generalize to a more expressive fragment of SL.
- Combine with reasoning about other kinds of data.
- Integrate into a program analysis framework.