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

APLAS'14

Introduction

- Procedure for checking entailments in separation logic

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

Introduction

- Procedure for checking entailments in separation logic

- Separation logic (SL)
  - formalism for reasoning about heaps
  - scalability — allows local reasoning
  - e.g. 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:

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

pure part

shape part

- Example:

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

Separation Logic

- Basic formulae of SL:

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

- Example:

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

- Inductive predicates:
  - abstraction
    - data structure of any length via recursion
  - Example (singly linked list):

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

Entailments in Separation Logic 1/2

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

Is φ an unfolding of ψ?

- Example:

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

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

- where

$$sll(E, F) \stackrel{\text{def}}{=} (E = F \wedge emp) \vee \big(E \neq F \wedge \exists X_{tl} \,.\, E \mapsto \{(\texttt{next}, X_{tl})\} * sll(X_{tl}, F)\big)$$

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
- our contribution: decision procedure for a practical fragment

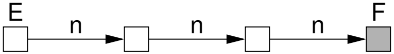
Considered Fragment 1/3

template for singly linked:

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

Supports various flavours of lists, including:

- **singly** linked lists



$\models sll(E, F)$

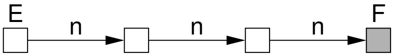
Considered Fragment 1/3

template for singly linked:

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

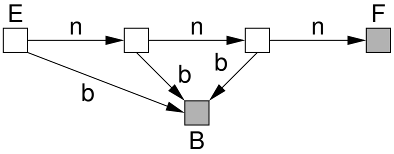
Supports various flavours of lists, including:

- **singly** linked lists



$\models sll(E, F)$

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

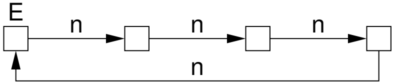


$\models sllb(E, F, B)$

Considered Fragment 2/3

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

- ...
- cyclic lists

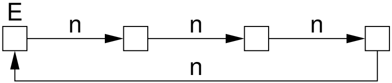


$\models sllc(E)$

Considered Fragment 2/3

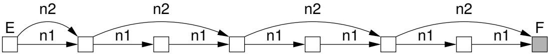
$$P(E, F, \overrightarrow{B}) = (E = F \wedge emp) \vee$$
$$(E \notin \{F\} \cup \overrightarrow{B} \wedge \exists X_{tl} \,.\, \mathbf{\Sigma}(\mathbf{E}, \mathbf{X}_{tl}, \overrightarrow{B}) * P(X_{tl}, F, \ldots))$$

- ...
- cyclic lists

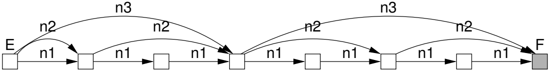


$\models sllc(E)$

- skip lists



$\models skl2(E, F)$

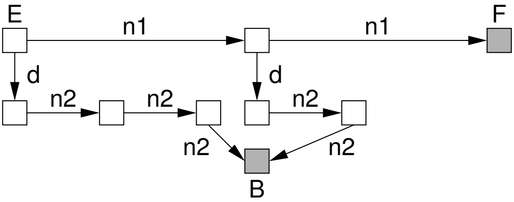


$\models skl3(E, F)$

Considered Fragment 3/3

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

- ...
- and their nested combinations



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

- (**doubly** linked lists)

Overview

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

1. Test entailment of pure parts (is $\Pi_\varphi \Rightarrow \Pi_\psi$ SAT?)

Overview

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

1. Test entailment of pure parts (is $\Pi_\varphi \Rightarrow \Pi_\psi$ SAT?)
2. 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. Test entailment of pure parts (is $\Pi_\varphi \Rightarrow \Pi_\psi$ SAT?)
2. Match every points-to $x \mapsto \{\dots\}$ in Σ_ψ with a points-to in Σ_φ
3. 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

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

Entailment of Pure Parts

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

- Construct Boolean abstractions of φ and ψ:
  - *BoolAbs*$[\varphi]$ encodes the pure part, equality and semantics of $*$

Entailment of Pure Parts

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

- Construct Boolean abstractions of φ and ψ:
  - *BoolAbs*[φ] encodes the pure part, equality and semantics of $*$
  - φ and *BoolAbs*[φ] are equisatisfiable
  - $\varphi \Rightarrow E = F$ iff *BoolAbs*[φ] $\Rightarrow [E = F]$
  - $\varphi \Rightarrow E \neq F$ iff *BoolAbs*[φ] $\Rightarrow \neg[E = F]$

Entailment of Pure Parts

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

- Construct Boolean abstractions of φ and ψ:
  - *BoolAbs*$[\varphi]$ encodes the pure part, equality and semantics of $*$
  - φ and *BoolAbs*$[\varphi]$ are equisatisfiable
  - $\varphi \Rightarrow E = F$ iff *BoolAbs*$[\varphi] \Rightarrow [E = F]$
  - $\varphi \Rightarrow E \neq F$ iff *BoolAbs*$[\varphi] \Rightarrow \neg[E = F]$
- Normalize φ and ψ according to *BoolAbs*$[\cdot] \leadsto \varphi', \psi'$
  - use SAT solver
  - add implied (dis)equalities
  - remove empty inductive predicates
    - e.g. if $E = F \wedge P(E, F)$, remove $P(E, F)$
- Test whether $\Pi_{\varphi'} \Rightarrow \Pi_{\psi'}$ is SAT

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 $\Sigma_{\varphi'}$:

- 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 $\Sigma_{\varphi'}$:

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

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

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

- points-to $E \mapsto \{(f_1, x_1), \dots\}$ in $\Sigma_{\psi'}$:
  - find $E \mapsto \{(f_1, x_1), \dots\}$ in Σ_{φ}
- 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 $\Sigma_{\varphi'}$:

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

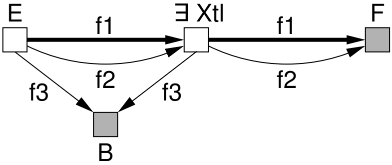
- points-to $E \mapsto \{(f_1, x_1), \ldots\}$ in $\Sigma_{\psi'}$:
  - find $E \mapsto \{(f_1, x_1), \ldots\}$ in Σ_{φ}
- inductive predicates $P(E, F, \overrightarrow{B})$ in $\Sigma_{\psi'}$:
  - in the order from the most specialized to the most general
  - look at $\Sigma_{\varphi'}$ as a graph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

$$\mathcal{T}_G \overset{?}{\in} \mathcal{L}(\mathcal{A}_{P(E,F,\overrightarrow{B})})$$

Entailment of Shape Parts

Transforming Graphs into Trees

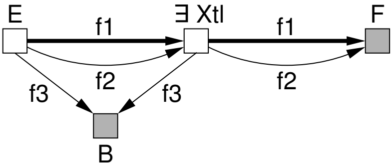
Dealing with joins ($>$ 1 incoming edges):



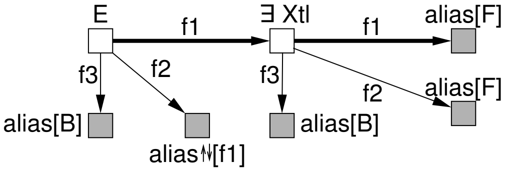
Entailment of Shape Parts

Transforming Graphs into Trees

Dealing with joins ($>$ 1 incoming edges):



- Spanning Tree (*ST*)
- split every join into several copies:
  - one for every incoming edge $\notin ST$
  - label it with $alias[Y]$, $alias{\uparrow}[f_1\ f_2\ \ldots]$, or $alias{\uparrow\downarrow}[f_1\ f_2\ \ldots]$



Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata 1/3

Tree automaton representing all possible unfoldings of P

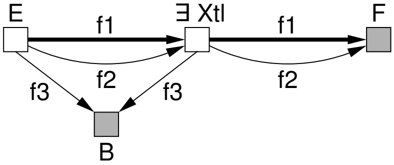
Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata 1/3

Tree automaton representing all possible unfoldings of P

The idea:

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



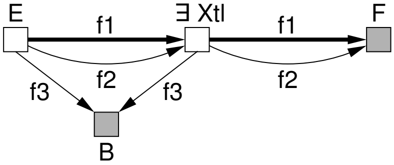
Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata 1/3

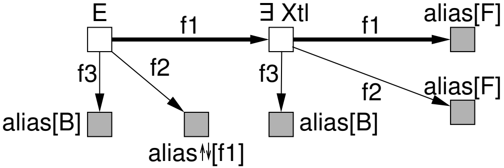
Tree automaton representing all possible unfoldings of P

The idea:

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



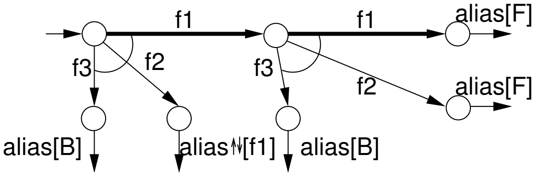
2. Get the tree encoding $\mathcal{T}_{P^{[2]}}$



Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata 2/3

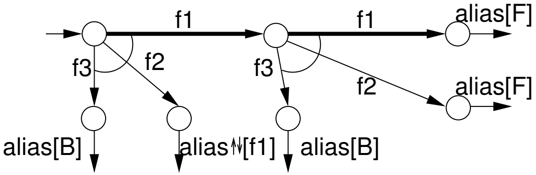
3. Transform $\mathcal{T}_{P[2]}$ into a tree automaton $\mathcal{A}_{P[2]}$ accepting $\{\mathcal{T}_{P[2]}\}$



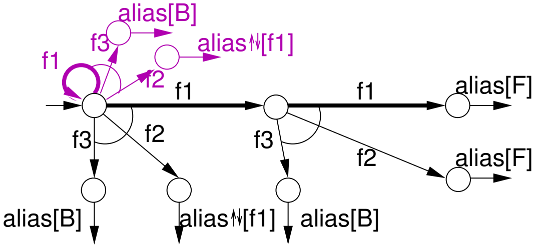
Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata 2/3

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



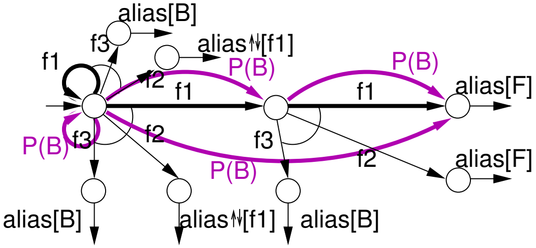
4 Add loop enabling construction of the list backbone of size ≥ 2



Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata 3/3

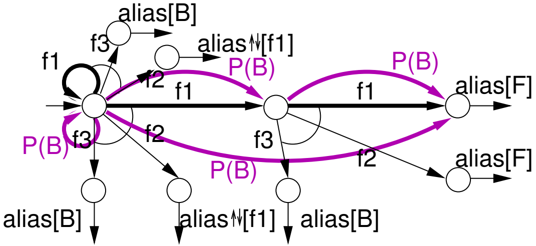
5. Duplicate backbone transitions with transitions over $P \rightsquigarrow \mathcal{A}_{P[2+]}$
   - to enable arbitrary interleaving



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

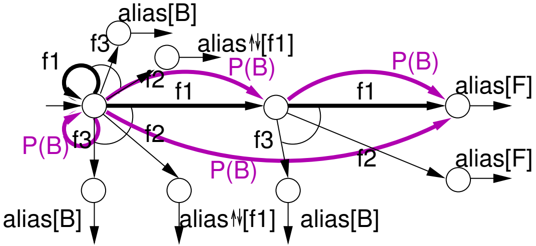


6. For every nested predicate edge over Q, insert $\mathcal{A}_Q$
   - and modify the aliases appearing in $\mathcal{A}_Q$

Entailment of Shape Parts

Transforming Inductive Predicates into Tree Automata 3/3

5. Duplicate backbone transitions with transitions over $P \leadsto \mathcal{A}_{P[2+]}$
   - 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 $\mathcal{A}_{P[1]} \leadsto \mathcal{A}_{P[1+]}$

Soundness, Completeness & Complexity

The decision procedure is

- sound

Soundness, Completeness & Complexity

The decision procedure is

- sound
- polynomial and incomplete
  - issues with empty nested lists

Soundness, Completeness & Complexity

The decision procedure is

- sound
- polynomial and incomplete
  - issues with empty nested lists
- extension: exponential and complete
  - for the considered fragment
  - exponential in the maximum height of the hierarchy of nested predicates

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: $< 8\,\mathrm{s}$ (2^{nd} 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: $0.35\,\mathrm{s}$ (1^{st} place)

Future work

- Generalize to a more expressive fragment of SL
- Integrate into a program analysis framework

Conclusion

- A decision procedure for a fragment of SL
  - practical fragment for lists
- Entailment queries split to simpler ones . . .
  - compositional
- . . . and reduced to tree automata membership
- Encouraging experimental results