Heap Automata

(ESOP 2017)

Christina Jansen[1] Jens Katelaan[2]

Christoph Matheja[1] Thomas Noll[1] Florian Zuleger[2]

[1] RWTH Aachen University
[2] TU Wien

Shonan Meeting 2017, Japan

Robustness of Symbolic Heap Separation Logic

- Symbolic heaps emerged as an idiomatic SL fragment employed by various automated verification tools.

Robustness of Symbolic Heap Separation Logic

- Symbolic heaps emerged as an idiomatic SL fragment employed by various automated verification tools.
- These tools rely on systems of inductive predicate definitions (SID) as data structure specifications.

Robustness of Symbolic Heap Separation Logic

- Symbolic heaps emerged as an idiomatic SL fragment employed by various automated verification tools.
- These tools rely on systems of inductive predicate definitions (SID) as data structure specifications.
- Ongoing trend: Allow user-supplied SIDs instead of handcrafted ones.

Robustness of Symbolic Heap Separation Logic

- Symbolic heaps emerged as an idiomatic SL fragment employed by various automated verification tools.
- These tools rely on systems of inductive predicate definitions (SID) as data structure specifications.
- Ongoing trend: Allow user-supplied SIDs instead of handcrafted ones.
- We consider two problems: Given an SID. . .

Robustness of Symbolic Heap Separation Logic

- Symbolic heaps emerged as an idiomatic SL fragment employed by various automated verification tools.
- These tools rely on systems of inductive predicate definitions (SID) as data structure specifications.
- Ongoing trend: Allow user-supplied SIDs instead of handcrafted ones.
- We consider two problems: Given an SID. . .
  1. prove that it is **robust**.

Robustness of Symbolic Heap Separation Logic

- Symbolic heaps emerged as an idiomatic SL fragment employed by various automated verification tools.
- These tools rely on systems of inductive predicate definitions (SID) as data structure specifications.
- Ongoing trend: Allow user-supplied SIDs instead of handcrafted ones.
- We consider two problems: Given an SID...
  1. prove that it is **robust**. — garbage-free, acyclic, satisfiable,...

Robustness of Symbolic Heap Separation Logic

- Symbolic heaps emerged as an idiomatic SL fragment employed by various automated verification tools.
- These tools rely on systems of inductive predicate definitions (SID) as data structure specifications.
- Ongoing trend: Allow user-supplied SIDs instead of handcrafted ones.
- We consider two problems: Given an SID...
  1. prove that it is **robust**. — garbage-free, acyclic, satisfiable,...
  2. **synthesize** a robust SID from it.

Shape Analysis + Temporal Properties

- Classical shape analysis properties: memory safety, ...

Shape Analysis + Temporal Properties

- Classical shape analysis properties: memory safety, ...
- Temporal properties:

Shape Analysis + Temporal Properties

- Classical shape analysis properties: memory safety, ...
- Temporal properties:
  - Every element is always reachable by either x or y

Shape Analysis + Temporal Properties

- Classical shape analysis properties: memory safety, ...
- Temporal properties:
  - Every element is always reachable by either x or y
  - Every element is eventually processed by procedure Q

Shape Analysis + Temporal Properties

- Classical shape analysis properties: memory safety, ...
- Temporal properties:
  - Every element is always reachable by either x or y
  - Every element is eventually processed by procedure Q
  - Whenever the heap contains garbage, it eventually does not

Shape Analysis + Temporal Properties

- Classical shape analysis properties: memory safety, ...
- Temporal properties:
  - Every element is always reachable by either x or y
  - Every element is eventually processed by procedure Q
  - Whenever the heap contains garbage, it eventually does not
  - All elements belong to a tree until inserted into a list

Shape Analysis + Temporal Properties

- Classical shape analysis properties: memory safety, ...
- Temporal properties:
  - Every element is always reachable by either x or y
  - Every element is eventually processed by procedure Q
  - Whenever the heap contains garbage, it eventually does not
  - All elements belong to a tree until inserted into a list
  - If an element is stored in x it will forever be the root of a tree

Shape Analysis + Temporal Properties

- Classical shape analysis properties: memory safety, ...
- Temporal properties:
  - Every element is always reachable by either x or y
  - Every element is eventually processed by procedure Q
  - Whenever the heap contains garbage, it eventually does not
  - All elements belong to a tree until inserted into a list
  - If an element is stored in x it will forever be the root of a tree
  - In every state the heap is either a tree or a doubly-linked list

Shape Analysis + Temporal Properties

- Classical shape analysis properties: memory safety, ...
- Temporal properties:
  - Every element is always reachable by either x or y
  - Every element is eventually processed by procedure Q
  - Whenever the heap contains garbage, it eventually does not
  - All elements belong to a tree until inserted into a list
  - If an element is stored in x it will forever be the root of a tree
  - In every state the heap is either a tree or a doubly-linked list
  - The successors of every original input element are restored upon termination

Shape Analysis + Model-Checking

- How do we prove temporal properties about symbolic heaps?

Shape Analysis + Model-Checking

- How do we prove temporal properties about symbolic heaps?
  - Generate labeled transition system using shape analysis

Shape Analysis + Model-Checking

- How do we prove temporal properties about symbolic heaps?
  - Generate labeled transition system using shape analysis
  - Every state corresponds to an SL formula φ

Shape Analysis + Model-Checking

- How do we prove temporal properties about symbolic heaps?
  - Generate labeled transition system using shape analysis
  - Every state corresponds to an SL formula φ
  - Apply standard model-checking to transition system

Shape Analysis + Model-Checking

- How do we prove temporal properties about symbolic heaps?
  - Generate labeled transition system using shape analysis
  - Every state corresponds to an SL formula φ
  - Apply standard model-checking to transition system
- Problem: Prove $\varphi \models \mathit{Prop}$

Shape Analysis + Model-Checking

- How do we prove temporal properties about symbolic heaps?
  - Generate labeled transition system using shape analysis
  - Every state corresponds to an SL formula φ
  - Apply standard model-checking to transition system
- Problem: Prove $\varphi \models \mathit{Prop}$ **for a few million formulas φ**

Shape Analysis + Model-Checking

- How do we prove temporal properties about symbolic heaps?
  - Generate labeled transition system using shape analysis
  - Every state corresponds to an SL formula φ
  - Apply standard model-checking to transition system
- Problem: Prove $\varphi \models \mathit{Prop}$ **for a few million formulas** φ
  1. **Synthesize** robust SID w.r.t. *Prop*

Shape Analysis + Model-Checking

- How do we prove temporal properties about symbolic heaps?
  - Generate labeled transition system using shape analysis
  - Every state corresponds to an SL formula φ
  - Apply standard model-checking to transition system
- Problem: Prove $\varphi \models \mathit{Prop}$ **for a few million formulas φ**
  1. **Synthesize** robust SID w.r.t. *Prop*
  2. Run shape analysis space with new SID

Shape Analysis + Model-Checking

- How do we prove temporal properties about symbolic heaps?
  - Generate labeled transition system using shape analysis
  - Every state corresponds to an SL formula φ
  - Apply standard model-checking to transition system
- Problem: Prove $\varphi \models \mathit{Prop}$ **for a few million formulas φ**
  1. **Synthesize** robust SID w.r.t. *Prop*
  2. Run shape analysis space with new SID
  3. Efficiently decide $\varphi \models \mathit{Prop}$ without looking into predicates

Overview of our Results

- We formally capture robustness properties by heap automata

Overview of our Results

- We formally capture robustness properties by heap automata
- We develop an algorithmic framework: For every heap automaton we obtain...

Overview of our Results

- We formally capture robustness properties by heap automata
- We develop an algorithmic framework: For every heap automaton we obtain...
  - a decision procedure for SID robustness

Overview of our Results

- We formally capture robustness properties by heap automata
- We develop an algorithmic framework: For every heap automaton we obtain...
  - a decision procedure for SID robustness
  - a synthesis procedure

Overview of our Results

- We formally capture robustness properties by heap automata
- We develop an algorithmic framework: For every heap automaton we obtain...
  - a decision procedure for SID robustness
  - a synthesis procedure and a complexity bound

Overview of our Results

- We formally capture robustness properties by heap automata
- We develop an algorithmic framework: For every heap automaton we obtain. . .
  - a decision procedure for SID robustness
  - a synthesis procedure and a complexity bound
- Considered robustness properties include acyclicity, garbage-freedom, establishment, reachability, satisfiability. . .

Overview of our Results

- We formally capture robustness properties by heap automata
- We develop an algorithmic framework: For every heap automaton we obtain...
  - a decision procedure for SID robustness
  - a synthesis procedure and a complexity bound
- Considered robustness properties include acyclicity, garbage-freedom, establishment, reachability, satisfiability...
- Implementation and experiments

Overview of our Results

- We formally capture robustness properties by heap automata
- We develop an algorithmic framework: For every heap automaton we obtain...
  - a decision procedure for SID robustness
  - a synthesis procedure and a complexity bound
- Considered robustness properties include acyclicity, garbage-freedom, establishment, reachability, satisfiability...
- Implementation and experiments
  - Standalone tool for SL

Overview of our Results

- We formally capture robustness properties by heap automata
- We develop an algorithmic framework: For every heap automaton we obtain...
  - a decision procedure for SID robustness
  - a synthesis procedure and a complexity bound
- Considered robustness properties include acyclicity, garbage-freedom, establishment, reachability, satisfiability...
- Implementation and experiments
  - Standalone tool for SL
  - Part of model-checking within ATTESTOR

Symbolic Heaps with Inductive Predicates

Terms: $t ::= x \mid \texttt{null}$

Symbolic Heaps with Inductive Predicates

Terms:	$t \ ::= x \mid \texttt{null}$	
Pure formulas:	$\pi \ ::= t = t \mid t \neq t$	($\Pi :$ set of pure formulas)

Symbolic Heaps with Inductive Predicates

Terms:	$t \ ::= x \mid \texttt{null}$	
Pure formulas:	$\pi \ ::= t = t \mid t \neq t$	$(\Pi:$ set of pure formulas$)$
Spatial formulas:	$\Sigma \ ::= \texttt{emp} \mid \qquad \mid$	$(\mathbf{t}:$ tuple of terms$)$

- $\texttt{emp}$ is the empty heap

Symbolic Heaps with Inductive Predicates

Terms:	$t ::= x \mid \texttt{null}$	
Pure formulas:	$\pi ::= t = t \mid t \neq t$	$(\Pi:$ set of pure formulas$)$
Spatial formulas:	$\Sigma ::= \texttt{emp} \mid x \mapsto \mathbf{t} \mid$	$(\mathbf{t}:$ tuple of terms$)$

- $\texttt{emp}$ is the empty heap
- $x \mapsto \mathbf{t}$ is a pointer to a single record

Symbolic Heaps with Inductive Predicates

Terms:	$t \ ::= x \mid \texttt{null}$	
Pure formulas:	$\pi \ ::= t = t \mid t \neq t$	(Π : set of pure formulas)
Spatial formulas:	$\Sigma \ ::= \texttt{emp} \mid x \mapsto \mathbf{t} \mid \Sigma * \Sigma$	($\mathbf{t}$: tuple of terms)

- $\texttt{emp}$ is the empty heap
- $x \mapsto \mathbf{t}$ is a pointer to a single record
- $*$ is the separating conjunction of two domain-disjoint heaps.

Symbolic Heaps with Inductive Predicates

Terms:	$t ::= x \mid \texttt{null}$	
Pure formulas:	$\pi ::= t = t \mid t \neq t$	$(\Pi:$ set of pure formulas$)$
Spatial formulas:	$\Sigma ::= \texttt{emp} \mid x \mapsto \mathbf{t} \mid \Sigma * \Sigma$	$(\mathbf{t}:$ tuple of terms$)$
Predicate calls:	$\Gamma ::= \texttt{emp} \mid P(\mathbf{t}) \mid \Gamma * \Gamma$	$(P:$ predicate symbol$)$

- $\texttt{emp}$ is the empty heap
- $x \mapsto \mathbf{t}$ is a pointer to a single record
- $*$ is the separating conjunction of two domain-disjoint heaps.

Symbolic Heaps with Inductive Predicates

Terms:	$t ::= x \mid \texttt{null}$	
Pure formulas:	$\pi ::= t = t \mid t \neq t$	(Π : set of pure formulas)
Spatial formulas:	$\Sigma ::= \texttt{emp} \mid x \mapsto \mathbf{t} \mid \Sigma * \Sigma$	($\mathbf{t}$: tuple of terms)
Predicate calls:	$\Gamma ::= \texttt{emp} \mid P(\mathbf{t}) \mid \Gamma * \Gamma$	(P : predicate symbol)
Symbolic heaps (SH):	$\varphi(\mathbf{x}) ::= \exists \mathbf{z} \,.\, \Sigma * \Gamma : \Pi$	($\mathbf{x}, \mathbf{z}$: tuples of variables)

- `emp` is the empty heap
- $x \mapsto \mathbf{t}$ is a pointer to a single record
- $*$ is the separating conjunction of two domain-disjoint heaps.

Symbolic Heaps with Inductive Predicates

Terms:	t	$::= x \mid \texttt{null}$
Pure formulas:	π	$::= t = t \mid t \neq t \qquad (\Pi:$ set of pure formulas$)$
Spatial formulas:	Σ	$::= \texttt{emp} \mid x \mapsto \mathbf{t} \mid \Sigma * \Sigma \quad (\mathbf{t}:$ tuple of terms$)$
Predicate calls:	Γ	$::= \texttt{emp} \mid P(\mathbf{t}) \mid \Gamma * \Gamma \quad (P:$ predicate symbol$)$
Symbolic heaps (SH):	$\varphi(\mathbf{x})$	$::= \exists \mathbf{z} \,.\, \Sigma * \Gamma : \Pi \quad (\mathbf{x}, \mathbf{z}:$ tuples of variables$)$
		$\varphi(\mathbf{x})$ is reduced if $\Gamma = \texttt{emp}$

- emp is the empty heap
- $x \mapsto \mathbf{t}$ is a pointer to a single record
- $*$ is the separating conjunction of two domain-disjoint heaps.

Systems of Inductive Definitions (SIDs)

An SID Φ is a finite set of rules of the form

$$\exists \mathbf{z} \,.\, \Sigma \,*\, \Gamma \,:\, \Pi \Rightarrow \; P(\mathbf{x})$$

Systems of Inductive Definitions (SIDs)

An SID Φ is a finite set of rules of the form

$$\exists \mathbf{z} \, . \, \Sigma \, * \, \Gamma \, : \, \Pi \Rightarrow \; P(\mathbf{x})$$

Example (Binary trees)

$$\begin{aligned} \texttt{emp} \, : \, \{x = \texttt{null}\} \; &\Rightarrow \; \textit{tree}(x) \\ \exists y, z \, . \, x \mapsto (y, z) * \textit{tree}(y) * \textit{tree}(z) \; &\Rightarrow \; \textit{tree}(x) \end{aligned}$$

Systems of Inductive Definitions (SIDs)

An SID Φ is a finite set of rules of the form

$$\exists \mathbf{z} \,.\, \Sigma \, * \, \Gamma \, : \, \Pi \Rightarrow \; P(\mathbf{x})$$

Example (Binary trees)

$$\begin{aligned} \mathtt{emp} \, : \, \{x = \mathtt{null}\} \; &\Rightarrow \textit{tree}(x) \\ \exists y, z \;.\; x \mapsto (y, z) * \textit{tree}(y) * \textit{tree}(z) \; &\Rightarrow \textit{tree}(x) \end{aligned}$$

Semantics of predicate calls is given by unfolding to reduced SHs collected in $\textit{unfold}_{\Phi}(P(\mathbf{x}))$.

Robustness Properties

Robustness properties are sets of reduced symbolic heaps (RSH).

Robustness Properties

Robustness properties are sets of reduced symbolic heaps (RSH).

Example

ESTABLISHED: no dangling pointers

Robustness Properties

Robustness properties are sets of reduced symbolic heaps (RSH).

Example

ESTABLISHED: no dangling pointers

SAT: all satisfiable RSHs

Robustness Properties

Robustness properties are sets of reduced symbolic heaps (RSH).

Example

ESTABLISHED: no dangling pointers

SAT: all satisfiable RSHs

GARBAGEFREE: Every location is reachable from a free variable

Robustness Properties: Subtleties

Is y reachable from x in $P(x, y)$?

Robustness Properties: Subtleties

Is y reachable from x in $P(x, y)$?

$$P(x, y) \xRightarrow{\text{unfold}} \exists(z_1, z_2) \,.\quad \Sigma \quad * \; P_1(z_1, z_2) \, * \, P_2(z_2, y) \; : \Pi$$

Robustness Properties: Subtleties

Is y reachable from x in $P(x, y)$?

$$P(x,y) \xRightarrow{\text{unfold}} \exists(z_1, z_2) \,.\, \underbrace{\Sigma}_{x \mapsto z_1} * \underbrace{P_1(z_1, z_2)}_{z_1 = z_2} * \underbrace{P_2(z_2, y)}_{z_2 \mapsto y} \;:\; \Pi$$

Robustness Properties: Subtleties

Is y reachable from x in $P(x, y)$?

$$P(x,y) \xrightarrow{\text{unfold}} \exists(z_1, z_2)\,.\, \underbrace{\Sigma}_{x \mapsto z_1} * \underbrace{P_1(z_1, z_2)}_{z_1 = z_2} * \underbrace{P_2(z_2, y)}_{z_2 \mapsto y} \,:\, \Pi$$

- Reachability might depend on unfoldings of all predicates

Robustness Properties: Subtleties

Is y reachable from x in $P(x, y)$?

$$P(x, y) \xRightarrow{\text{unfold}} \exists(z_1, z_2)\,.\, \underbrace{\Sigma}_{x \mapsto z_1} * \underbrace{P_1(z_1, z_2)}_{z_1 = z_2} * \underbrace{P_2(z_2, y)}_{z_2 \mapsto y} : \Pi$$

- Reachability might depend on unfoldings of all predicates
- How do we know that some other predicate does not invalidate reachability, e.g. $z_1 \neq z_2$?

Robustness Properties: Subtleties

Is y reachable from x in $P(x, y)$?

$$P(x, y) \xRightarrow{\text{unfold}} \exists(z_1, z_2)\,.\, \underbrace{\Sigma}_{x \mapsto z_1} * \underbrace{P_1(z_1, z_2)}_{z_1 = z_2} * \underbrace{P_2(z_2, y)}_{z_2 \mapsto y} : \Pi$$

- Reachability might depend on unfoldings of all predicates
- How do we know that some other predicate does not invalidate reachability, e.g. $z_1 \neq z_2$?
- How do we prove reachability for

Robustness Properties: Subtleties

Is y reachable from x in $P(x, y)$?

$$P(x,y) \xrightarrow{\text{unfold}} \exists(z_1, z_2)\,.\, \underbrace{\Sigma}_{x \mapsto z_1} * \underbrace{P_1(z_1, z_2)}_{z_1 = z_2} * \underbrace{P_2(z_2, y)}_{z_2 \mapsto y} : \Pi$$

- Reachability might depend on unfoldings of all predicates
- How do we know that some other predicate does not invalidate reachability, e.g. $z_1 \neq z_2$?
- How do we prove reachability for

  all unfoldings

Robustness Properties: Subtleties

Is y reachable from x in $P(x, y)$?

$$P(x,y) \xRightarrow{\text{unfold}} \exists(z_1, z_2)\,.\ \underbrace{\Sigma}_{x \mapsto z_1} * \underbrace{P_1(z_1, z_2)}_{z_1 = z_2} * \underbrace{P_2(z_2, y)}_{z_2 \mapsto y}\ :\ \Pi$$

- Reachability might depend on unfoldings of all predicates
- How do we know that some other predicate does not invalidate reachability, e.g. $z_1 \neq z_2$?
- How do we prove reachability for

  all unfoldings

  of **arbitrary symbolic heaps**

Robustness Properties: Subtleties

Is y reachable from x in $P(x, y)$?

$$P(x,y) \xrightarrow{\text{unfold}} \exists(z_1, z_2) \, . \, \underbrace{\Sigma}_{x \mapsto z_1} * \underbrace{P_1(z_1, z_2)}_{z_1 = z_2} * \underbrace{P_2(z_2, y)}_{z_2 \mapsto y} \; : \; \Pi$$

- Reachability might depend on unfoldings of all predicates
- How do we know that some other predicate does not invalidate reachability, e.g. $z_1 \neq z_2$?
- How do we prove reachability for

  all unfoldings

  of **arbitrary symbolic heaps**

  in **arbitrary SIDs**?

Heap Automata: Compositionality

We reason compositionally while unfolding a symbolic heap

$$\varphi(\mathbf{x}) \;=\; \exists \mathbf{z} \,.\, \Sigma * \;\; P_1(\mathbf{x}_1) \;\; * \ldots * \;\; P_m(\mathbf{x}_m) \;\; : \Pi$$

Heap Automata: Compositionality

We reason compositionally while unfolding a symbolic heap

$$\varphi(\mathbf{x}) \; = \; \exists \mathbf{z} \, . \, \Sigma * \overbrace{P_1(\mathbf{x}_1)}^{\text{property } q_1} * \ldots * \overbrace{P_m(\mathbf{x}_m)}^{\text{property } q_m} : \Pi$$

Soundness: If P_k has an unfolding with property q_k $(1 \leq k \leq m)$

Heap Automata: Compositionality

We reason compositionally while unfolding a symbolic heap

$$\varphi(\mathbf{x}) \;=\; \underbrace{\exists \mathbf{z} \,.\, \Sigma * \overbrace{P_1(\mathbf{x}_1)}^{\text{property } q_1} * \ldots * \overbrace{P_m(\mathbf{x}_m)}^{\text{property } q_m} : \Pi}_{\text{property } q}$$

Soundness: If P_k has an unfolding with property q_k $(1 \leq k \leq m)$ and for those unfoldings $\varphi(\mathbf{x})$ has property q

Heap Automata: Compositionality

We reason compositionally while unfolding a symbolic heap

$$\varphi(\mathbf{x}) \;=\; \underbrace{\exists \mathbf{z}\,.\,\Sigma * \overbrace{P_1(\mathbf{x}_1)}^{\text{property } q_1} * \ldots * \overbrace{P_m(\mathbf{x}_m)}^{\text{property } q_m} : \Pi}_{\text{property } q}$$

Soundness: If P_k has an unfolding with property q_k $(1 \leq k \leq m)$ and for those unfoldings $\varphi(\mathbf{x})$ has property q then $\varphi(\mathbf{x})$ has an unfolding with property q.

Heap Automata: Compositionality

We reason compositionally while unfolding a symbolic heap

$$\varphi(\mathbf{x}) \;=\; \underbrace{\exists \mathbf{z} \,.\, \Sigma * \overbrace{P_1(\mathbf{x}_1)}^{\text{property } q_1} * \ldots * \overbrace{P_m(\mathbf{x}_m)}^{\text{property } q_m} : \Pi}_{\text{property } q}$$

Soundness: If P_k has an unfolding with property q_k $(1 \leq k \leq m)$ and for those unfoldings $\varphi(\mathbf{x})$ has property q then $\varphi(\mathbf{x})$ has an unfolding with property q.

Completeness: If $\varphi(\mathbf{x})$ has an unfolding with property q

Heap Automata: Compositionality

We reason compositionally while unfolding a symbolic heap

$$\varphi(\mathbf{x}) \;=\; \underbrace{\exists \mathbf{z}\,.\,\Sigma * \overbrace{P_1(\mathbf{x}_1)}^{\text{property } q_1} * \ldots * \overbrace{P_m(\mathbf{x}_m)}^{\text{property } q_m} : \Pi}_{\text{property } q}$$

Soundness: If P_k has an unfolding with property q_k $(1 \leq k \leq m)$ and for those unfoldings $\varphi(\mathbf{x})$ has property q then $\varphi(\mathbf{x})$ has an unfolding with property q.

Completeness: If $\varphi(\mathbf{x})$ has an unfolding with property q then there are unfoldings of P_k with some property q_k

Heap Automata: Compositionality

We reason compositionally while unfolding a symbolic heap

$$\varphi(\mathbf{x}) \;=\; \underbrace{\exists \mathbf{z} \,.\, \Sigma * \overbrace{P_1(\mathbf{x}_1)}^{\text{property } q_1} * \ldots * \overbrace{P_m(\mathbf{x}_m)}^{\text{property } q_m} : \Pi}_{\text{property } q}$$

Soundness: If P_k has an unfolding with property q_k $(1 \leq k \leq m)$ and for those unfoldings $\varphi(\mathbf{x})$ has property q then $\varphi(\mathbf{x})$ has an unfolding with property q.

Completeness: If $\varphi(\mathbf{x})$ has an unfolding with property q then there are unfoldings of P_k with some property q_k and for those unfoldings $\varphi(\mathbf{x})$ has property q.

Heap Automata: Definition

Definition

A heap automaton is a tuple $\mathcal{A} = (Q, \rightarrow, F)$, where

Heap Automata: Definition

Definition

A heap automaton is a tuple $\mathcal{A} = (Q, \rightarrow, F)$, where

- Q is a finite set of states,

Heap Automata: Definition

Definition

A heap automaton is a tuple $\mathcal{A} = (Q, \rightarrow, F)$, where

- Q is a finite set of states,
- $F \subseteq Q$ is a set of final states, and

Heap Automata: Definition

Definition

A heap automaton is a tuple $\mathcal{A} = (Q, \rightarrow, F)$, where

- Q is a finite set of states,
- $F \subseteq Q$ is a set of final states, and
- $\rightarrow \subseteq Q^* \times \mathit{SH} \times Q$ is a transition relation such that

Heap Automata: Definition

Definition

A heap automaton is a tuple $\mathcal{A} = (Q, \rightarrow, F)$, where

- Q is a finite set of states,
- $F \subseteq Q$ is a set of final states, and
- $\rightarrow \subseteq Q^* \times \mathit{SH} \times Q$ is a transition relation such that
  - $\rightarrow$ is compositional (prev. slide), and

Heap Automata: Definition

Definition

A heap automaton is a tuple $\mathcal{A} = (Q, \rightarrow, F)$, where

- Q is a finite set of states,
- $F \subseteq Q$ is a set of final states, and
- $\rightarrow \subseteq Q^* \times \mathit{SH} \times Q$ is a transition relation such that
    - $\rightarrow$ is compositional (prev. slide), and
    - $\rightarrow$ is decidable.

Heap Automata: Definition

Definition

A heap automaton is a tuple $\mathcal{A} = (Q, \rightarrow, F)$, where

- Q is a finite set of states,
- $F \subseteq Q$ is a set of final states, and
- $\rightarrow \subseteq Q^* \times \mathit{SH} \times Q$ is a transition relation such that
  - $\rightarrow$ is compositional (prev. slide), and
  - $\rightarrow$ is decidable.

The language $L(\mathcal{A})$ of heap automaton $\mathcal{A}$ is the set of all reduced symbolic heaps with a transition to a final state.

Heap Automata: Results

Given SID Φ,

Heap Automata: Results

Given SID Φ, heap automaton $\mathcal{A}$, and

Heap Automata: Results

Given SID Φ, heap automaton $\mathcal{A}$, and symbolic heap $\varphi(\mathbf{x})$...

Heap Automata: Results

Given SID Φ, heap automaton $\mathcal{A}$, and symbolic heap $\varphi(\mathbf{x})$...

Theorem (Refinement Theorem)

One can effectively construct an SID Ψ such that

$$\forall P \;:\; \mathit{unfold}_{\Psi}(P(\mathbf{x})) = \mathit{unfold}_{\Phi}(P(\mathbf{x})) \cap L(\mathcal{A}).$$

Heap Automata: Results

Given SID Φ, heap automaton $\mathcal{A}$, and symbolic heap $\varphi(\mathbf{x})$...

Theorem (Refinement Theorem)

One can effectively construct an SID Ψ such that

$$\forall P \;:\; \mathit{unfold}_{\Psi}(P(\mathbf{x})) = \mathit{unfold}_{\Phi}(P(\mathbf{x})) \cap L(\mathcal{A}).$$

Theorem

1. $\mathit{size}(\Psi) \;\leq\; \mathit{size}(\Phi) \cdot \mathit{size}(\mathcal{A})^{\#\text{pred. calls}}$

Heap Automata: Results

Given SID Φ, heap automaton $\mathcal{A}$, and symbolic heap $\varphi(\mathbf{x})$...

Theorem (Refinement Theorem)

One can effectively construct an SID Ψ such that

$$\forall P \ : \ \mathit{unfold}_{\Psi}(P(\mathbf{x})) = \mathit{unfold}_{\Phi}(P(\mathbf{x})) \cap L(\mathcal{A}).$$

Theorem

1. $\mathit{size}(\Psi) \ \leq \ \mathit{size}(\Phi) \cdot \mathit{size}(\mathcal{A})^{\#\mathit{pred.\ calls}}$
2. *It is decidable in linear time whether $\mathit{unfold}_{\Phi}(\varphi(\mathbf{x}))$ is empty.*

Heap Automata: Results

Given SID Φ, heap automaton $\mathcal{A}$, and symbolic heap $\varphi(\mathbf{x})\ldots$

Theorem (Refinement Theorem)

One can effectively construct an SID Ψ such that

$$\forall P \ : \ \mathit{unfold}_{\Psi}(P(\mathbf{x})) = \mathit{unfold}_{\Phi}(P(\mathbf{x})) \cap L(\mathcal{A}).$$

Theorem

1. $\mathit{size}(\Psi) \ \leq \ \mathit{size}(\Phi) \cdot \mathit{size}(\mathcal{A})^{\#\text{pred. calls}}$
2. *It is decidable in linear time whether $\mathit{unfold}_{\Phi}(\varphi(\mathbf{x}))$ is empty.*
3. *Languages of heap automata are effectively closed under union, intersection and complement.*

Heap Automata: Results

Given SID Φ, heap automaton $\mathcal{A}$, and symbolic heap $\varphi(\mathbf{x})\ldots$

Theorem (Refinement Theorem)

One can effectively construct an SID Ψ such that

$$\forall P \;:\; \textit{unfold}_{\Psi}(P(\mathbf{x})) = \textit{unfold}_{\Phi}(P(\mathbf{x})) \cap L(\mathcal{A}).$$

Theorem

1. $\textit{size}(\Psi) \;\leq\; \textit{size}(\Phi) \cdot \textit{size}(\mathcal{A})^{\#\textit{pred. calls}}$
2. *It is decidable in linear time whether $\textit{unfold}_{\Phi}(\varphi(\mathbf{x}))$ is empty.*
3. *Languages of heap automata are effectively closed under union, intersection and complement.*
4. *It is decidable whether $\textit{unfold}_{\Phi}(\varphi(\mathbf{x})) \cap L(\mathcal{A}) \neq \emptyset$.*

Heap Automata: Results

Given SID Φ, heap automaton $\mathcal{A}$, and symbolic heap $\varphi(\mathbf{x})$...

Theorem (Refinement Theorem)

One can effectively construct an SID Ψ such that

$$\forall P \;:\; \mathit{unfold}_{\Psi}(P(\mathbf{x})) = \mathit{unfold}_{\Phi}(P(\mathbf{x})) \cap L(\mathcal{A}).$$

Theorem

1. $\mathit{size}(\Psi) \;\leq\; \mathit{size}(\Phi) \cdot \mathit{size}(\mathcal{A})^{\#\mathit{pred.\ calls}}$
2. *It is decidable in linear time whether $\mathit{unfold}_{\Phi}(\varphi(\mathbf{x}))$ is empty.*
3. *Languages of heap automata are effectively closed under union, intersection and complement.*
4. *It is decidable whether $\mathit{unfold}_{\Phi}(\varphi(\mathbf{x})) \cap L(\mathcal{A}) \neq \emptyset$.*
5. *It is decidable whether $\mathit{unfold}_{\Phi}(\varphi(\mathbf{x})) \subseteq L(\mathcal{A})$.*

A Zoo of Robustness Properties

We constructed heap automata for the following properties:

Property

A Zoo of Robustness Properties

We constructed heap automata for the following properties:

Property
satisfiability

A Zoo of Robustness Properties

We constructed heap automata for the following properties:

Property
satisfiability
model-checking

A Zoo of Robustness Properties

We constructed heap automata for the following properties:

Property
satisfiability
model-checking
garbage-freedom

A Zoo of Robustness Properties

We constructed heap automata for the following properties:

Property
satisfiability
model-checking
garbage-freedom
acyclicity

A Zoo of Robustness Properties

We constructed heap automata for the following properties:

Property
satisfiability
model-checking
garbage-freedom
acyclicity
reachability

A Zoo of Robustness Properties

We constructed heap automata for the following properties:

Property
satisfiability
model-checking
garbage-freedom
acyclicity
reachability
establishment

A Zoo of Robustness Properties

We constructed heap automata for the following properties:

Property	Complexity
satisfiability	ExpTime-C[1]
model-checking	ExpTime-C[1]
garbage-freedom	
acyclicity	
reachability	
establishment	

[1] (Brotherston et al., 2014) and (Brotherston et al., 2016)

A Zoo of Robustness Properties

We constructed heap automata for the following properties:

Property	Complexity
satisfiability	ExpTime-C[1]
model-checking	ExpTime-C[1]
garbage-freedom	ExpTime-C
acyclicity	ExpTime-C
reachability	ExpTime-C
establishment	ExpTime-C

[1] (Brotherston et al., 2014) and (Brotherston et al., 2016)

A Zoo of Robustness Properties

We constructed heap automata for the following properties:

Property	Complexity	FV bounded
satisfiability	ExpTime-C[1]	
model-checking	ExpTime-C[1]	
garbage-freedom	ExpTime-C	
acyclicity	ExpTime-C	
reachability	ExpTime-C	
establishment	ExpTime-C	

[1] (Brotherston et al., 2014) and (Brotherston et al., 2016)

A Zoo of Robustness Properties

We constructed heap automata for the following properties:

Property	Complexity	FV bounded
satisfiability	ExpTime-C[1]	NP-C[1]
model-checking	ExpTime-C[1]	NP-C[1]
garbage-freedom	ExpTime-C	
acyclicity	ExpTime-C	
reachability	ExpTime-C	
establishment	ExpTime-C	

[1] (Brotherston et al., 2014) and (Brotherston et al., 2016)

A Zoo of Robustness Properties

We constructed heap automata for the following properties:

Property	Complexity	FV bounded
satisfiability	ExpTime-C[1]	NP-C[1]
model-checking	ExpTime-C[1]	NP-C[1]
garbage-freedom	ExpTime-C	coNP-C
acyclicity	ExpTime-C	coNP-C
reachability	ExpTime-C	coNP-C
establishment	ExpTime-C	coNP-C

[1] (Brotherston et al., 2014) and (Brotherston et al., 2016)

A Zoo of Robustness Properties

We constructed heap automata for the following properties:

Property	Complexity	FV bounded
satisfiability	ExpTime-C[1]	NP-C[1]
model-checking	ExpTime-C[1]	NP-C[1]
garbage-freedom	ExpTime-C	coNP-C
acyclicity	ExpTime-C	coNP-C
reachability	ExpTime-C	coNP-C
establishment	ExpTime-C	coNP-C

[1] (Brotherston et al., 2014) and (Brotherston et al., 2016)

All of these problems are PTime–complete for an additionally bounded number of predicate calls.

Implementation: HARRSH[1]

[1]Heap Automata for Reasoning about Robustness of Symbolic Heaps

Implementation: Harrsh[1]

- Implemented framework and heap automata in Scala

[1] Heap Automata for Reasoning about Robustness of Symbolic Heaps

Implementation: Harrsh[1]

- Implemented framework and heap automata in Scala
- No other tool supports checking robustness properties

[1] Heap Automata for Reasoning about Robustness of Symbolic Heaps

Implementation: HARRSH[1]

- Implemented framework and heap automata in Scala
- No other tool supports checking robustness properties
- Notable exception: CYCLIST can check satisfiability

[1] Heap Automata for Reasoning about Robustness of Symbolic Heaps

Implementation: Harrsh[1]

- Implemented framework and heap automata in Scala
- No other tool supports checking robustness properties
- Notable exception: Cyclist can check satisfiability
- Benchmarks are taken from Cyclist

[1] Heap Automata for Reasoning about Robustness of Symbolic Heaps

Implementation: Harrsh[1]

- Implemented framework and heap automata in Scala
- No other tool supports checking robustness properties
- Notable exception: Cyclist can check satisfiability
- Benchmarks are taken from Cyclist
- 8 common SIDs from the literature: 0.3s to check all robustness properties.

[1] Heap Automata for Reasoning about Robustness of Symbolic Heaps

Implementation: HARRSH[1]

- Implemented framework and heap automata in Scala
- No other tool supports checking robustness properties
- Notable exception: CYCLIST can check satisfiability
- Benchmarks are taken from CYCLIST
- 8 common SIDs from the literature: 0.3s to check all robustness properties.
- 45945 SIDs generated by CABER from C source code

[1]Heap Automata for Reasoning about Robustness of Symbolic Heaps

Implementation: HARRSH[1]

- Implemented framework and heap automata in Scala
- No other tool supports checking robustness properties
- Notable exception: CYCLIST can check satisfiability
- Benchmarks are taken from CYCLIST
- 8 common SIDs from the literature: 0.3s to check all robustness properties.
- 45945 SIDs generated by CABER from C source code
  - Satisfiability: HARRSH: 12.5s   CYCLIST: 44.9s

[1] Heap Automata for Reasoning about Robustness of Symbolic Heaps

Implementation: Harrsh[1]

- Implemented framework and heap automata in Scala
- No other tool supports checking robustness properties
- Notable exception: Cyclist can check satisfiability
- Benchmarks are taken from Cyclist
- 8 common SIDs from the literature: 0.3s to check all robustness properties.
- 45945 SIDs generated by Caber from C source code
  - Satisfiability: Harrsh: 12.5s Cyclist: 44.9s
  - Other robustness properties: ranging from 7.2s to 18.5s

[1] Heap Automata for Reasoning about Robustness of Symbolic Heaps

Implementation: HARRSH[1]

- Implemented framework and heap automata in Scala
- No other tool supports checking robustness properties
- Notable exception: CYCLIST can check satisfiability
- Benchmarks are taken from CYCLIST
- 8 common SIDs from the literature: 0.3s to check all robustness properties.
- 45945 SIDs generated by CABER from C source code
  - Satisfiability: HARRSH: 12.5s CYCLIST: 44.9s
  - Other robustness properties: ranging from 7.2s to 18.5s
- Satisfiability on worst-case instance
  HARRSH: 169s CYCLIST: 164s

[1] Heap Automata for Reasoning about Robustness of Symbolic Heaps

Application to Model-Checking

- Implemented framework and heap automata in Attestor

Application to Model-Checking

- Implemented framework and heap automata in ATTESTOR
- Supported heap automata in LTL formulas:

Application to Model-Checking

- Implemented framework and heap automata in Attestor
- Supported heap automata in LTL formulas:
  - Reachability

Application to Model-Checking

- Implemented framework and heap automata in Attestor
- Supported heap automata in LTL formulas:
  - Reachability
  - Acyclicity

Application to Model-Checking

- Implemented framework and heap automata in Attestor
- Supported heap automata in LTL formulas:
  - Reachability
  - Acyclicity
  - Garbage-Freedom

Application to Model-Checking

- Implemented framework and heap automata in Attestor
- Supported heap automata in LTL formulas:
  - Reachability
  - Acyclicity
  - Garbage-Freedom
  - Reachability

Application to Model-Checking

- Implemented framework and heap automata in ATTESTOR
- Supported heap automata in LTL formulas:
  - Reachability
  - Acyclicity
  - Garbage-Freedom
  - Reachability
  - Shape: The heap is a tree, sll, dll...

Application to Model-Checking

- Implemented framework and heap automata in ATTESTOR
- Supported heap automata in LTL formulas:
  - Reachability
  - Acyclicity
  - Garbage-Freedom
  - Reachability
  - Shape: The heap is a tree, sll, dll...
  - Completeness: Every element of the initial heap has been accessed (by a given variable)

Application to Model-Checking

- Implemented framework and heap automata in ATTESTOR
- Supported heap automata in LTL formulas:
  - Reachability
  - Acyclicity
  - Garbage-Freedom
  - Reachability
  - Shape: The heap is a tree, sll, dll...
  - Completeness: Every element of the initial heap has been accessed (by a given variable)
  - Preservation: The successors of each element are as in the initial heap

A few Experiments

- 2.9GHz Intel Core i5 Laptop, JVM limited to 2GB of RAM
- State space generation (SSG): null pointer dereferences, memory leaks

Program	Property	SSG (s)	Model-Checking (s)
`SLL.reversal`	reachability	0.12	0.02
`SLL.reversal`	completeness	0.13	0.02
`DLL.traversal`	completeness	0.24	0.10
`DLL.traversal`	preservation	0.33	0.32
`DLL.reversal`	shape	0.14	0.05
`DLL.reversal`	reachability	0.18	0.02
`DLL.reversal`	completeness	0.24	0.15
`BT.lindstrom`	term. at root	0.19	0.03
`BT.lindstrom`	shape	0.20	0.17
`BT.lindstrom`	completeness	0.62	0.46
`BT.lindstrom`	preservation	0.38	0.70

What else?

Heap automata…

- …can generate counterexamples for robustness properties

What else?

Heap automata...

- ...can generate counterexamples for robustness properties
- ...can be applied to discharge certain entailments

What else?

Heap automata…

- …can generate counterexamples for robustness properties
- …can be applied to discharge certain entailments
  - restricted to SHs φ, ψ and SID Φ without dangling pointers

What else?

Heap automata...

- ...can generate counterexamples for robustness properties
- ...can be applied to discharge certain entailments
  - restricted to SHs φ, ψ and SID Φ without dangling pointers
  - given heap automata for all predicates in Φ, it is decidable whether $\varphi \models_{\Phi} \psi$.

What else?

Heap automata...

- ...can generate counterexamples for robustness properties
- ...can be applied to discharge certain entailments
  - restricted to SHs φ, ψ and SID Φ without dangling pointers
  - given heap automata for all predicates in Φ, it is decidable whether $\varphi \models_{\Phi} \psi$.
  - enables systematic approach to construct entailment checkers

What else?

Heap automata...

- ...can generate counterexamples for robustness properties
- ...can be applied to discharge certain entailments
  - restricted to SHs φ, ψ and SID Φ without dangling pointers
  - given heap automata for all predicates in Φ, it is decidable whether $\varphi \models_{\Phi} \psi$.
  - enables systematic approach to construct entailment checkers
  - entailments are decidable in $\textsc{ExpTime}$ if heap automata are at most exponentially large.

Summary

- Algorithmic framework for deciding and synthesizing robustness properties based on heap automata

Summary

- Algorithmic framework for deciding and synthesizing robustness properties based on heap automata
- Complexity analysis for common robustness properties

Summary

- Algorithmic framework for deciding and synthesizing robustness properties based on heap automata
- Complexity analysis for common robustness properties
- Robustness checker: https://bitbucket.org/jkatelaan/harrsh

Summary

- Algorithmic framework for deciding and synthesizing robustness properties based on heap automata
- Complexity analysis for common robustness properties
- Robustness checker: `https://bitbucket.org/jkatelaan/harrsh`
- Shape analysis + model-checking: `https://moves-rwth.github.io/attestor`

Summary

- Algorithmic framework for deciding and synthesizing robustness properties based on heap automata
- Complexity analysis for common robustness properties
- Robustness checker:
  `https://bitbucket.org/jkatelaan/harrsh`
- Shape analysis + model-checking:
  `https://moves-rwth.github.io/attestor`

Future Work

Summary

- Algorithmic framework for deciding and synthesizing robustness properties based on heap automata
- Complexity analysis for common robustness properties
- Robustness checker:
  `https://bitbucket.org/jkatelaan/harrsh`
- Shape analysis + model-checking:
  `https://moves-rwth.github.io/attestor`

Future Work

- More robustness properties

Summary

- Algorithmic framework for deciding and synthesizing robustness properties based on heap automata
- Complexity analysis for common robustness properties
- Robustness checker: `https://bitbucket.org/jkatelaan/harrsh`
- Shape analysis + model-checking: `https://moves-rwth.github.io/attestor`

Future Work

- More robustness properties
- More experiments

Summary

- Algorithmic framework for deciding and synthesizing robustness properties based on heap automata
- Complexity analysis for common robustness properties
- Robustness checker: `https://bitbucket.org/jkatelaan/harrsh`
- Shape analysis + model-checking: `https://moves-rwth.github.io/attestor`

Future Work

- More robustness properties
- More experiments
- Characterization of data structures that can be specified by heap automata

Summary

- Algorithmic framework for deciding and synthesizing robustness properties based on heap automata
- Complexity analysis for common robustness properties
- Robustness checker:
  `https://bitbucket.org/jkatelaan/harrsh`
- Shape analysis + model-checking:
  `https://moves-rwth.github.io/attestor`

Future Work

- More robustness properties
- More experiments
- Characterization of data structures that can be specified by heap automata
  - Synthesize heap automata from backward-confluent SIDs?

Backup Slides

Heap Automata: Formal Definition of Compositionality

$\varphi[P/\tau]$ unfolds P by τ

Heap Automata: Formal Definition of Compositionality

$\varphi[P/\tau]$ unfolds P by τ

Definition

A heap automaton $\mathfrak{A} = (Q, \mathit{SH}_{\mathcal{C}}, \rightarrow, F)$ is compositional if

Heap Automata: Formal Definition of Compositionality

$\varphi[P/\tau]$ unfolds P by τ

Definition

A heap automaton $\mathfrak{A} = (Q, \mathit{SH}_{\mathcal{C}}, \rightarrow, F)$ is compositional if for every $p \in Q$ and every $\varphi \in \mathit{SH}_{\mathcal{C}}$ with predicate calls $P_1, \ldots, P_m$ and all reduced symbolic heaps $\tau_1, \ldots, \tau_m \in \mathit{RSH}_{\mathcal{C}}$:

Heap Automata: Formal Definition of Compositionality

$\varphi[P/\tau]$ unfolds P by τ

Definition

A heap automaton $\mathfrak{A} = (Q, \mathit{SH}_{\mathcal{C}}, \rightarrow, F)$ is compositional if for every $p \in Q$ and every $\varphi \in \mathit{SH}_{\mathcal{C}}$ with predicate calls $P_1, \ldots, P_m$ and all reduced symbolic heaps $\tau_1, \ldots, \tau_m \in \mathit{RSH}_{\mathcal{C}}$:

$$\exists \mathbf{q} \in Q^m \,.\, \mathbf{q} \xrightarrow{\varphi} p$$

Heap Automata: Formal Definition of Compositionality

$\varphi[P/\tau]$ unfolds P by τ

Definition

A heap automaton $\mathfrak{A} = (Q, \mathit{SH}_{\mathcal{C}}, \rightarrow, F)$ is compositional if for every $p \in Q$ and every $\varphi \in \mathit{SH}_{\mathcal{C}}$ with predicate calls $P_1, \ldots, P_m$ and all reduced symbolic heaps $\tau_1, \ldots, \tau_m \in \mathit{RSH}_{\mathcal{C}}$:

$$\exists \mathbf{q} \in Q^m \;.\; \mathbf{q} \xrightarrow{\varphi} p \quad \text{and} \quad \bigwedge_{1 \leq i \leq m} \varepsilon \xrightarrow{\tau_i} \mathbf{q}[i]$$

Heap Automata: Formal Definition of Compositionality

$\varphi[P/\tau]$ unfolds P by τ

Definition

A heap automaton $\mathfrak{A} = (Q, \mathit{SH}_{\mathcal{C}}, \rightarrow, F)$ is compositional if for every $p \in Q$ and every $\varphi \in \mathit{SH}_{\mathcal{C}}$ with predicate calls $P_1, \ldots, P_m$ and all reduced symbolic heaps $\tau_1, \ldots, \tau_m \in \mathit{RSH}_{\mathcal{C}}$:

$$\exists \mathbf{q} \in Q^m \,.\, \mathbf{q} \xrightarrow{\varphi} p \quad \text{and} \quad \bigwedge_{1 \leq i \leq m} \varepsilon \xrightarrow{\tau_i} \mathbf{q}[i]$$

if and only if

Heap Automata: Formal Definition of Compositionality

$\varphi[P/\tau]$ unfolds P by τ

Definition

A heap automaton $\mathfrak{A} = (Q, \mathit{SH}_{\mathcal{C}}, \rightarrow, F)$ is compositional if for every $p \in Q$ and every $\varphi \in \mathit{SH}_{\mathcal{C}}$ with predicate calls $P_1, \ldots, P_m$ and all reduced symbolic heaps $\tau_1, \ldots, \tau_m \in \mathit{RSH}_{\mathcal{C}}$:

$$\exists \mathbf{q} \in Q^m \,.\, \mathbf{q} \xrightarrow{\varphi} p \quad \text{and} \quad \bigwedge_{1 \leq i \leq m} \varepsilon \xrightarrow{\tau_i} \mathbf{q}[i]$$

if and only if

$$\varepsilon \xrightarrow{\varphi[P_1/\tau_1, \ldots, P_m/\tau_m]} p$$

Heap Automata: Formal Definition of Compositionality

$\varphi[P/\tau]$ unfolds P by τ

Definition

A heap automaton $\mathfrak{A} = (Q, \mathit{SH}_{\mathcal{C}}, \rightarrow, F)$ is compositional if for every $p \in Q$ and every $\varphi \in \mathit{SH}_{\mathcal{C}}$ with predicate calls $P_1, \ldots, P_m$ and all reduced symbolic heaps $\tau_1, \ldots, \tau_m \in \mathit{RSH}_{\mathcal{C}}$:

$$\exists \mathbf{q} \in Q^m \,.\, \mathbf{q} \xrightarrow{\varphi} p \quad \text{and} \quad \bigwedge_{1 \leq i \leq m} \varepsilon \xrightarrow{\tau_i} \mathbf{q}[i]$$

if and only if

$$\varepsilon \xrightarrow{\varphi[P_1/\tau_1, \ldots, P_m/\tau_m]} p$$

$$L(\mathfrak{A}) \triangleq \{\tau \in \mathit{RSH}_{\mathcal{C}} \mid \exists p \in F \,.\, \varepsilon \xrightarrow{\tau} p\}$$

The Entailment Problem

Definition (Entailment Problem)

Given an SID Φ and symbolic heaps φ, ψ, decide whether

$$\varphi \models_\Phi \psi \;\Leftrightarrow\; \forall s, h \;.\; s, h \models_\Phi \varphi \text{ implies } s, h \models_\Phi \psi$$

The Entailment Problem

Definition (Entailment Problem)

Given an SID Φ and symbolic heaps φ, ψ, decide whether

$$\varphi \models_\Phi \psi \;\Leftrightarrow\; \forall s, h \,.\, s, h \models_\Phi \varphi \text{ implies } s, h \models_\Phi \psi$$

- Crucial for automated program verification based on separation logic

The Entailment Problem

Definition (Entailment Problem)

Given an SID Φ and symbolic heaps φ, ψ, decide whether

$$\varphi \models_\Phi \psi \;\Leftrightarrow\; \forall s, h \;.\; s, h \models_\Phi \varphi \text{ implies } s, h \models_\Phi \psi$$

- Crucial for automated program verification based on separation logic
- Antonopolous et al.: The entailment problem is undecidable

The Entailment Problem

Definition (Entailment Problem)

Given an SID Φ and symbolic heaps φ, ψ, decide whether

$$\varphi \models_\Phi \psi \;\Leftrightarrow\; \forall s, h \,.\, s, h \models_\Phi \varphi \text{ implies } s, h \models_\Phi \psi$$

- Crucial for automated program verification based on separation logic
- Antonopolous et al.: The entailment problem is undecidable
- Most tools use highly-sepcialized techniques for fixed SIDs

The Entailment Problem

Definition (Entailment Problem)

Given an SID Φ and symbolic heaps φ, ψ, decide whether

$$\varphi \models_\Phi \psi \;\Leftrightarrow\; \forall s,h \,.\, s,h \models_\Phi \varphi \text{ implies } s,h \models_\Phi \psi$$

- Crucial for automated program verification based on separation logic
- Antonopolous et al.: The entailment problem is undecidable
- Most tools use highly-sepcialized techniques for fixed SIDs
- Our approach: Use heap automata as framework instead

Well-determined Symbolic Heaps

Definition

- A reduced symbolic heap is well-determined if it is satisfiable and all of its models are isomorphic.

Well-determined Symbolic Heaps

Definition

- A reduced symbolic heap is well-determined if it is satisfiable and all of its models are isomorphic.
- A symbolic heap is well-determined if its unfoldings are.

Well-determined Symbolic Heaps

Definition

- A reduced symbolic heap is well-determined if it is satisfiable and all of its models are isomorphic.
- A symbolic heap is well-determined if its unfoldings are.
- An SID is well-determined if all symbolic heaps in its rules are.

Well-determined Symbolic Heaps

Definition

- A reduced symbolic heap is well-determined if it is satisfiable and all of its models are isomorphic.
- A symbolic heap is well-determined if its unfoldings are.
- An SID is well-determined if all symbolic heaps in its rules are.

Example

$$\tau(x) \triangleq \exists z. x \mapsto z : \{x \neq z\} \qquad \text{not well-determined}$$

$$\varphi(x) \triangleq \exists z. x \mapsto z * z \mapsto \texttt{null} \qquad \text{well-determined}$$

Entailment between Predicates

Theorem

Let Φ be a well-determined SID over a class $\mathcal{C}$ and P, Q be predicate names of the same rank.

Entailment between Predicates

Theorem

Let Φ be a well-determined SID over a class $\mathcal{C}$ and P, Q be predicate names of the same rank.

Then $P(\mathbf{x}) \models_\Phi Q(\mathbf{x})$ is decidable if there is a heap automaton accepting

$$L(P, \Phi) \triangleq \{\sigma \in \mathit{RSH}_{\mathcal{C}} \mid \exists \tau \in \mathit{unfold}_\Phi(Q) \,.\, \sigma \models \tau\}.$$

Entailment between Predicates

Theorem

Let Φ be a well-determined SID over a class $\mathcal{C}$ and P, Q be predicate names of the same rank.

Then $P(\mathbf{x}) \models_\Phi Q(\mathbf{x})$ is decidable if there is a heap automaton accepting

$$L(P, \Phi) \ \triangleq \ \{\sigma \in \mathit{RSH}_{\mathcal{C}} \mid \exists \tau \in \mathit{unfold}_\Phi(Q) \ . \ \sigma \models \tau\}.$$

Example

(cyclic, doubly-linked) lists, skip-lists, trees, ...

Entailment between Symbolic Heaps

Theorem

Let Φ be a well-determined SID over a class $\mathcal{C}$ and P, Q be predicate names of the same rank.

Entailment between Symbolic Heaps

Theorem

Let Φ be a well-determined SID over a class $\mathcal{C}$ and P, Q be predicate names of the same rank. Moreover, let $\varphi(\mathbf{x})$, $\psi(\mathbf{x})$ be well-determined symbolic heaps over $\mathcal{C}$.

Entailment between Symbolic Heaps

Theorem

Let Φ be a well-determined SID over a class $\mathcal{C}$ and P, Q be predicate names of the same rank. Moreover, let $\varphi(\mathbf{x})$, $\psi(\mathbf{x})$ be well-determined symbolic heaps over $\mathcal{C}$.

Then $\varphi(\mathbf{x}) \models_{\Phi} \psi(\mathbf{x})$ is decidable if there is a heap automaton $\mathfrak{A}(P)$ accepting $L(P, \Phi)$ for each predicate name P occurring in Φ.

Entailment between Symbolic Heaps

Theorem

Let Φ be a well-determined SID over a class $\mathcal{C}$ and P, Q be predicate names of the same rank. Moreover, let $\varphi(\mathbf{x})$, $\psi(\mathbf{x})$ be well-determined symbolic heaps over $\mathcal{C}$.

Then $\varphi(\mathbf{x}) \models_\Phi \psi(\mathbf{x})$ is decidable if there is a heap automaton $\mathfrak{A}(P)$ accepting $L(P, \Phi)$ for each predicate name P occurring in Φ.

Theorem

For each automaton $\mathfrak{A}(P)$ from above, let $|Q_{\mathfrak{A}(P)}| \leq 2^{\mathit{poly}(\alpha)}$ and $| \rightarrow_{\mathfrak{A}(P)} |$ be decidable in $\textsc{ExpTime}$.

Entailment between Symbolic Heaps

Theorem

Let Φ be a well-determined SID over a class $\mathcal{C}$ and P, Q be predicate names of the same rank. Moreover, let $\varphi(\mathbf{x})$, $\psi(\mathbf{x})$ be well-determined symbolic heaps over $\mathcal{C}$.

Then $\varphi(\mathbf{x}) \models_\Phi \psi(\mathbf{x})$ is decidable if there is a heap automaton $\mathfrak{A}(P)$ accepting $L(P, \Phi)$ for each predicate name P occurring in Φ.

Theorem

For each automaton $\mathfrak{A}(P)$ from above, let $|Q_{\mathfrak{A}(P)}| \leq 2^{\mathsf{poly}(\alpha)}$ and $| \rightarrow_{\mathfrak{A}(P)} |$ be decidable in $\textsc{ExpTime}$.

Then the entailment problem is in $\textsc{ExpTime}$.

Entailment between Symbolic Heaps

Theorem

Let Φ be a well-determined SID over a class $\mathcal{C}$ and P, Q be predicate names of the same rank. Moreover, let $\varphi(\mathbf{x})$, $\psi(\mathbf{x})$ be well-determined symbolic heaps over $\mathcal{C}$.

Then $\varphi(\mathbf{x}) \models_{\Phi} \psi(\mathbf{x})$ is decidable if there is a heap automaton $\mathfrak{A}(P)$ accepting $L(P, \Phi)$ for each predicate name P occurring in Φ.

Theorem

For each automaton $\mathfrak{A}(P)$ from above, let $|Q_{\mathfrak{A}(P)}| \leq 2^{\mathit{poly}(\alpha)}$ and $| \rightarrow_{\mathfrak{A}(P)} |$ be decidable in $\textsc{ExpTime}$.

Then the entailment problem is in $\textsc{ExpTime}$.

Even for simple trees entailment becomes $\textsc{ExpTime}$–hard.