Heap Automata

(ESOP 2017)

Christina Jansen\textsuperscript{1}  Jens Katelaan\textsuperscript{2}
Christoph Matheja\textsuperscript{1}  Thomas Noll\textsuperscript{1}  Florian Zuleger\textsuperscript{2}

\textsuperscript{1} RWTH Aachen University
\textsuperscript{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.
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.
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.
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
How do we prove temporal properties about symbolic heaps?

- Generate labeled transition system using shape analysis.
- Every state corresponds to an SL formula $\phi$.
- Apply standard model-checking to transition system.
- Problem: Prove $\phi \models \text{Prop}$ for a few million formulas.
- Synthesize robust SID w.r.t. $\text{Prop}$.
- Run shape analysis space with new SID.
- Efficiently decide $\phi \models \text{Prop}$ without looking into predicates.
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 $\varphi$

Problem: Prove $\varphi \models \text{Prop}$ for a few million formulas

- Synthesize robust SID w.r.t. $\text{Prop}$
- Run shape analysis space with new SID
- Efficiently decide $\varphi \models \text{Prop}$ without looking into predicates
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 $\varphi$
- Apply standard model-checking to transition system
How do we prove temporal properties about symbolic heaps?

- Generate labeled transition system using shape analysis
- Every state corresponds to an SL formula $\varphi$
- Apply standard model-checking to transition system

Problem: Prove $\varphi \models Prop$
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 $\varphi$
  - Apply standard model-checking to transition system
- Problem: Prove $\varphi \models Prop$ for a few million formulas $\varphi$
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 $\varphi$
  - Apply standard model-checking to transition system
- Problem: Prove $\varphi \models Prop$ for a few million formulas $\varphi$
  1 Synthesize robust SID w.r.t. $Prop$
How do we prove temporal properties about symbolic heaps?

- Generate labeled transition system using shape analysis
- Every state corresponds to an SL formula $\varphi$
- Apply standard model-checking to transition system

Problem: Prove $\varphi \models Prop$ for a few million formulas $\varphi$

1. Synthesize robust SID w.r.t. $Prop$
2. Run shape analysis space with new SID
How do we prove temporal properties about symbolic heaps?

- Generate labeled transition system using shape analysis
- Every state corresponds to an SL formula $\varphi$
- Apply standard model-checking to transition system

Problem: Prove $\varphi \models Prop$ for a few million formulas $\varphi$

1. Synthesize robust SID w.r.t. $Prop$
2. Run shape analysis space with new SID
3. Efficiently decide $\varphi \models Prop$ 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

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
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: \[ t ::= x | \text{null} \]
Symbolic Heaps with Inductive Predicates

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

Pure formulas: $\pi ::= t = t \mid t \neq t$  \hspace{1cm} (\Pi : \text{set of pure formulas})
Symbolic Heaps with Inductive Predicates

Terms: \[ t ::= x \mid \text{null} \]

Pure formulas: \[ \pi ::= t = t \mid t \neq t \] \((\Pi: \text{set of pure formulas})\)

Spatial formulas: \[ \Sigma ::= \text{emp} \mid \Sigma \ast \Sigma \] \((t: \text{tuple of terms})\)

- \text{emp} is the empty heap
Symbolic Heaps with Inductive Predicates

Terms: \[ t ::= x \mid \text{null} \]

Pure formulas: \[ \pi ::= t = t \mid t \not= t \] (\(\Pi\) : set of pure formulas)

Spatial formulas: \[ \Sigma ::= \text{emp} \mid x \mapsto t \] (\(t : \text{tuple of terms}\))

- \text{emp} is the empty heap
- \(x \mapsto t\) is a pointer to a single record
Symbolic Heaps with Inductive Predicates

Terms: \[ t ::= x \mid \text{null} \]

Pure formulas: \[ \pi ::= t = t \mid t \neq t \quad (\Pi : \text{set of pure formulas}) \]

Spatial formulas: \[ \Sigma ::= \text{emp} \mid x \mapsto t \mid \Sigma \ast \Sigma \quad (t : \text{tuple of terms}) \]

- \text{emp} is the empty heap
- \( x \mapsto t \) is a pointer to a single record
- \( \ast \) is the separating conjunction of two domain-disjoint heaps.
Symbolic Heaps with Inductive Predicates

Terms: \( t ::= x \mid \text{null} \)

Pure formulas: \( \pi ::= t = t \mid t \neq t \) \((\Pi : \text{set of pure formulas})\)

Spatial formulas: \( \Sigma ::= \text{emp} \mid x \mapsto t \mid \Sigma \ast \Sigma \) \((t : \text{tuple of terms})\)

Predicate calls: \( \Gamma ::= \text{emp} \mid P(t) \mid \Gamma \ast \Gamma \) \((P : \text{predicate symbol})\)

- emp is the empty heap
- \( x \mapsto t \) is a pointer to a single record
- \( \ast \) is the separating conjunction of two domain-disjoint heaps.
Symbolic Heaps with Inductive Predicates

Terms: \( t ::= x \mid \text{null} \)

Pure formulas: \( \pi ::= t = t \mid t \neq t \) \((\Pi : \text{set of pure formulas})\)

Spatial formulas: \( \Sigma ::= \text{emp} \mid x \mapsto t \mid \Sigma \ast \Sigma \) \((t : \text{tuple of terms})\)

Predicate calls: \( \Gamma ::= \text{emp} \mid P(t) \mid \Gamma \ast \Gamma \) \((P : \text{predicate symbol})\)

Symbolic heaps (SH): \( \varphi(x) ::= \exists z. \Sigma \ast \Gamma : \Pi \) \((x, z : \text{tuples of variables})\)

- \(\text{emp}\) is the empty heap
- \(x \mapsto t\) is a pointer to a single record
- \(\ast\) is the separating conjunction of two domain-disjoint heaps.
Symbolic Heaps with Inductive Predicates

Terms:  \( t ::= x \mid \text{null} \)

Pure formulas:  \( \pi ::= t = t \mid t \neq t \)  \( (\Pi : \text{set of pure formulas}) \)

Spatial formulas:  \( \Sigma ::= \text{emp} \mid x \mapsto t \mid \Sigma \ast \Sigma \)  \( (t : \text{tuple of terms}) \)

Predicate calls:  \( \Gamma ::= \text{emp} \mid P(t) \mid \Gamma \ast \Gamma \)  \( (P : \text{predicate symbol}) \)

Symbolic heaps (SH):  \( \varphi(x) ::= \exists z. \Sigma \ast \Gamma : \Pi \)  \( (x, z : \text{tuples of variables}) \)

\( \varphi(x) \) is reduced if \( \Gamma = \text{emp} \)

- \( \text{emp} \) is the empty heap
- \( x \mapsto t \) is a pointer to a single record
- \( \ast \) is the separating conjunction of two domain-disjoint heaps.
Systems of Inductive Definitions (SIDs)

An SID $\Phi$ is a finite set of rules of the form

$$\exists z. \Sigma * \Gamma : \Pi \Rightarrow P(x)$$

Example (Binary trees)

- $\text{emp}: \{x = \text{null} \} \Rightarrow \text{tree}(x)$
- $\exists y,z. x \mapsto \cdot (y,z) * \text{tree}(y) * \text{tree}(z) \Rightarrow \text{tree}(x)$

Semantics of predicate calls is given by unfolding to reduced SHs collected in $\text{unfold } \Phi(P(x))$. 
An SID $\Phi$ is a finite set of rules of the form

$$\exists z . \Sigma * \Gamma : \Pi \Rightarrow P(x)$$

**Example (Binary trees)**

- $\text{emp} : \{x = \text{null}\} \Rightarrow \text{tree}(x)$
- $\exists y, z . x \mapsto (y, z) * \text{tree}(y) * \text{tree}(z) \Rightarrow \text{tree}(x)$
Systems of Inductive Definitions (SIDs)

An SID $\Phi$ is a finite set of rules of the form

$$\exists z. \Sigma \ast \Gamma : \Pi \Rightarrow P(x)$$

**Example (Binary trees)**

$$\text{emp} : \{x = \text{null}\} \Rightarrow \text{tree}(x)$$

$$\exists y, z. x \mapsto (y, z) \ast \text{tree}(y) \ast \text{tree}(z) \Rightarrow \text{tree}(x)$$

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

$$P(x,y) \xrightarrow{\text{unfold}} \exists (z_1, z_2). \Sigma * P_1(z_1, z_2) * P_2(z_2, y) : \Pi$$
Robustness Properties: Subtleties

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

\[ P(x, y) \xrightarrow{\text{unfold}} \exists (z_1, z_2). \sum_{x \mapsto z_1} * P_1(z_1, z_2) * P_2(z_2, y) : \Pi \]
Robustness Properties: Subtleties

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

$$P(x, y) \xrightarrow{\text{unfold}} \exists(z_1, z_2) \cdot \sum_{x \mapsto z_1} P_1(z_1, z_2) \cdot P_2(z_2, y) : \Pi$$

- Reachability might depend on unfoldings of all predicates
Robustness Properties: Subtleties

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

$$P(x, y) \xrightarrow{\text{unfold}} \exists(z_1, z_2) \cdot \sum_{x \mapsto z_1} * P_1(z_1, z_2) * P_2(z_2, y) : \Pi$$

- 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 \( P(x, y) \)?

\[
P(x, y) \xrightarrow{\text{unfold}} \exists(z_1, z_2). \sum_{x \mapsto z_1} P_1(z_1, z_2) \times P_2(z_2, y) : \Pi
\]

- 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 $P(x, y)$?

$$P(x, y) \xrightarrow{\text{unfold}} \exists(z_1, z_2). \sum_{x \mapsto z_1} P_1(z_1, z_2) \ast P_2(z_2, y) : \Pi$$

- 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 $P(x, y)$?

$$P(x, y) \xrightarrow{\text{unfold}} \exists (z_1, z_2) \cdot \sum_{x \rightarrow z_1} \ast \overset{z_1=z_2}{P_1(z_1, z_2)} \ast \overset{z_2 \rightarrow y}{P_2(z_2, y)} : \Pi$$

- 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 $P(x, y)$?

$$P(x, y) \xrightarrow{\text{unfold}} \exists(z_1, z_2) . \sum_{x \rightarrow z_1} \bullet P_1(z_1, z_2) \bullet P_2(z_2, y) : \Pi$$

- 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

\[ \varphi(x) = \exists z \cdot \Sigma \ast P_1(x_1) \ast \ldots \ast P_m(x_m) : \Pi \]
Heap Automata: Compositionality

We reason compositionally while unfolding a symbolic heap

$$\varphi(x) = \exists z. \sum \ast P_1(x_1) \ast \ldots \ast P_m(x_m) : \Pi$$

**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

\[ \varphi(x) = \exists z . \sum \ast P_1(x_1) \ast \ldots \ast P_m(x_m) : \Pi \]

**Soundness:** If \( P_k \) has an unfolding with property \( q_k \) \( (1 \leq k \leq m) \) and for those unfoldings \( \varphi(x) \) has property \( q \)
Heap Automata: Compositionality

We reason compositionally while unfolding a symbolic heap

\[
\varphi(x) = \exists z . \sum * \left( P_1(x_1) * \ldots * P_m(x_m) \right) : \Pi
\]

**Soundness:** If \( P_k \) has an unfolding with property \( q_k \) \((1 \leq k \leq m)\) and for those unfoldings \( \varphi(x) \) has property \( q \) then \( \varphi(x) \) has an unfolding with property \( q \).
Heap Automata: Compositionality

We reason compositionally while unfolding a symbolic heap

$$\varphi(x) = \exists z . \sum \ast P_1(x_1) \ast \ldots \ast P_m(x_m) : \Pi$$

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

**Completeness:** If $\varphi(x)$ has an unfolding with property $q$
Heap Automata: Compositionality

We reason **compositionally** while unfolding a symbolic heap

\[ \varphi(x) = \exists z . \sum * \left( P_1(x_1) * \ldots * P_m(x_m) \right) : \Pi \] 

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

**Completeness:** If \( \varphi(x) \) 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

\[ \varphi(x) = \exists z . \sum \ast P_1(x_1) \ast \ldots \ast P_m(x_m) : \Pi \]

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

**Completeness:** If \( \varphi(x) \) has an unfolding with property \( q \) then there are unfoldings of \( P_k \) with some property \( q_k \) and for those unfoldings \( \varphi(x) \) has property \( q \).
Heap Automata: Definition

Definition

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

- $Q$ is a finite set of states,
- $F \subseteq Q$ is a set of final states, and
A heap automaton is a tuple \( A = (Q, \rightarrow, F) \), where

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

Theorem (Refinement Theorem)
One can effectively construct an SID $\Psi$ such that $\forall P: \text{unfold } \Psi(P(x)) = \text{unfold } \Phi(P(x)) \cap L(A)$. 

Theorem 1
$\text{size}(\Psi) \leq \text{size}(\Phi) \cdot \text{size}(A)$. 

It is decidable in linear time whether $\text{unfold } \Phi(\varphi(x))$ is empty. 

Languages of heap automata are effectively closed under union, intersection and complement. 

It is decidable whether $\text{unfold } \Phi(\varphi(x)) \cap L(A) \neq \emptyset$. 

It is decidable whether $\text{unfold } \Phi(\varphi(x)) \subseteq L(A)$. 

12
Heap Automata: Results

Given SID $\Phi$, heap automaton $A$, and
Heap Automata: Results

Given SID $\Phi$, heap automaton $A$, and symbolic heap $\varphi(x)$. . .
Heap Automata: Results

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

**Theorem (Refinement Theorem)**

*One can effectively construct an SID $\Psi$ such that*

$$\forall P : \text{unfold}_\Psi(P(x)) = \text{unfold}_\Phi(P(x)) \cap L(\mathcal{A}).$$
Heap Automata: Results

Given SID \( \Phi \), heap automaton \( A \), and symbolic heap \( \varphi(x) \)...

**Theorem (Refinement Theorem)**

One can effectively construct an SID \( \Psi \) such that

\[
\forall P : unfold_\Psi(P(x)) = unfold_\Phi(P(x)) \cap L(A).
\]

**Theorem**

1. \( size(\Psi) \leq size(\Phi) \cdot size(A) \# \text{pred. calls} \)
Heap Automata: Results

Given SID $\Phi$, heap automaton $A$, and symbolic heap $\varphi(x)$. . .

Theorem (Refinement Theorem)

One can effectively construct an SID $\Psi$ such that

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

Theorem

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

2. It is decidable in linear time whether $\text{unfold}_\Phi(\varphi(x))$ is empty.
Heap Automata: Results

Given SID $\Phi$, heap automaton $A$, and symbolic heap $\varphi(x)$...

**Theorem (Refinement Theorem)**

*One can effectively construct an SID $\Psi$ such that*

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

**Theorem**

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

2. *It is decidable in linear time whether $\text{unfold}_\Phi(\varphi(x))$ is empty.*

3. *Languages of heap automata are effectively closed under union, intersection and complement.*
Heap Automata: Results

Given SID $\Phi$, heap automaton $A$, and symbolic heap $\varphi(x)$...

**Theorem (Refinement Theorem)**

One can effectively construct an SID $\Psi$ such that

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

**Theorem**

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

Given SID $\Phi$, heap automaton $A$, and symbolic heap $\varphi(x)$...

**Theorem (Refinement Theorem)**

One can effectively construct an SID $\Psi$ such that

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

**Theorem**

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

We constructed heap automata for the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>FV bounded satisfiability</td>
<td>ExpTime, $\text{coNP}$-complete</td>
</tr>
<tr>
<td>model-checking</td>
<td>ExpTime, $\text{coNP}$-complete</td>
</tr>
<tr>
<td>garbage-freedom</td>
<td>ExpTime, $\text{coNP}$-complete</td>
</tr>
<tr>
<td>acyclicity</td>
<td>ExpTime, $\text{coNP}$-complete</td>
</tr>
<tr>
<td>reachability</td>
<td>ExpTime, $\text{coNP}$-complete</td>
</tr>
<tr>
<td>establishment</td>
<td>ExpTime, $\text{coNP}$-complete</td>
</tr>
</tbody>
</table>

(Brotherston et al., 2014) and (Brotherston et al., 2016)

All of these problems are $\text{PTime}$–complete for an additionally bounded number of predicate calls.
A Zoo of Robustness Properties

We constructed heap automata for the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfiability</td>
<td>ExpTime -C1</td>
</tr>
<tr>
<td></td>
<td>NP -C1</td>
</tr>
<tr>
<td></td>
<td>coNP -C1</td>
</tr>
<tr>
<td></td>
<td>ExpTime -C1</td>
</tr>
<tr>
<td></td>
<td>coNP -C1</td>
</tr>
<tr>
<td></td>
<td>ExpTime -C1</td>
</tr>
<tr>
<td></td>
<td>coNP -C1</td>
</tr>
</tbody>
</table>

(Brotherston et al., 2014) and (Brotherston et al., 2016)

All of these problems are PTime-complete for an additionally bounded number of predicate calls.
We constructed heap automata for the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfiability</td>
<td>ExpTime -C</td>
</tr>
<tr>
<td>model-checking</td>
<td>NP -C</td>
</tr>
<tr>
<td>garbage-freedom</td>
<td>ExpTime -C</td>
</tr>
<tr>
<td>acyclicity</td>
<td>ExpTime -C</td>
</tr>
<tr>
<td>reachability</td>
<td>ExpTime -C</td>
</tr>
<tr>
<td>establishment</td>
<td>ExpTime -C</td>
</tr>
</tbody>
</table>

13

(Brotherston et al., 2014) and (Brotherston et al., 2016) All of these problems are PTime–complete for an additionally bounded number of predicate calls.
A Zoo of Robustness Properties

We constructed heap automata for the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfiability</td>
<td>ExpTime -C 1</td>
</tr>
<tr>
<td>model-checking</td>
<td>ExpTime -C 1</td>
</tr>
<tr>
<td>garbage-freedom</td>
<td>ExpTime coNP -C 1</td>
</tr>
<tr>
<td>acyclicity</td>
<td>ExpTime coNP -C 1</td>
</tr>
<tr>
<td>reachability</td>
<td>ExpTime coNP -C 1</td>
</tr>
<tr>
<td>establishment</td>
<td>ExpTime coNP -C 1</td>
</tr>
</tbody>
</table>

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

(Brotherston et al., 2014) and (Brotherston et al., 2016)
A Zoo of Robustness Properties

We constructed heap automata for the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfiability</td>
<td>ExpTime -C</td>
</tr>
<tr>
<td>model-checking</td>
<td>ExpTime -C</td>
</tr>
<tr>
<td>garbage-freedom</td>
<td>ExpTime -C</td>
</tr>
<tr>
<td>acyclicity</td>
<td>ExpTime -C</td>
</tr>
</tbody>
</table>

establishment

All of these problems are $PTime -complete for an additionally bounded number of predicate calls.
A Zoo of Robustness Properties

We constructed heap automata for the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfiability</td>
<td>ExpTime-C</td>
</tr>
<tr>
<td>model-checking</td>
<td>ExpTime-C</td>
</tr>
<tr>
<td>garbage-freedom</td>
<td>ExpTime-coNP-C</td>
</tr>
<tr>
<td>acyclicity</td>
<td>ExpTime-coNP-C</td>
</tr>
<tr>
<td>reachability</td>
<td>ExpTime-coNP-C</td>
</tr>
</tbody>
</table>

13
### A Zoo of Robustness Properties

We constructed heap automata for the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfiability</td>
<td>ExpTime-C1</td>
</tr>
<tr>
<td>model-checking</td>
<td>ExpTime-C1</td>
</tr>
<tr>
<td>garbage-freedom</td>
<td>ExpTime-C1</td>
</tr>
<tr>
<td>acyclicity</td>
<td>ExpTime-C1</td>
</tr>
<tr>
<td>reachability</td>
<td>ExpTime-C1</td>
</tr>
<tr>
<td>establishment</td>
<td>ExpTime-C1</td>
</tr>
</tbody>
</table>

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

(Brotherston et al., 2014) and (Brotherston et al., 2016)
A Zoo of Robustness Properties

We constructed heap automata for the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfiability</td>
<td>ExpTime-C$^1$</td>
</tr>
<tr>
<td>model-checking</td>
<td>ExpTime-C$^1$</td>
</tr>
<tr>
<td>garbage-freedom</td>
<td></td>
</tr>
<tr>
<td>acyclicity</td>
<td></td>
</tr>
<tr>
<td>reachability</td>
<td></td>
</tr>
<tr>
<td>establishment</td>
<td></td>
</tr>
</tbody>
</table>

1 (Brotherston et al., 2014) and (Brotherston et al., 2016)
A Zoo of Robustness Properties

We constructed heap automata for the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfiability</td>
<td>ExpTime-C$^1$</td>
</tr>
<tr>
<td>model-checking</td>
<td>ExpTime-C$^1$</td>
</tr>
<tr>
<td>garbage-freedom</td>
<td>ExpTime-C</td>
</tr>
<tr>
<td>acyclicity</td>
<td>ExpTime-C</td>
</tr>
<tr>
<td>reachability</td>
<td>ExpTime-C</td>
</tr>
<tr>
<td>establishment</td>
<td>ExpTime-C</td>
</tr>
</tbody>
</table>

1 (Brotherston et al., 2014) and (Brotherston et al., 2016)
## A Zoo of Robustness Properties

We constructed heap automata for the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
<th>FV bounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfiability</td>
<td>EXP(^{\text{TIME}})-C(^{1})</td>
<td></td>
</tr>
<tr>
<td>model-checking</td>
<td>EXP(^{\text{TIME}})-C(^{1})</td>
<td></td>
</tr>
<tr>
<td>garbage-freedom</td>
<td>EXP(^{\text{TIME}})-C</td>
<td></td>
</tr>
<tr>
<td>acyclicity</td>
<td>EXP(^{\text{TIME}})-C</td>
<td></td>
</tr>
<tr>
<td>reachability</td>
<td>EXP(^{\text{TIME}})-C</td>
<td></td>
</tr>
<tr>
<td>establishment</td>
<td>EXP(^{\text{TIME}})-C</td>
<td></td>
</tr>
</tbody>
</table>

\(^{1}\) (Brotherston et al., 2014) and (Brotherston et al., 2016)
A Zoo of Robustness Properties

We constructed heap automata for the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
<th>FV bounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfiability</td>
<td>$\text{ExpTime-C}^1$</td>
<td>$\text{NP-C}^1$</td>
</tr>
<tr>
<td>model-checking</td>
<td>$\text{ExpTime-C}^1$</td>
<td>$\text{NP-C}^1$</td>
</tr>
<tr>
<td>garbage-freedom</td>
<td>$\text{ExpTime-C}$</td>
<td></td>
</tr>
<tr>
<td>acyclicity</td>
<td>$\text{ExpTime-C}$</td>
<td></td>
</tr>
<tr>
<td>reachability</td>
<td>$\text{ExpTime-C}$</td>
<td></td>
</tr>
<tr>
<td>establishment</td>
<td>$\text{ExpTime-C}$</td>
<td></td>
</tr>
</tbody>
</table>

1 (Brotherston et al., 2014) and (Brotherston et al., 2016)
### A Zoo of Robustness Properties

We constructed heap automata for the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
<th>FV bounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfiability</td>
<td>$\text{ExpTime-C}^1$</td>
<td>$\text{NP-C}^1$</td>
</tr>
<tr>
<td>model-checking</td>
<td>$\text{ExpTime-C}^1$</td>
<td>$\text{NP-C}^1$</td>
</tr>
<tr>
<td>garbage-freedom</td>
<td>$\text{ExpTime-C}$</td>
<td>$\text{coNP-C}$</td>
</tr>
<tr>
<td>acyclicity</td>
<td>$\text{ExpTime-C}$</td>
<td>$\text{coNP-C}$</td>
</tr>
<tr>
<td>reachability</td>
<td>$\text{ExpTime-C}$</td>
<td>$\text{coNP-C}$</td>
</tr>
<tr>
<td>establishment</td>
<td>$\text{ExpTime-C}$</td>
<td>$\text{coNP-C}$</td>
</tr>
</tbody>
</table>

1 (Brotherston et al., 2014) and (Brotherston et al., 2016)
A Zoo of Robustness Properties

We constructed heap automata for the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Complexity</th>
<th>FV bounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfiability</td>
<td>(\text{ExpTime-C}^1)</td>
<td>(\text{NP-C}^1)</td>
</tr>
<tr>
<td>model-checking</td>
<td>(\text{ExpTime-C}^1)</td>
<td>(\text{NP-C}^1)</td>
</tr>
<tr>
<td>garbage-freedom</td>
<td>(\text{ExpTime-C})</td>
<td>(\text{coNP-C})</td>
</tr>
<tr>
<td>acyclicity</td>
<td>(\text{ExpTime-C})</td>
<td>(\text{coNP-C})</td>
</tr>
<tr>
<td>reachability</td>
<td>(\text{ExpTime-C})</td>
<td>(\text{coNP-C})</td>
</tr>
<tr>
<td>establishment</td>
<td>(\text{ExpTime-C})</td>
<td>(\text{coNP-C})</td>
</tr>
</tbody>
</table>

\(^1\) (Brotherston et al., 2014) and (Brotherston et al., 2016)

All of these problems are \(\text{PTime}\)–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\textsuperscript{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.

\textsuperscript{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\textsuperscript{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

---

\textsuperscript{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\textsuperscript{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

\textsuperscript{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 \textit{Attestor}
- Supported heap automata in LTL formulas:
  - Reachability
  - Acyclicity
  - Garbage-Freedom
  - Reachability
  - \textbf{Shape}: The heap is a tree, sll, dll... 
  - \textbf{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

<table>
<thead>
<tr>
<th>Program</th>
<th>Property</th>
<th>SSG (s)</th>
<th>Model-Checking (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLL.reversal</td>
<td>reachability</td>
<td>0.12</td>
<td>0.02</td>
</tr>
<tr>
<td>SLL.reversal</td>
<td>completeness</td>
<td>0.13</td>
<td>0.02</td>
</tr>
<tr>
<td>DLL.traversal</td>
<td>completeness</td>
<td>0.24</td>
<td>0.10</td>
</tr>
<tr>
<td>DLL.traversal</td>
<td>preservation</td>
<td>0.33</td>
<td>0.32</td>
</tr>
<tr>
<td>DLL.reversal</td>
<td>shape</td>
<td>0.14</td>
<td>0.05</td>
</tr>
<tr>
<td>DLL.reversal</td>
<td>reachability</td>
<td>0.18</td>
<td>0.02</td>
</tr>
<tr>
<td>DLL.reversal</td>
<td>completeness</td>
<td>0.24</td>
<td>0.15</td>
</tr>
<tr>
<td>BT.lindstrom</td>
<td>term. at root</td>
<td>0.19</td>
<td>0.03</td>
</tr>
<tr>
<td>BT.lindstrom</td>
<td>shape</td>
<td>0.20</td>
<td>0.17</td>
</tr>
<tr>
<td>BT.lindstrom</td>
<td>completeness</td>
<td>0.62</td>
<td>0.46</td>
</tr>
<tr>
<td>BT.lindstrom</td>
<td>preservation</td>
<td>0.38</td>
<td>0.70</td>
</tr>
</tbody>
</table>
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 $\varphi$, $\psi$ and SID $\Phi$ without dangling pointers
What else?

Heap automata...

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

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

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

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?
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

\[ \varphi[P/\tau] \text{ unfolds } P \text{ by } \tau \]
Heap Automata: Formal Definition of Compositionality

\( \varphi[P/\tau] \) unfolds \( P \) by \( \tau \)

Definition

A heap automaton \( \mathcal{A} = (Q, SH_C, \rightarrow, F) \) is compositional if
Heap Automata: Formal Definition of Compositionality

ϕ[P/τ] unfolds P by τ

Definition

A heap automaton \( \mathcal{A} = (Q, \text{SH}_C, \rightarrow, F) \) is compositional if for every \( p \in Q \) and every \( \varphi \in \text{SH}_C \) with predicate calls \( P_1, \ldots, P_m \) and all reduced symbolic heaps \( \tau_1, \ldots, \tau_m \in \text{RSH}_C \):
Heap Automata: Formal Definition of Compositionality

\( \varphi[P/\tau] \) unfolds \( P \) by \( \tau \)

**Definition**

A heap automaton \( \mathcal{A} = (Q, SH_C, \rightarrow, F) \) is **compositional** if for every \( p \in Q \) and every \( \varphi \in SH_C \) with predicate calls \( P_1, \ldots, P_m \) and all reduced symbolic heaps \( \tau_1, \ldots, \tau_m \in RSH_C \):

\[
\exists q \in Q^m . \ q \xrightarrow{\varphi} p
\]
Definition

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

$$\exists q \in Q^m \cdot q \xrightarrow{\varphi} p \quad \text{and} \quad \bigwedge_{1 \leq i \leq m} \varepsilon \xrightarrow{\tau_i} q[i]$$
Heap Automata: Formal Definition of Compositionality

\( \varphi[P/\tau] \) unfolds \( P \) by \( \tau \)

**Definition**

A heap automaton \( \mathcal{A} = (Q, SH_C, \rightarrow, F) \) is **compositional** if for every \( p \in Q \) and every \( \varphi \in SH_C \) with predicate calls \( P_1, \ldots, P_m \) and all reduced symbolic heaps \( \tau_1, \ldots, \tau_m \in RSH_C \):

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

if and only if

\[ L(\mathcal{A}) \equiv \{ \tau \in RSH_C | \exists p \in F . \in \xrightarrow{\tau} p \} \]
Heap Automata: Formal Definition of Compositionality

\[ \varphi[P/\tau] \text{ unfolds } P \text{ by } \tau \]

**Definition**

A heap automaton \( \mathcal{A} = (Q, SH_C, \to, F) \) is compositional if for every \( p \in Q \) and every \( \varphi \in SH_C \) with predicate calls \( P_1, \ldots, P_m \) and all reduced symbolic heaps \( \tau_1, \ldots, \tau_m \in RSH_C \):

\[ \exists q \in Q^m \cdot q \xrightarrow{\varphi} p \text{ and } \bigwedge_{1 \leq i \leq m} \varepsilon \xrightarrow{\tau_i} q[i] \]

if and only if

\[ \varepsilon \xrightarrow{\varphi[P_1/\tau_1, \ldots, P_m/\tau_m]} p \]
Heap Automata: Formal Definition of Compositionality

\[ \varphi[P/\tau] \text{ unfolds } P \text{ by } \tau \]

**Definition**

A heap automaton \( \mathcal{A} = (Q, SH_C, \rightarrow, F) \) is **compositional** if for every \( p \in Q \) and every \( \varphi \in SH_C \) with predicate calls \( P_1, \ldots, P_m \) and all reduced symbolic heaps \( \tau_1, \ldots, \tau_m \in RSH_C \):

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

if and only if

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

\[
L(\mathcal{A}) \triangleq \{ \tau \in RSH_C \mid \exists p \in F . \varepsilon \xrightarrow{\tau} p \} 
\]
The Entailment Problem

**Definition (Entailment Problem)**

Given an SID $\Phi$ and symbolic heaps $\varphi, \psi$, decide whether

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

Crucial for automated program verification based on separation logic.

Antonopolous et al.: The entailment problem is undecidable.

Most tools use highly specialized techniques for fixed SIDs.

Our approach: Use heap automata as framework instead.
The Entailment Problem

Definition (Entailment Problem)

Given an SID $\Phi$ and symbolic heaps $\varphi, \psi$, 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 $\Phi$ and symbolic heaps $\varphi, \psi$, decide whether

$\varphi \models_\Phi \psi \iff \forall s, h . s, h \models_\Phi \varphi \implies s, h \models_\Phi \psi$

- 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 $\Phi$ and symbolic heaps $\varphi, \psi$, decide whether

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

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

Definition (Entailment Problem)

Given an SID $\Phi$ and symbolic heaps $\varphi, \psi$, decide whether

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

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

Example

\[
\tau(x) \equiv \exists z. x \mapsto z : \{x \neq z\}
\]

not well-determined

\[
\phi(x) \equiv \exists z. x \mapsto z \ast z \mapsto \text{null}
\]

well-determined
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**

\[
\tau(x) \triangleq \exists z. x \mapsto z : \{x \neq z\} \quad \text{not well-determined}
\]

\[
\phi(x) \triangleq \exists z. x \mapsto z * z \mapsto \text{null} \quad \text{well-determined}
\]
Entailment between Predicates

**Theorem**

*Let $\Phi$ be a well-determined SID over a class $C$ and $P, Q$ be predicate names of the same rank.*
Entailment between Predicates

*Theorem*

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

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

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

Theorem

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

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

$$L(P, \Phi) \triangleq \{ \sigma \in RSH_\mathcal{C} \mid \exists \tau \in unfold_\Phi(Q) . \sigma \models \tau \}.$$

Example

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

Let $\Phi$ be a well-determined SID over a class $\mathcal{C}$ and $P, Q$ be predicate names of the same rank. Moreover, let $\phi(x)$, $\psi(x)$ be well-determined symbolic heaps over $\mathcal{C}$. Then $\phi(x) \models \Phi \psi(x)$ is decidable if there is a heap automaton $A(P)$ accepting $L(P, \Phi)$ for each predicate name $P$ occurring in $\Phi$. Theorem

For each automaton $A(P)$ from above, let $\mid Q \mid A(P) \leq 2^{\text{poly}(\alpha)}$ and $\mid \rightarrow A(P) \mid$ be decidable in $\text{ExpTime}$. Then the entailment problem is in $\text{ExpTime}$. Even for simple trees entailment becomes $\text{ExpTime}$–hard.
Entailment between Symbolic Heaps

Theorem

Let $\Phi$ be a well-determined SID over a class $C$ and $P,Q$ be predicate names of the same rank. Moreover, let $\varphi(x), \psi(x)$ be well-determined symbolic heaps over $C$. Then $\varphi(x) \models \Phi \psi(x)$ is decidable if there is a heap automaton $A(P)$ accepting $L(P, \Phi)$ for each predicate name $P$ occurring in $\Phi$. Theorem

For each automaton $A(P)$ from above, let $\|Q\|_A(P) \leq 2^{\text{poly}(\alpha)}$ and $\|\rightarrow A(P)\|$ be decidable in ExpTime. Then the entailment problem is in ExpTime. Even for simple trees entailment becomes ExpTime–hard.
Entailment between Symbolic Heaps

Theorem

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

Then \( \varphi(x) \models_{\Phi} \psi(x) \) is decidable if there is a heap automaton \( A(P) \) accepting \( L(P, \Phi) \) for each predicate name \( P \) occurring in \( \Phi \).
Entailment between Symbolic Heaps

Theorem

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

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

Theorem

For each automaton $A(P)$ from above, let $|Q_{A(P)}| \leq 2^{\text{poly}(\alpha)}$ and $|\rightarrow A(P)|$ be decidable in $\text{EXPTIME}$.
Entailment between Symbolic Heaps

**Theorem**

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

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

**Theorem**

For each automaton $\mathcal{A}(P)$ from above, let $|Q_{\mathcal{A}(P)}| \leq 2^{\text{poly}(\alpha)}$ and $|\rightarrow_{\mathcal{A}(P)}|$ be decidable in $\text{EXPTime}$. Then the entailment problem is in $\text{EXPTime}$. 
Entailment between Symbolic Heaps

Theorem

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

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

Theorem

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

Then the entailment problem is in ExpTime.

Even for simple trees entailment becomes ExpTime–hard.