Compositional Refinement of Separation Logic with Recursive Definitions

Christina Jansen    Christoph Matheja    Thomas Noll

Software Modeling and Verification Group

http://moves.rwth-aachen.de/

DCON 2016
March 7, 2016; Saarbrücken
Motivation

Okay, human.  
Huh?  
Before you hit 'compile', listen up.

You know when you're falling asleep, and you imagine yourself walking or something.

And suddenly you misstep, stumble, and jolt awake?

Yeah!

Well, that's what a segfault feels like.  
Double-check your damn pointers, okay?

https://xkcd.com/371
Motivation

Separation Logic

- Hoare Logic
- Pointers
- Dynamic data structures
- Local reasoning
Motivation

https://xkcd.com/371

Concurrent Separation Logic

- Hoare Logic
- Pointers
- Dynamic data structures
- Local reasoning
- Permissions (omitted)
Motivation

Concurrent Separation Logic

- Hoare Logic
- Pointers
- Dynamic data structures
- Local reasoning
- Permissions (omitted)

https://xkcd.com/371
Some Informal Properties

- Satisfiability: there exists a model of $\varphi$

- Model-Checking: $h$ is a model of $\varphi$

- Entailment: each model of $\varphi$ is a model of

- Establishment: each variable in $\varphi$ is eventually allocated

- Equality: $x, y$ are guaranteed to be equal in $\varphi$

- Reachability: $y$ is reachable from $x$ in each heap specified by $\varphi$

- Connectivity: each heap specified by $\varphi$ is connected
Some Informal Properties

- Satisfiability: there exists a model of \( \varphi \)

- Model-Checking: \( h \) is a model of \( \varphi \)
Some Informal Properties

- Satisfiability: there exists a model of \( \varphi \)
- Model-Checking: \( h \) is a model of \( \varphi \)
- Entailment: each model of \( \varphi \) is a model of \( \psi \)
Some Informal Properties

- Satisfiability: there exists a model of $\varphi$

- Model-Checking: $h$ is a model of $\varphi$

- Entailment: each model of $\varphi$ is a model of $\psi$

- Establishment: each variable in $\varphi$ is eventually allocated
Some Informal Properties

- Satisfiability: there exists a model of $\varphi$

- Model-Checking: $h$ is a model of $\varphi$

- Entailment: each model of $\varphi$ is a model of $\psi$

- Establishment: each variable in $\varphi$ is eventually allocated

- Equality: $x, y$ are guaranteed to be equal in $\varphi$
Some Informal Properties

- Satisfiability: there exists a model of $\varphi$

- Model-Checking: $h$ is a model of $\varphi$

- Entailment: each model of $\varphi$ is a model of $\psi$

- Establishment: each variable in $\varphi$ is eventually allocated

- Equality: $x, y$ are guaranteed to be equal in $\varphi$

- Reachability: $y$ is reachable from $x$ in each heap specified by $\varphi$
Some Informal Properties

- Satisfiability: there exists a model of $\varphi$
- Model-Checking: $h$ is a model of $\varphi$
- Entailment: each model of $\varphi$ is a model of $\psi$
- Establishment: each variable in $\varphi$ is eventually allocated
- Equality: $x, y$ are guaranteed to be equal in $\varphi$
- Reachability: $y$ is reachable from $x$ in each heap specified by $\varphi$
- Connectivity: each heap specified by $\varphi$ is connected
Some Informal Properties

- Satisfiability: there exists a model of $\varphi$
- Model-Checking: $h$ is a model of $\varphi$
- Entailment: each model of $\varphi$ is a model of $\psi$
- Establishment: each variable in $\varphi$ is eventually allocated
- Equality: $x, y$ are guaranteed to be equal in $\varphi$
- Reachability: $y$ is reachable from $x$ in each heap specified by $\varphi$
- Connectivity: each heap specified by $\varphi$ is connected

How to decide each of these properties?
Some Informal Properties

- Satisfiability: there exists a model of $\varphi$
- Model-Checking: $h$ is a model of $\varphi$
- Entailment: each model of $\varphi$ is a model of $\psi$
- Establishment: each variable in $\varphi$ is eventually allocated
- Equality: $x, y$ are guaranteed to be equal in $\varphi$
- Reachability: $y$ is reachable from $x$ in each heap specified by $\varphi$
- Connectivity: each heap specified by $\varphi$ is connected

How to decide and establish each of these properties?
Outline

1. Motivation

2. Symbolic Heaps with Recursive Definitions

3. A Refinement Theorem

4. Applications

5. Conclusion
Symbolic Heaps with Recursive Definitions

States: $\sigma = (s, h)$, stack $s : \text{Var} \rightarrow \mathbb{Z}$, heap $h : \mathbb{Z} \rightarrow \mathbb{Z}^+$

Terms are variables $x, y, z, \ldots$ or the constant nil.

Spatial formulas $\Sigma$ and pure formulas $\pi$ are given by

$$
\Sigma ::= \text{emp} \mid x \mapsto a \mid P a \mid \Sigma \ast \Sigma
$$

$$
\pi ::= a = b \mid a \neq b
$$

where $P$ is a predicate symbol and $a, b$ are tuples of terms.

Symbolic heap: $\varphi x \triangleq \exists z . \Sigma : \Pi$, where $\Pi$ is a finite set of pure formulas
Symbolic Heaps with Recursive Definitions

States: $\sigma = (s, h)$, stack $s : Var \rightarrow \mathbb{Z}$, heap $h : \mathbb{Z} \rightarrow \mathbb{Z}^+$

Terms are variables $x, y, z, \ldots$ or the constant nil.

Spatial formulas $\Sigma$ and pure formulas $\pi$ are given by

\[
\Sigma ::= \text{emp} \mid x \mapsto a \mid Pa \mid \Sigma \ast \Sigma
\]
\[
\pi ::= a = b \mid a \neq b
\]

where $P$ is a predicate symbol and $a, b$ are tuples of terms.

Symbolic heap: $\varphi x \triangleq \exists z . \Sigma : \Pi$, where $\Pi$ is a finite set of pure formulas

System of recursive definitions: finite set $\Phi$ of rules $P \Rightarrow \varphi x$

Example

\[
tree \Rightarrow \text{emp} : \{x = \text{nil}\}
\]
\[
tree \Rightarrow \exists y, z . x \mapsto y, z \ast tree y \ast tree z
\]
Example: Destruction of a Binary Tree

Example

\[
\begin{align*}
\text{tree} & \Rightarrow \text{emp} : \{ x = \text{nil} \} \\
\text{tree} & \Rightarrow \exists y, z . \ x \mapsto y, z \ast \text{tree} y \ast \text{tree} z
\end{align*}
\]
Example: Destruction of a Binary Tree

Example

\[
\begin{align*}
\text{tree} & \Rightarrow \text{emp}: \{ x = \text{nil} \} \\
\text{tree} & \Rightarrow \exists y, z . \ x \mapsto y, z * \text{tree} y * \text{tree} z
\end{align*}
\]

cleanup(x) {
    if (x != 0) {
        l := x.left;
        r := x.right;
        cleanup(x.left);
        cleanup(x.right);
        free(x);
    }
}
Example: Destruction of a Binary Tree

Example

\[
\begin{align*}
tree & \Rightarrow \text{emp}: \{ x = \text{nil} \} \\
tree & \Rightarrow \exists y, z. \ x \mapsto y, z \cdot tree y \cdot tree z
\end{align*}
\]

cleanup(x) { 
    \{ tree x \}
    if (x != 0) {
        l := x.left;
        r := x.right;
        cleanup(x.left);
        cleanup(x.right);
        free(x);
    }
} 
\{ emp \}
Example: Destruction of a Binary Tree

Example

\[ \text{tree} \Rightarrow \text{emp} : \{ x = \text{nil} \} \]
\[ \text{tree} \Rightarrow \exists y, z . \ x \mapsto y, z * \text{tree y} * \text{tree z} \]

```
cleanup(x) {
    {tree x}
    if (x != 0) {
        l := x.left;
        r := x.right;
        cleanup(x.left);
        cleanup(x.right);
        free(x);
    }
} {emp}
```
Example: Destruction of a Binary Tree

Example

\[
\begin{align*}
\text{tree} & \Rightarrow \text{emp}: \{ x = \text{nil} \} \\
\text{tree} & \Rightarrow \exists y, z . \ x \mapsto y, z \ast \text{tree} y \ast \text{tree} z
\end{align*}
\]

```c
cleanup(x) {
    if (x != 0) {
        l := x.left;
        r := x.right;
        cleanup(x.left);
        cleanup(x.right);
        free(x);
    }
}
```

{\text{tree} x}
Example: Destruction of a Binary Tree

Example

\[
\begin{align*}
  \text{tree} & \Rightarrow \text{emp} : \{ x = \text{nil} \} \\
  \text{tree} & \Rightarrow \exists y, z . \ x \mapsto y, \ z \ast \text{tree} y \ast \text{tree} z
\end{align*}
\]

\[
\begin{align*}
\text{cleanup} (x) & \{ \text{tree} x \} \\
& \{ \exists y, z . \ x \mapsto y, \ z \ast \text{tree} y \ast \text{tree} z \}
\end{align*}
\]

\[
\begin{align*}
\text{cleanup} (x) & \{ \text{tree} x \} \\
& \{ \exists y, z . \ x \mapsto y, \ z \ast \text{tree} y \ast \text{tree} z \}
\end{align*}
\]

\[
\begin{align*}
\text{cleanup} (x) & \{ \text{tree} x \} \\
& \{ \exists y, z . \ x \mapsto y, \ z \ast \text{tree} y \ast \text{tree} z \}
\end{align*}
\]
Example: Destruction of a Binary Tree

Example

tree ⇒ emp: \{ x = nil \}

\[ tree ⇒ \exists y, z . \; x \mapsto y, z * \; \text{tree} \; y * \; \text{tree} \; z \]

cleanup(x) {
    if (x != 0) {
        l := x.left;
        r := x.right;
        cleanup(x.left);
        cleanup(x.right);
        free(x);
    }
}

\{ tree \; x \} \quad \{ \exists y, z . \; x \mapsto y, z * \; \text{tree} \; y * \; \text{tree} \; z \} \quad \{ \exists z . \; x \mapsto (l, z) * \; \text{tree} \; l * \; \text{tree} \; z \} \quad \{ emp \}
Example: Destruction of a Binary Tree

tree ⇒ emp: \{x = \text{nil}\}

\[
\begin{align*}
tree & \Rightarrow \exists y, z . \ x \mapsto y, z \ast \text{tree } y \ast \text{tree } z
\end{align*}
\]

cleanup(x) {
    if (x != 0) {
        l := x.left;
        r := x.right;
        cleanup(x.left);
        cleanup(x.right);
        free(x);
    }
}

\[
\begin{align*}
\text{cleanup}(x) & \quad \{\text{tree } x\}
\text{cleanup}(x) & \quad \{\exists y, z . \ x \mapsto y, z \ast \text{tree } y \ast \text{tree } z\}
\text{cleanup}(x) & \quad \{\exists z . \ x \mapsto (l, z) \ast \text{tree } l \ast \text{tree } z\}
\text{cleanup}(x) & \quad \{x \mapsto l, r \ast \text{tree } l \ast \text{tree } r\}
\text{cleanup}(x) & \quad \{\text{emp}\}
\end{align*}
\]
Example: Destruction of a Binary Tree

Example

\[
\begin{align*}
\text{tree} & \Rightarrow \text{emp}: \{x = \text{nil}\} \\
\text{tree} & \Rightarrow \exists y, z. \ x \mapsto y, z \ast \text{tree} y \ast \text{tree} z
\end{align*}
\]

cleanup(x) {
    if (x != 0) {
        l := x.left;
        r := x.right;
        cleanup(x.left);
        cleanup(x.right);
        free(x);
    }
}

\{tree x\}
\{\exists y, z. \ x \mapsto y, z \ast \text{tree} y \ast \text{tree} z\}
\{\exists z. \ x \mapsto (l, z) \ast \text{tree} l \ast \text{tree} z\}
\{x \mapsto l, r \ast \text{tree} l \ast \text{tree} r\}
\{x \mapsto l, r \ast \text{emp} \ast \text{tree} r\}
\{\text{emp}\}
Example: Destruction of a Binary Tree

Example

tree ⇒ emp: \{x = nil\}

tree ⇒ ∃y, z . x ↦ y, z * tree y * tree z

cleanup(x) {
    if (x != 0) {
        l := x.left;
        r := x.right;
        cleanup(x.left);
        cleanup(x.right);
        free(x);
    }
}

{tree x}

{∃y, z . x ↦ y, z * tree y * tree z}

{∃z . x ↦ (l, z) * tree l * tree z}

{x ↦ l, r * tree l * tree r}

{x ↦ l, r * emp * tree r}

{x ↦ l, r * emp * emp}

{emp}
Example: Destruction of a Binary Tree

Example

\[
tree \Rightarrow \text{emp} : \{ x = \text{nil} \}
\]
\[
tree \Rightarrow \exists y, z . \ x \mapsto y, z * \text{tree} \ y * \text{tree} \ z
\]

\[
\text{cleanup(x) \{ }
\]
\[
\text{if (x \neq 0) \{ }
\]
\[
l := x.\text{left};
\]
\[
r := x.\text{right};
\]
\[
\text{cleanup(x.left);}
\]
\[
\text{cleanup(x.right);}
\]
\[
\text{free(x);} \}
\]
\[
\}
\]
Example: Destruction of a Binary Tree

Example

\[ \text{tree} \Rightarrow \text{emp} : \{ x = \text{nil} \} \]
\[ \text{tree} \Rightarrow \exists y, z . \ x \mapsto y, z * \text{tree} y * \text{tree} z \]

```plaintext
cleanup(x) {
    if (x != 0) {
        l := x.left;
        r := x.right;
        cleanup(x.left);
        cleanup(x.right);
        free(x);
    }
}
```

\[ \{ \text{tree} x \} \]
\[ \{ \exists y, z . \ x \mapsto y, z * \text{tree} y * \text{tree} z \} \]
\[ \{ \exists z . \ x \mapsto (l, z) * \text{tree} l * \text{tree} z \} \]
\[ \{ x \mapsto l, r * \text{tree} l * \text{tree} r \} \]
\[ \{ x \mapsto l, r * \text{emp} * \text{tree} r \} \]
\[ \{ x \mapsto l, r * \text{emp} * \text{emp} \} \]
\[ \{ \text{emp} * \text{emp} * \text{emp} \} \]
\[ \{ \text{emp} \} \]
\[ \{ \text{emp} \} \]
### Semantics

States: \( \sigma = s, h \), stack \( s : \text{Var} \rightarrow \mathbb{Z} \), heap \( h : \mathbb{Z} \rightarrow \mathbb{Z}^+ \)

\[
\Sigma ::= \text{emp} \mid x \mapsto a \mid Pa \mid \Sigma \ast \Sigma
\]

\( \pi ::= a = b \mid a \neq b \)

#### Semantics of Separation Logic

- \( s, h \models \text{emp} \iff \text{dom } h = \emptyset \)
- \( s, h \models x \mapsto a \iff \text{dom } h = \{sx\}, \ h \circ sx = sa \)
- \( s, h \models Px \) via unfolding trees (next slide)
- \( s, h \models \Sigma_1 \ast \Sigma_2 \iff h = h_1 \cup h_2 \) and \( s, h_1 \models \Sigma_1 \) and \( s, h_2 \models \Sigma_2 \)
  
  (\( h \) is partitioned into \( h_1, h_2 \))
- \( s, h \models a \sim b \iff sa \sim sb \)
- \( s, h \models \exists z : \Sigma : \Pi \iff \exists a \in \mathbb{Z}^+ . s[z \mapsto a], h \models \Sigma \)
  and \( \forall \pi \in \Pi . s[z \mapsto a], h \models \pi \)
Unfolding Trees

An unfolding tree $t$ captures an unrolling of predicates in a symbolic heap:

$$Tx \ast R$$

$$\exists y, z \cdot x \mapsto y, z \ast \text{tree } y \ast \text{tree } z$$

$$\exists y, z \cdot x \mapsto y, z \ast \text{tree } y \ast \text{tree } z \quad \text{emp: } \{x = \text{nil}\}$$

$$\text{emp: } \{x = \text{nil}\} \quad \text{emp: } \{x = \text{nil}\}$$
**Unfolding Trees**

An *unfolding tree* $t$ captures an unrolling of predicates in a symbolic heap:

\[
Tx * R \\
\downarrow \\
\exists y, z . x \mapsto y, z * \textit{tree } y * \textit{tree } z
\]

Corresponding unfolding $\llbracket t \rrbracket$:

\[
\exists z . x \mapsto z_1, z_2 * z_1 \mapsto z_3, z_4 * \text{emp } * \text{emp } * \text{emp } * R : \{z_2 = z_3 = z_4 = \text{nil}\}
\]

\[
\equiv \exists z . x \mapsto z_1, \text{nil } * z_1 \mapsto \text{emp }, \text{emp } * R
\]
Unfolding Trees

An unfolding tree $t$ captures an unrolling of predicates in a symbolic heap:

$$T_x \ast R$$

\[\exists y, z. x \mapsto y, z \ast \text{tree } y \ast \text{tree } z\]

Corresponding unfolding $[t]$:

$$\exists z. x \mapsto z_1, z_2 \ast z_1 \mapsto z_3, z_4 \ast \text{emp} \ast \text{emp} \ast \text{emp} \ast R : \{z_2 = z_3 = z_4 = \text{nil}\}$$

$$\equiv \exists z. x \mapsto z_1, \text{nil} \ast z_1 \mapsto \text{emp}, \text{emp} \ast R$$

$$s, h \models_{\Phi} P_x \iff \exists \text{ unfolding tree } t \text{ of } P. s, h \models [t]$$
Outline

1. Motivation

2. Symbolic Heaps with Recursive Definitions

3. A Refinement Theorem

4. Applications

5. Conclusion
Refinement – Overview

Merriam Webster on Refinement:
the act or process of removing unwanted substances from something
Refinement – Overview

Merriam Webster on Refinement:
the act or process of removing unwanted substances from something

Given:
- System of recursive definitions $\Phi$
- Symbolic heap $\phi x$
- Set of fully unfolded symbolic heaps $\mathcal{H} \subseteq TSLF$
Refinement – Overview

Merriam Webster on Refinement:
the act or process of removing unwanted substances from something

Given:
- System of recursive definitions $\Phi$
- Symbolic heap $\varphi x$
- Set of fully unfolded symbolic heaps $\mathcal{H} \subseteq TSLF$

Tasks:
- Provide $\Psi$ and $\psi x$ capturing exactly the unfoldings of $\Phi$ and $\varphi x$ in $\mathcal{H}$.
- Decide whether each unfolding of $\varphi x$ is in $\mathcal{H}$.
- Decide whether there exists an unfolding of $\varphi x$ in $\mathcal{H}$.
A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists z\. \Sigma : \Pi$ equal to nil?
A Toy Problem

Is some variable of symbolic heap \( \varphi \triangleq \exists z . \Sigma : \Pi \) equal to \( \text{nil} \)?

\[ \exists y, z . \ x \mapsto y, z \]
A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists z . \Sigma : \Pi$ equal to nil?

\[ \exists y, z . x \mapsto y, z \]

no
A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists z . \Sigma : \Pi$ equal to nil?

\[
\text{no} \quad \exists y, z . x \mapsto y, z \quad \ast \quad \text{tree } y \ast \quad \text{tree } z
\]
A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists z . \sum : \Pi$ equal to nil?

```
no

no

no

\exists y, z . x \mapsto y, z \ast \text{tree } y \ast \text{tree } z

no

no
```
Is some variable of symbolic heap $\varphi \triangleq \exists z . \Sigma : \Pi$ equal to nil?

- yes
- $\exists y, z . x \mapsto y, z \ast \text{tree } y \ast \text{tree } z$

- no

This is a transition relation of a finite tree automaton. Naive decision procedure:
1. If $x = \text{nil}$ then return yes.
2. Compute least set $Y$ of predicate symbols yielding yes.
3. Check whether $\text{Pred}(\varphi) \backslash Y \neq \emptyset$.

Refinement: replace each predicate call $P$ by $(P; \text{yes})$ and $(P; \text{no})$ according to observation from above.
**A Toy Problem**

Is some variable of symbolic heap $\varphi \triangleq \exists z \cdot \Sigma : \Pi$ equal to nil?

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```
A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists z . \Sigma : \Pi$ equal to nil?

\[
\begin{align*}
\text{yes} & \\
\uparrow & \\
\text{no} & \quad \exists y, z . x \leftrightarrow y, z \quad \ast \text{tree } y \ast \text{tree } z
\end{align*}
\]

\[
\begin{align*}
\text{yes} & \\
\uparrow & \\
\text{yes} & \\
\text{yes} & \\
\end{align*}
\]
A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists z. \Sigma : \Pi$ equal to nil?

\[
\begin{array}{c}
\text{yes} \\
\uparrow \\
\text{no} \\
\exists y, z. x \mapsto y, z \quad \ast \quad \text{tree } y \ast \ast \text{tree } z
\end{array}
\]

This is a transition relation of a finite tree automaton.
A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists z . \Sigma : \Pi$ equal to nil?

This is a transition relation of a finite tree automaton.

Naive decision procedure:

1. If $x = \text{nil} \in \Pi$ then return yes.
2. Compute least set $\mathcal{Y}$ of predicate symbols yielding yes.
3. Check whether $Pred(\varphi) \cap \mathcal{Y} \neq \emptyset$. 
A Toy Problem

Is some variable of symbolic heap \( \varphi \triangleq \exists z . \Sigma : \Pi \) equal to nil?

\[
\begin{array}{c}
\text{yes} \\
\uparrow \\
\text{no} \\
\exists y, z . \ x \mapsto y, z \ * \ tree y \ * \ tree z \\
\uparrow \\
\text{yes} \quad \text{yes}
\end{array}
\]

This is a transition relation of a finite tree automaton.

Naive decision procedure:

1. If \( x = \text{nil} \in \Pi \) then return yes.
2. Compute least set \( \mathcal{Y} \) of predicate symbols yielding yes.
3. Check whether \( \text{Pred}(\varphi) \cap \mathcal{Y} \neq \emptyset \).

Refinement: replace each predicate call \( P \) by \( (P, \text{yes}) \) and \( (P, \text{no}) \) according to observation from above.
Compositional Heap Properties

$\mathcal{H} \subseteq TSLF$  \hspace{2em} $\mathbb{H}$ finite set  \hspace{2em} $\mathcal{H}_H \subseteq H \times TSLF$  \hspace{2em} $\Delta[\Phi] \subseteq H \times SLF(\Phi) \times H^*$

$(\mathcal{H}_H, \Delta[\Phi])$ is compositional if for each unfolding tree $t$ of each symbolic heap $\varphi x$:

$(p, [t]) \in \mathcal{H}_H$ if and only if $(p, \varphi x, q) \in \Delta[\Phi]$ and $(q_i, [t_i]) \in \mathcal{H}_H$

(where $1 \leq i \leq |Calls(\varphi x)|$).
Compositional Heap Properties

\( \mathcal{H} \subseteq TSLF \) \( \mathcal{H} \) finite set \( \mathcal{H}_H \subseteq \mathcal{H} \times TSLF \) \( \Delta[\Phi] \subseteq \mathcal{H} \times SLF(\Phi) \times \mathcal{H}^* \)

(\( \mathcal{H}_H, \Delta[\Phi] \)) is compositional if for each unfolding tree \( t \) of each symbolic heap \( \varphi x \):

\((p, [t]) \in \mathcal{H}_H \) if and only if \((p, \varphi x, q) \in \Delta[\Phi] \) and \((q_i, [t_i]) \in \mathcal{H}_H \)

(where \( 1 \leq i \leq |Calls(\varphi x)| \)).

\( \mathcal{H}_H \) is compositional if for each \( \Phi \) there exists \( \Delta[\Phi] \) such that (\( \mathcal{H}_H, \Delta[\Phi] \)) is compositional.
Compositional Heap Properties

\( \mathcal{H} \subseteq TSLF \quad \mathbb{H} \) finite set \quad \mathcal{H}_{\mathbb{H}} \subseteq \mathbb{H} \times TSLF \quad \Delta[\Phi] \subseteq \mathbb{H} \times SLF(\Phi) \times \mathbb{H}^* \\

(\mathcal{H}_{\mathbb{H}}, \Delta[\Phi]) \) is compositional if for each unfolding tree \( t \) of each symbolic heap \( \varphi x \):

\((p, [t]) \in \mathcal{H}_{\mathbb{H}} \) if and only if \((p, \varphi x, q) \in \Delta[\Phi] \) and \((q_i, [t_i]) \in \mathcal{H}_{\mathbb{H}} \)

(where \( 1 \leq i \leq |Calls(\varphi x)| \)).

\( \mathcal{H}_{\mathbb{H}} \) is compositional if for each \( \Phi \) there exists \( \Delta[\Phi] \) such that \((\mathcal{H}_{\mathbb{H}}, \Delta[\Phi]) \) is compositional.

\( \mathcal{H} \) is a compositional compositional heap property if there exists a compositional \( \mathcal{H}_{\mathbb{H}} \) and \( \mathbb{H}_0 \subseteq \mathbb{H} \) such that

\( \varphi \in \mathcal{H} \) if and only if \( \exists h \in \mathbb{H}_0 \ . \ (h, \varphi) \in \mathcal{H}_{\mathbb{H}} \).
Is some variable of a symbolic heap equal to nil?

\[ H \triangleq \{ \exists z. \Sigma : \Pi \in TSLF \mid \exists x \in Var. x = \text{nil} \in \Pi \} \]
Compositional Heap Properties – Toy Example

$\mathcal{H} \subseteq TSLF$  \quad $\mathbb{H}$ finite set  \quad $\mathcal{H}_\mathbb{H} \subseteq \mathbb{H} \times TSLF$  \quad $\Delta[\Phi] \subseteq \mathbb{H} \times SLF(\Phi) \times \mathbb{H}^*$

Is some variable of a symbolic heap equal to nil?

$$\mathcal{H} \triangleq \{ \exists z \cdot \Sigma : \Pi \in TSLF \mid \exists x \in \text{Var}. \ x = \text{nil} \in \Pi \}$$

$\mathcal{H}$ is a compositional heap property:

- $\mathbb{H} \triangleq \{0, 1\}$, $\mathbb{H}_0 \triangleq \{1\}$

- $(q, \varphi) \in \mathcal{H}_{\mathbb{H}}$ iff $(q = 0 \land \varphi \notin \mathcal{P}) \lor (q = 1 \land \varphi \in \mathcal{P})$

- $(p, \exists z \cdot \Sigma : \Pi, q_1 \ldots q_n) \in \Delta[\Phi]$ iff $p = \max\{[\exists x \in \text{Var}. \ x = \text{nil} \in \Pi], q_1, \ldots, q_n\}$
A Refinement Theorem

\[ \mathcal{H} \subseteq TSLF \quad \mathbb{H} \text{ finite set} \quad \mathcal{H}_H \subseteq \mathbb{H} \times TSLF \quad \Delta[\Phi] \subseteq \mathbb{H} \times SLF(\Phi) \times \mathbb{H}^* \]

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
</table>
| For each compositional heap property \( \mathcal{H} \), \( \varphi \in SLF \) and \( \Phi \in SRD \) one can effectively construct \( \Psi \in SRD \) and \( \psi \in SLF \) such that for each unfolding tree \( t \) of \( \varphi \) in \( \Phi \):
| \( [t] \in \mathcal{H} \iff \exists \text{ unfolding tree } t' \text{ of } \psi \text{ in } \Psi \text{ such that } [t] \equiv [t'] \). |
| Moreover, if \( cw \) is the maximal number of predicate calls:
| \[ |\Psi| + |\psi| \leq H^{cw+1} \cdot (|\Phi| + |\varphi|) + |H_0| \cdot |\varphi| \] |
A Refinement Theorem

\[ \mathcal{H} \subseteq TSLF \quad \mathbb{H} \text{ finite set} \quad \mathcal{H}_\mathbb{H} \subseteq \mathbb{H} \times TSLF \quad \Delta[\Phi] \subseteq \mathbb{H} \times SLF(\Phi) \times \mathbb{H}^* \]

Theorem

For each compositional heap property \( \mathcal{H} \), \( \varphi \in SLF \) and \( \Phi \in SRD \) one can effectively construct \( \Psi \in SRD \) and \( \psi \in SLF \) such that for each unfolding tree \( t \) of \( \varphi \) in \( \Phi \):

\[ \llbracket t \rrbracket \in \mathcal{H} \text{ iff } \exists \text{ unfolding tree } t' \text{ of } \psi \text{ in } \Psi \text{ such that } \llbracket t \rrbracket \equiv \llbracket t' \rrbracket. \]

Moreover, if \( cw \) is the maximal number of predicate calls:

\[ |\Psi| + |\psi| \leq \mathbb{H}^{cw+1} \cdot (|\Phi| + |\varphi|) + |\mathbb{H}_0| \cdot |\varphi| \]

The refinement of \( \Phi \) by \( \mathcal{H}_\mathbb{H} \) is the least set \( \Phi \upharpoonright \mathcal{H}_Q \) satisfying

\[ P \implies \varphi \in \Phi \text{ and } (p, \varphi, q) \in \Delta[\Phi] \text{ implies } (P, p) \implies \varphi[q] \]

where each predicate call \( P_i z \) is replaced by \( (P_i, q_i) z \) in \( \varphi[q] \).
A Refinement Theorem

Theorem

Compositional heap properties are closed under Boolean operations.

Theorem

For each compositional heap property $\mathcal{H}$, $\varphi \in SLF$ and $\Phi \in SRD$ it is decidable whether there exists an unfolding tree $t$ of $\varphi$ in $\Phi$ such that $[t] \in \mathcal{H}$.

Complexity of deciding $\varphi \in \mathcal{H}$:

$$O \left( \left[ H^{cw+1} \cdot (|\Phi| + |\varphi|) + H_0 \cdot |\varphi| \right] \cdot T(\Delta[\Phi]) \right)$$
Outline

1. Motivation

2. Symbolic Heaps with Recursive Definitions

3. A Refinement Theorem

4. Applications

5. Conclusion
Satisifiability

SL−SAT: Given $\Phi \in SRD$ and $\varphi \in SLF$, decide whether $\varphi$ is satisfiable.

$$\mathcal{S}_k \triangleq \{ \varphi x \in TSLF_k \mid \exists s, h . \ s, h \models \varphi x \}$$

where $k$ bounds the number of free variables.
**Satisfiability**

SL–SAT: Given $\Phi \in SRD$ and $\varphi \in SLF$, decide whether $\varphi$ is satisfiable.

$$
\mathcal{S}_k \triangleq \{ \varphi x \in TSLF_k \mid \exists s, h. \ s, h \models \varphi x \}
$$

where $k$ bounds the number of free variables.

**Theorem**

$\mathcal{S}_k$ is a compositional heap property. Moreover, $\mathcal{S}$ can be chosen such that $\mathcal{S} \in \mathcal{O}(2^{k^2})$. 

Theorem

- $\text{SL-SAT}$ is ExpTime–complete (Brotherstone, 2013)
- $\text{SL-SAT}_k$ is NP–complete for $k \geq 3$ (Brotherstone, 2013)
- $\text{SL-SAT}_k; c_{w}$ is PTime–complete for $c_{w} \leq 2$, $k \leq 0$
Satisfiability

SL-SAT: Given $\Phi \in SRD$ and $\varphi \in SLF$, decide whether $\varphi$ is satisfiable.

$$G_k \triangleq \{ \varphi x \in TSLF_k \mid \exists s, h. s, h \models \varphi x \}$$

where $k$ bounds the number of free variables.

**Theorem**

$G_k$ is a compositional heap property. Moreover, $G$ can be chosen such that $G \in O(2^{k^2})$.

**Theorem**

- SL-SAT is ExpTime-complete (Brotherstone, 2013)
- SL-SAT$_k$ is NP-complete for $k \geq 3$ (Brotherstone, 2013)
- SL-SAT$_{k,cw}$ is PTime-complete for $cw \geq 2$, $k \geq 0$
Model-Checking

SL-MC: Given $\Phi \in SRD$, $\varphi \in SLF$ and $s, h$, decide whether $s, h \models \varphi$.

$$M_k \triangleq \{ \varphi x \in TSLF_k \mid s, h \models \varphi x \}$$

where $k$ bounds the number of free variables.

**Theorem**

$M_k$ is a compositional heap property.

**Theorem**

- SL-MC is $\text{ExpTime}$–complete (Brotherstone, 2016)
- SL-MC$_k$ is $\text{NP}$–complete for $k \geq 3$ (Brotherstone, 2016)
- SL-MC$_{k,cw}$ is $\text{PTime}$–complete for $cw \geq 2$, $k \geq 0$
Establishment

Establishment: Given $\Phi \in SRD$, decide whether each existentially quantified variable in each unfolding is eventually allocated.

$$\mathcal{E}_k \triangleq \{ \varphi \in TSLF_k \mid \forall y \in BV(\varphi) . y \text{ allocated} \}$$

where $k$ bounds the number of free variables.

Theorem

$\mathcal{E}_k$ is a compositional heap property.

Theorem

- Establishment is in $\mathsf{ExpTime}$
- Establishment$_k$ is in $\mathsf{NP}$ for $k \geq 3$
- Establishment$_{k,cw}$ is $\mathsf{PTIME}$–complete for $cw \geq 2$, $k \geq 0$
Equality of Program Variables

EqVar: Given $\Phi \in SRD$, $\varphi x \in SLF$ and $i, j$, decide whether $x_i = x_j$ holds for some unfolding.

$$\mathcal{V}_k \triangleq \{ \varphi \in TSLF_k \mid x_i = x_j \in \Pi^*(\varphi) \text{ and } x_i \neq x_j \notin \Pi^*(\varphi) \}$$

where $k$ bounds the number of free variables.

**Theorem**

$\mathcal{V}_k$ is a compositional heap property.

**Theorem**

- EqVar is in ExpTime–complete.
- EqVar$_k$ is in NP–complete for $k \geq 3$
- EqVar$_{k,cw}$ is PTime–complete for $cw \geq 2$, $k \geq 0$
Reachability

Reachability: Given $\Phi \in SRD$, $\varphi x \in SLF$ and $i,j$, decide whether $x_i \leadsto x_j$ holds for some unfolding.

$$\mathcal{R}_k \triangleq \{ \varphi \in TSLF_k \mid x_i \leadsto x_j \}$$

where $k$ bounds the number of free variables.

**Theorem**

$\mathcal{R}_k$ is a compositional heap property.

**Theorem**

- Reachability is in $\text{EXPTime}$–complete.
- Reachability$_k$ is in $\text{NP}$–complete for $k \geq 3$
- Reachability$_{k,cw}$ is $\text{PTIME}$–complete for $cw \geq 2$, $k \geq 0$
Entailment

Entailment: Given $\Phi \in SRD$, $\varphi, \psi \in SLF$, decide whether

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

Entailment: Given $\Phi \in SRD$, $\varphi, \psi \in SLF$, decide whether

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

- Entailment is undecidable.
- Most tools use tailored procedures for fixed systems of recursive definitions.
Entailment

Entailment: Given $\Phi \in SRD$, $\varphi, \psi \in SLF$, decide whether

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

- Entailment is undecidable.
- Most tools use tailored procedures for fixed systems of recursive definitions.

**Definition**

$\varphi, \psi \in TSLF$ are $\mathcal{H}$–congruent if for each $\vartheta \in SLF$ with one predicate call $P$,

$$\vartheta[\varphi / P] \in \mathcal{H} \text{ iff } \vartheta[\psi / P] \in \mathcal{H}.$$
Entailment

Entailment: Given $\Phi \in SRD$, $\varphi, \psi \in SLF$, decide whether

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

- Entailment is undecidable.
- Most tools use tailored procedures for fixed systems of recursive definitions.

**Definition**

$\varphi, \psi \in TSLF$ are $\mathcal{H}$–congruent if for each $\vartheta \in SLF$ with one predicate call $P$,

$$\vartheta[\varphi / P] \in \mathcal{H} \text{ iff } \vartheta[\psi / P] \in \mathcal{H}.$$ 

**Theorem**

$\mathcal{H}$ is compositional if the $\mathcal{H}$–congruence is of finite index.
Entailment

Entailment: Given $\Phi \in SRD$, $\varphi, \psi \in SLF$, decide whether
\[
\forall s, h . \ s, h \models_\Phi \varphi \implies s, h \models_\Phi \psi.
\]

- Entailment is undecidable.
- Most tools use tailored procedures for fixed systems of recursive definitions.

**Definition**

$\varphi, \psi \in TSLF$ are $\mathcal{H}$–congruent if for each $\vartheta \in SLF$ with one predicate call $P$,
\[
\vartheta[\varphi / P] \in \mathcal{H} \iff \vartheta[\psi / P] \in \mathcal{H}.
\]

**Theorem**

$\mathcal{H}$ is compositional if the $\mathcal{H}$–congruence is of finite index.

We obtain a finite index for formulas in SL–fragments where the entailment problem is known to be decidable.
Entailment

Entailment: Given $\Phi \in SRD$, $\varphi, \psi \in SLF$, decide whether

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

- Entailment is undecidable.  
- Most tools use tailored procedures for fixed systems of recursive definitions.

**Definition**

$\varphi, \psi \in TSLF$ are $\mathcal{H}$–congruent if for each $\vartheta \in SLF$ with one predicate call $P$,

$$\vartheta[\varphi / P] \in \mathcal{H} \iff \vartheta[\psi / P] \in \mathcal{H}.$$ 

**Theorem**

$\mathcal{H}$ is compositional if the $\mathcal{H}$–congruence is of finite index.

We obtain a finite index for formulas in SL–fragments where the entailment problem is known to be decidable.

**Open problem:** How to compute equivalence classes of $\mathcal{H}$–congruences for restricted fragments?
Conclusion

1. Symbolic Heaps with Recursive Definitions

\[ \varphi x \triangleq \exists z . \Sigma : \Pi, \quad \Phi \triangleq \{ P \Rightarrow \varphi x | \ldots \}, \quad \text{fragment used by most tools} \]

2. A Refinement Theorem

Refine \( \Phi \) and \( \varphi x \) such that each unfolding satisfies property \( \mathcal{H} \).

3. Applications

Satisfiability, Model-Checking, Establishment, Reachability, \ldots are compositional

4. Future Work

Apply compositional heap properties to the entailment problem for restricted fragments