Compositional Refinement of Separation Logic with Recursive Definitions

Christina Jansen **Christoph Matheja** **Thomas Noll**

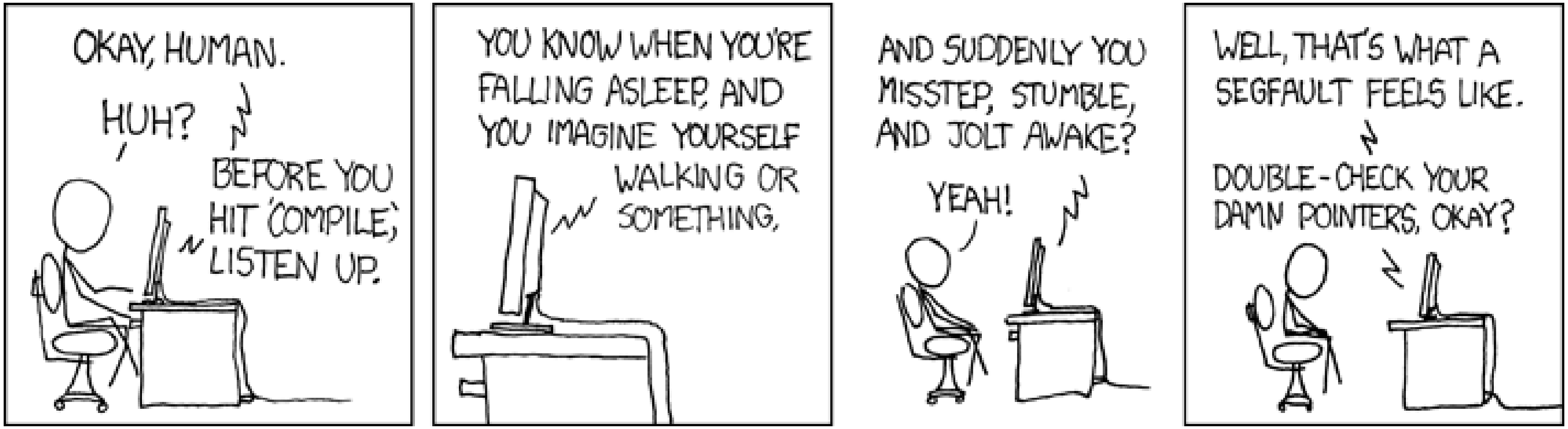
Software Modeling and Verification Group

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

DCON 2016

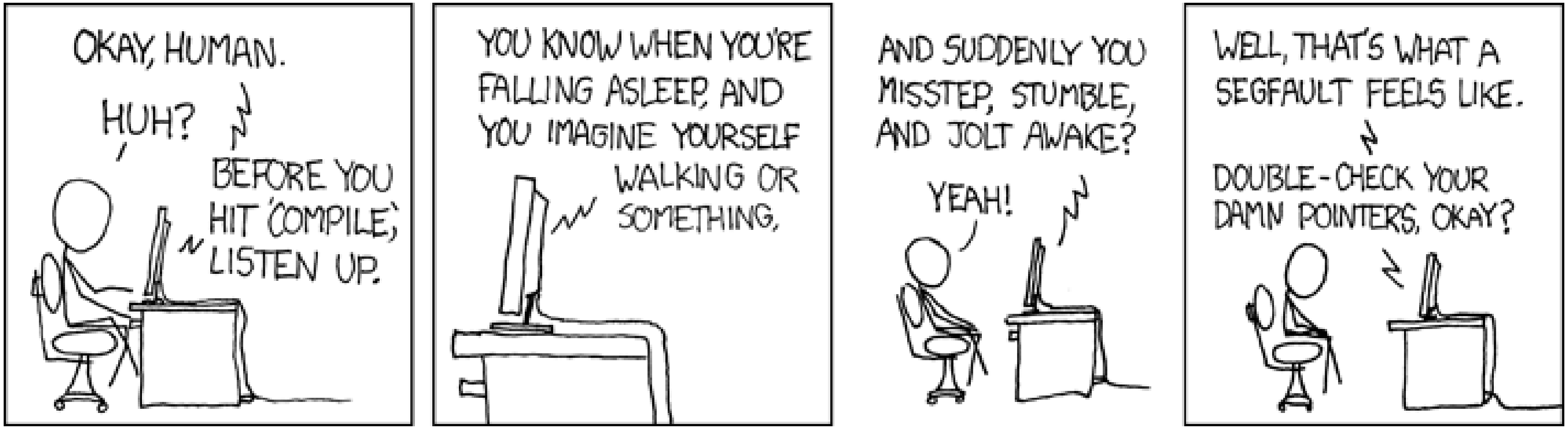
March 7, 2016; Saarbrücken

Motivation



https://xkcd.com/371

Motivation

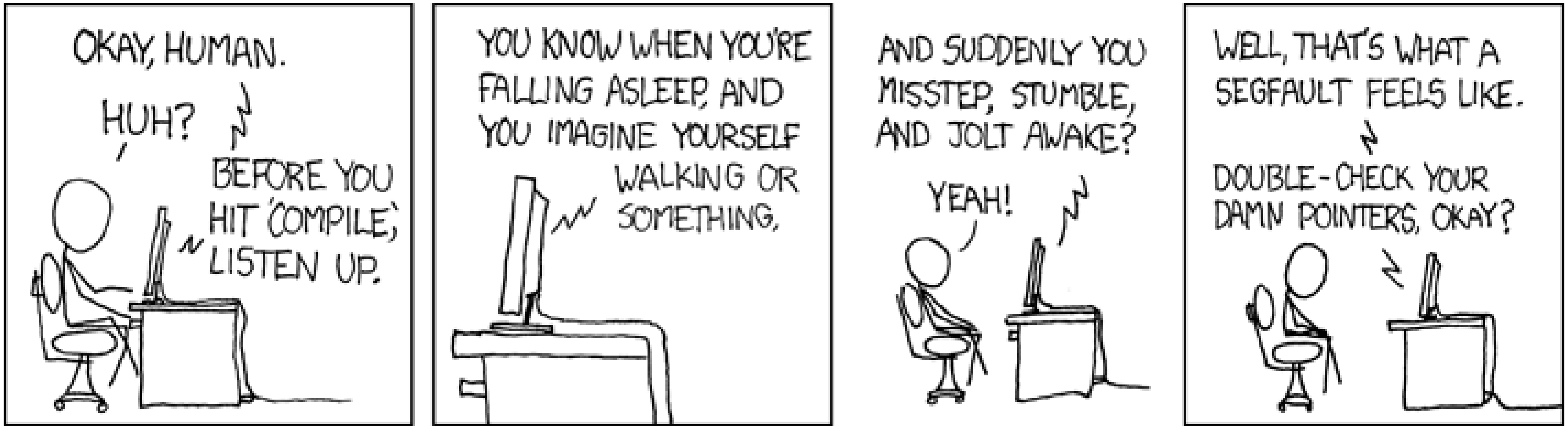


https://xkcd.com/371

Separation Logic

Hoare Logic
+ Pointers
+ Dynamic data structures
+ Local reasoning

Motivation

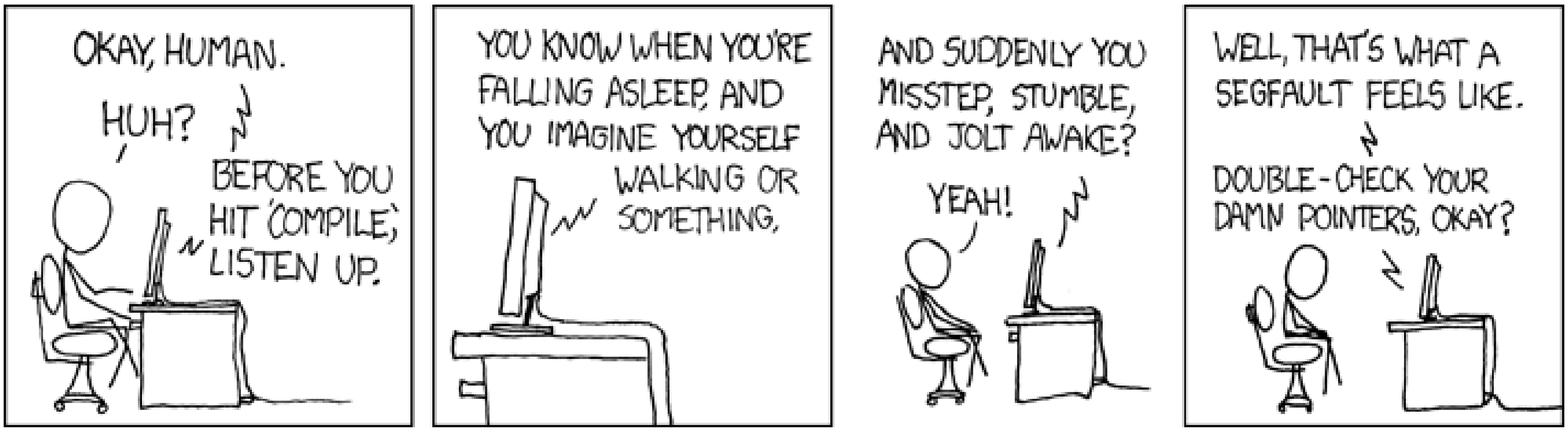


https://xkcd.com/371

Concurrent Separation Logic

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

Motivation



https://xkcd.com/371

Concurrent Separation Logic

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

Some Informal Properties

- Satisfiability: there exists a model of φ

Some Informal Properties

- Satisfiability: there exists a model of φ
- Model-Checking: h is a model of φ

Some Informal Properties

- Satisfiability: there exists a model of φ
- Model-Checking: h is a model of φ
- Entailment: each model of φ is a model of ψ

Some Informal Properties

- Satisfiability: there exists a model of φ
- Model-Checking: h is a model of φ
- Entailment: each model of φ is a model of ψ
- Establishment: each variable in φ is eventually allocated

Some Informal Properties

- Satisfiability: there exists a model of φ
- Model-Checking: h is a model of φ
- Entailment: each model of φ is a model of ψ
- Establishment: each variable in φ is eventually allocated
- Equality: x, y are guaranteed to be equal in φ

Some Informal Properties

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

Some Informal Properties

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

Some Informal Properties

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

How to decide each of these properties?

Some Informal Properties

- Satisfiability: there exists a model of φ
- Model-Checking: h is a model of φ
- Entailment: each model of φ is a model of ψ
- Establishment: each variable in φ is eventually allocated
- Equality: x, y are guaranteed to be equal in φ
- Reachability: y is reachable from x in each heap specified by φ
- Connectivity: each heap specified by φ 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 $s : Var \dashrightarrow \mathbb{Z}$, heap $h : \mathbb{Z} \dashrightarrow \mathbb{Z}^+$

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

Spatial formulas Σ and pure formulas π are given by

$$\begin{aligned} \Sigma &::= \mathtt{emp} \mid x \mapsto \mathbf{a} \mid P\mathbf{a} \mid \Sigma * \Sigma \\ \pi &::= \mathbf{a} = \mathbf{b} \mid \mathbf{a} \neq \mathbf{b} \end{aligned}$$

where P is a predicate symbol and $\mathbf{a}, \mathbf{b}$ are tuples of terms.

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

Symbolic Heaps with Recursive Definitions

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

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

Spatial formulas Σ and pure formulas π are given by

$$\begin{array}{rcl} \Sigma & ::= & \mathtt{emp} \mid x \mapsto \mathbf{a} \mid P\mathbf{a} \mid \Sigma * \Sigma \\ \pi & ::= & \mathbf{a} = \mathbf{b} \mid \mathbf{a} \neq \mathbf{b} \end{array}$$

where P is a predicate symbol and $\mathbf{a}, \mathbf{b}$ are tuples of terms.

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

System of recursive definitions: finite set Φ of rules $P \Rightarrow \varphi\mathbf{x}$

Example

$$\begin{array}{l} tree \Rightarrow \mathtt{emp} : \{x = \mathtt{nil}\} \\ tree \Rightarrow \exists y, z \,.\, x \mapsto y, z * tree\, y * tree\, z \end{array}$$

Example: Destruction of a Binary Tree

Example

$$\mathit{tree} \Rightarrow \texttt{emp} : \{x = \texttt{nil}\}$$
$$\mathit{tree} \Rightarrow \exists y, z \;.\; x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z$$

Example: Destruction of a Binary Tree

Example

$$tree \Rightarrow \mathtt{emp} : \{x = \mathtt{nil}\}$$
$$tree \Rightarrow \exists y, z \;.\; x \mapsto y, z * tree\, y * tree\, z$$

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

Example: Destruction of a Binary Tree

Example

$$\mathit{tree} \Rightarrow \mathtt{emp} : \{x = \mathtt{nil}\}$$
$$\mathit{tree} \Rightarrow \exists y, z \,.\, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z$$

```
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

$$\mathit{tree} \Rightarrow \mathtt{emp} : \{x = \mathtt{nil}\}$$
$$\mathit{tree} \Rightarrow \exists y, z \,.\, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z$$

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

$\{\mathit{tree}\, x\}$

$\{\mathit{emp}\}$

Example: Destruction of a Binary Tree

Example

$$\mathit{tree} \Rightarrow \mathtt{emp} : \{x = \mathtt{nil}\}$$
$$\mathit{tree} \Rightarrow \exists y, z \,.\, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z$$

```
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

$$tree \Rightarrow \mathtt{emp} : \{x = \mathtt{nil}\}$$
$$tree \Rightarrow \exists y, z \;.\; x \mapsto y, z * tree\, y * tree\, z$$

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

$\{tree\, x\}$

$\{\exists y, z \;.\; x \mapsto y, z * tree\, y * tree\, z\}$

$\{emp\}$

Example: Destruction of a Binary Tree

Example

$$\mathit{tree} \Rightarrow \texttt{emp} : \{x = \texttt{nil}\}$$
$$\mathit{tree} \Rightarrow \exists y, z \,.\, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z$$

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

$\{\mathit{tree}\, x\}$

$\{\exists y, z \,.\, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z\}$

$\{\exists z \,.\, x \mapsto (l, z) * \mathit{tree}\, l * \mathit{tree}\, z\}$

$\{\mathit{emp}\}$

Example: Destruction of a Binary Tree

Example

$$tree \Rightarrow \texttt{emp} : \{x = \texttt{nil}\}$$
$$tree \Rightarrow \exists y, z \,.\, x \mapsto y, z * tree\, y * tree\, z$$

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

$\{tree\, x\}$

$\{\exists y, z \,.\, x \mapsto y, z * tree\, y * tree\, z\}$

$\{\exists z \,.\, x \mapsto (l, z) * tree\, l * tree\, z\}$

$\{x \mapsto l, r * tree\, l * tree\, r\}$

$\{emp\}$

Example: Destruction of a Binary Tree

Example

$$tree \Rightarrow \mathtt{emp} : \{x = \mathtt{nil}\}$$
$$tree \Rightarrow \exists y, z \,.\, x \mapsto y, z * tree\, y * tree\, z$$

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

$\{tree\, x\}$

$\{\exists y, z \,.\, x \mapsto y, z * tree\, y * tree\, z\}$

$\{\exists z \,.\, x \mapsto (l, z) * tree\, l * tree\, z\}$

$\{x \mapsto l, r * tree\, l * tree\, r\}$

$\{x \mapsto l, r * emp * tree\, r\}$

$\{emp\}$

Example: Destruction of a Binary Tree

Example

$$tree \Rightarrow \texttt{emp} : \{x = \texttt{nil}\}$$
$$tree \Rightarrow \exists y, z \,.\, x \mapsto y, z * tree\, y * tree\, z$$

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

$\{tree\, x\}$

$\{\exists y, z \,.\, x \mapsto y, z * tree\, y * tree\, z\}$

$\{\exists z \,.\, x \mapsto (l, z) * tree\, l * tree\, z\}$

$\{x \mapsto l, r * tree\, l * tree\, r\}$

$\{x \mapsto l, r * emp * tree\, r\}$

$\{x \mapsto l, r * emp * emp\}$

$\{emp\}$

Example: Destruction of a Binary Tree

Example

$$
\begin{aligned}
\mathit{tree} &\Rightarrow \texttt{emp} : \{x = \texttt{nil}\} \\
\mathit{tree} &\Rightarrow \exists y, z \,.\, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z
\end{aligned}
$$

`cleanup(x) {`	$\{\mathit{tree}\, x\}$
`  if (x != 0) {`	$\{\exists y, z \,.\, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z\}$
`    l := x.left;`	$\{\exists z \,.\, x \mapsto (l, z) * \mathit{tree}\, l * \mathit{tree}\, z\}$
`    r := x.right;`	$\{x \mapsto l, r * \mathit{tree}\, l * \mathit{tree}\, r\}$
`    cleanup(x.left);`	$\{x \mapsto l, r * \mathit{emp} * \mathit{tree}\, r\}$
`    cleanup(x.right);`	$\{x \mapsto l, r * \mathit{emp} * \mathit{emp}\}$
`    free(x);`	$\{\mathit{emp} * \mathit{emp} * \mathit{emp}\}$
`  }`	
`}`	$\{\mathit{emp}\}$

Example: Destruction of a Binary Tree

Example

$$
\begin{aligned}
\mathit{tree} &\Rightarrow \mathtt{emp} : \{x = \mathtt{nil}\} \\
\mathit{tree} &\Rightarrow \exists y, z \,.\, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z
\end{aligned}
$$

`cleanup(x) {`	$\{\mathit{tree}\, x\}$
`  if (x != 0) {`	$\{\exists y, z \,.\, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z\}$
`    l := x.left;`	$\{\exists z \,.\, x \mapsto (l, z) * \mathit{tree}\, l * \mathit{tree}\, z\}$
`    r := x.right;`	$\{x \mapsto l, r * \mathit{tree}\, l * \mathit{tree}\, r\}$
`    cleanup(x.left);`	$\{x \mapsto l, r * \mathit{emp} * \mathit{tree}\, r\}$
`    cleanup(x.right);`	$\{x \mapsto l, r * \mathit{emp} * \mathit{emp}\}$
`    free(x);`	$\{\mathit{emp} * \mathit{emp} * \mathit{emp}\}$
`  }`	$\{\mathit{emp}\}$
`}`	$\{\mathit{emp}\}$

Semantics

$$\text{States: } \sigma = s, h, \quad \text{stack } s : \mathit{Var} \dashrightarrow \mathbb{Z}, \quad \text{heap } h : \mathbb{Z} \dashrightarrow \mathbb{Z}^{+}$$

$$\Sigma \ ::= \ \mathtt{emp} \mid x \mapsto \mathbf{a} \mid P\mathbf{a} \mid \Sigma * \Sigma \qquad \pi \ ::= \ \mathbf{a} = \mathbf{b} \mid \mathbf{a} \neq \mathbf{b}$$

Semantics of Separation Logic

$$s, h \models \mathit{emp} \ \Leftrightarrow \ \mathit{dom}\, h = \emptyset$$

$$s, h \models x \mapsto \mathbf{a} \ \Leftrightarrow \ \mathit{dom}\, h = \{sx\}, \ h \circ s\, x = s\, \mathbf{a}$$

$$s, h \models P\, x \qquad \text{via unfolding trees (next slide)}$$

$$s, h \models \Sigma_1 * \Sigma_2 \ \Leftrightarrow \ h = h_1 \uplus h_2 \text{ and } s, h_1 \models \Sigma_1 \text{ and } s, h_2 \models \Sigma_2$$

$$(h \text{ is partitioned into } h_1, h_2)$$

$$s, h \models a \sim b \ \Leftrightarrow \ s\, a \sim s\, b$$

$$s, h \models \exists \mathbf{z} \,.\, \Sigma : \Pi \ \Leftrightarrow \ \exists \mathbf{a} \in \mathbb{Z}^{+} \,.\, s[\mathbf{z} \mapsto \mathbf{a}], h \models \Sigma$$

$$\text{and } \forall \pi \in \Pi \,.\, s[\mathbf{z} \mapsto \mathbf{a}], h \models \pi$$

Unfolding Trees

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

$$\begin{tikzcd}
 & & Tx * R \arrow[d] & \\
 & & \exists y, z \,.\, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z \arrow[dl] \arrow[dr] & \\
 & \exists y, z \,.\, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z \arrow[dl] \arrow[dr] & & \mathtt{emp} : \{x = \mathtt{nil}\} \\
\mathtt{emp} : \{x = \mathtt{nil}\} & & \mathtt{emp} : \{x = \mathtt{nil}\} &
\end{tikzcd}$$

Unfolding Trees

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

$$\begin{tikzcd}
& & Tx * R \arrow[d] & \\
& & \exists y, z \,.\, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z \arrow[dl] \arrow[dr] & \\
& \exists y, z \,.\, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z \arrow[dl] \arrow[dr] & & \mathtt{emp} : \{x = \mathtt{nil}\} \\
\mathtt{emp} : \{x = \mathtt{nil}\} & & \mathtt{emp} : \{x = \mathtt{nil}\} &
\end{tikzcd}$$

Corresponding unfolding $[\![t]\!]$:

$$\exists \mathbf{z} \,.\, x \mapsto z_1, z_2 * z_1 \mapsto z_3, z_4 * \mathtt{emp} * \mathtt{emp} * \mathtt{emp} * R \ : \ \{z_2 = z_3 = z_4 = \mathtt{nil}\}$$

$$\equiv \ \exists \mathbf{z} \,.\, x \mapsto z_1, \mathtt{nil} * z_1 \mapsto \mathtt{emp}, \mathtt{emp} * R$$

Unfolding Trees

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

$$\begin{tikzcd}[column sep=small]
 & & Tx * R \arrow[d] & \\
 & & \exists y, z \, . \, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z \arrow[dl] \arrow[dr] & \\
 & \exists y, z \, . \, x \mapsto y, z * \mathit{tree}\, y * \mathit{tree}\, z \arrow[dl] \arrow[dr] & & \mathtt{emp} : \{x = \mathtt{nil}\} \\
\mathtt{emp} : \{x = \mathtt{nil}\} & & \mathtt{emp} : \{x = \mathtt{nil}\} &
\end{tikzcd}$$

Corresponding unfolding $[\![t]\!]$:

$$\exists \mathbf{z} \, . \, x \mapsto z_1, z_2 * z_1 \mapsto z_3, z_4 * \mathtt{emp} * \mathtt{emp} * \mathtt{emp} * R \;:\; \{z_2 = z_3 = z_4 = \mathtt{nil}\}$$

$$\equiv \; \exists \mathbf{z} \, . \, x \mapsto z_1, \mathtt{nil} * z_1 \mapsto \mathtt{emp}, \mathtt{emp} * R$$

$$s, h \models_\Phi P\mathbf{x} \;\Leftrightarrow\; \exists \text{ unfolding tree } t \text{ of } P \, . \, s, h \models [\![t]\!]$$

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 Φ
- Symbolic heap $\varphi\mathbf{x}$
- Set of fully unfolded symbolic heaps $\mathfrak{H} \subseteq \mathit{TSLF}$

Refinement – Overview

Merriam Webster on Refinement:

the act or process of removing unwanted substances from something

Given:

- System of recursive definitions Φ
- Symbolic heap $\varphi\mathbf{x}$
- Set of fully unfolded symbolic heaps $\mathfrak{H} \subseteq$ *TSLF*

Tasks:

- Provide Ψ and $\psi\mathbf{x}$ capturing exactly the unfoldings of Φ and $\varphi\mathbf{x}$ in $\mathfrak{H}$.
- Decide whether each unfolding of $\varphi\mathbf{x}$ is in $\mathfrak{H}$.
- Decide whether there exists an unfolding of $\varphi\mathbf{x}$ in $\mathfrak{H}$.

A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists \mathbf{z} \,.\, \Sigma : \Pi$ *equal to* `nil`*?*

A Toy Problem

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

$$\exists y, z \,.\, x \mapsto y, z$$

A Toy Problem

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

$$\begin{array}{c} \textit{no} \\ \uparrow \\ \exists y, z \,.\, x \mapsto y, z \end{array}$$

A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists \mathbf{z} \,.\, \Sigma : \Pi$ *equal to* `nil`*?*

no $\qquad \exists y, z \,.\, x \mapsto y, z \;\; * \; \mathit{tree}\, y \; * \; \mathit{tree}\, z$

A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists \mathbf{z} \,.\, \Sigma : \Pi$ *equal to* `nil`*?*

$$\text{no} \qquad \underbrace{\exists y, z \,.\, x \mapsto y, z}_{\uparrow\ \text{no}} \ * \ \underbrace{\mathit{tree}\, y}_{\text{no}} \ * \ \underbrace{\mathit{tree}\, z}_{\text{no}}$$

A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists \mathbf{z} \,.\, \Sigma : \Pi$ *equal to* `nil`*?*

$$\text{yes}$$

$$\uparrow$$

$$\textit{no} \qquad \exists y, z \,.\, x \mapsto y, z \;*\; \mathit{tree}\, y \;*\; \mathit{tree}\, z$$

$$\uparrow \qquad \uparrow$$

$$\textit{no} \qquad \textit{yes}$$

A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists \mathbf{z} \,.\, \Sigma : \Pi$ *equal to* `nil`*?*

$$\textit{no} \qquad \exists y, z \,.\, \overset{\textit{yes}}{\overset{\uparrow}{x}} \mapsto y, z \;*\; \underset{\textit{yes}}{\underset{\uparrow}{\mathit{tree}\, y}} \;*\; \underset{\textit{no}}{\underset{\uparrow}{\mathit{tree}\, z}}$$

A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists \mathbf{z} \,.\, \Sigma : \Pi$ *equal to* `nil`*?*

$$\overbrace{\exists y, z \,.\, x \mapsto y, z}^{\textit{yes}} * \underbrace{\textit{tree}\, y}_{\textit{yes}} * \underbrace{\textit{tree}\, z}_{\textit{yes}}$$

no

A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists \mathbf{z} \,.\, \Sigma : \Pi$ *equal to* `nil`*?*

$$\begin{array}{cccccc} & \textit{yes} & & & & \\ & \uparrow & & & & \\ \textit{no} & \exists y, z \,.\, x \mapsto y, z & * & \mathit{tree}\, y & * & \mathit{tree}\, z \\ & & & \uparrow & & \uparrow \\ & & & \textit{yes} & & \textit{yes} \end{array}$$

This is a transition relation of a finite tree automaton.

A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists \mathbf{z} \,.\, \Sigma : \Pi$ *equal to* `nil`*?*

yes

no $\qquad \exists y, z \,.\, x \mapsto y, z \; * \; \mathit{tree}\, y \; * \; \mathit{tree}\, z$

yes *yes*

This is a transition relation of a finite tree automaton.

Naive decision procedure:

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

A Toy Problem

Is some variable of symbolic heap $\varphi \triangleq \exists \mathbf{z} \,.\, \Sigma : \Pi$ *equal to* $\texttt{nil}$*?*

$$
\begin{array}{ccccc}
 & \textit{yes} & & & \\
 & \uparrow & & & \\
\textit{no} \qquad & \exists y, z \,.\, x \mapsto y, z & * & \textit{tree}\, y & * \;\; \textit{tree}\, z \\
 & & & \uparrow & \uparrow \\
 & & & \textit{yes} & \textit{yes}
\end{array}
$$

This is a transition relation of a finite tree automaton.

Naive decision procedure:

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

Refinement: replace each predicate call P by $(P, \textit{yes})$ and $(P, \textit{no})$ according to observation from above.

Compositional Heap Properties

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

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

$$(p, [\![t]\!]) \in \mathfrak{H}_{\mathbb{H}} \text{ if and only if } (p, \varphi\mathbf{x}, \mathbf{q}) \in \Delta[\![\Phi]\!] \text{ and } (\mathbf{q}_i, [\![t|_i]\!]) \in \mathfrak{H}_{\mathbb{H}}$$

(where $1 \leq i \leq |Calls(\varphi\mathbf{x})|$).

Compositional Heap Properties

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

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

$$(p, [\![t]\!]) \in \mathfrak{H}_{\mathbb{H}} \text{ if and only if } (p, \varphi\mathbf{x}, \mathbf{q}) \in \Delta[\![\Phi]\!] \text{ and } (\mathbf{q}_i, [\![t|_i]\!]) \in \mathfrak{H}_{\mathbb{H}}$$

(where $1 \leq i \leq |Calls(\varphi\mathbf{x})|$).

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

Compositional Heap Properties

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

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

$$(p, [\![t]\!]) \in \mathfrak{H}_{\mathbb{H}} \text{ if and only if } (p, \varphi\mathbf{x}, \mathbf{q}) \in \Delta[\![\Phi]\!] \text{ and } (\mathbf{q}_i, [\![t|_i]\!]) \in \mathfrak{H}_{\mathbb{H}}$$

(where $1 \leq i \leq |Calls(\varphi\mathbf{x})|$).

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

$\mathfrak{H}$ is a compositional compositional heap property if there exists a compositional $\mathfrak{H}_{\mathbb{H}}$ and $\mathbb{H}_0 \subseteq \mathbb{H}$ such that

$$\varphi \in \mathfrak{H} \text{ if and only if } \exists h \in \mathbb{H}_0 \;.\; (h, \varphi) \in \mathfrak{H}_{\mathbb{H}}.$$

Compositional Heap Properties – Toy Example

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

Is some variable of a symbolic heap equal to `nil`*?*

$$\mathfrak{H} \triangleq \{\exists \mathbf{z} \,.\, \Sigma : \Pi \in TSLF \mid \exists x \in Var \,.\, x = \mathtt{nil} \in \Pi\}$$

Compositional Heap Properties – Toy Example

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

Is some variable of a symbolic heap equal to `nil`*?*

$$\mathfrak{H} \triangleq \{\exists \mathbf{z} \,.\, \Sigma : \Pi \in TSLF \mid \exists x \in Var \,.\, x = \mathtt{nil} \in \Pi\}$$

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

- $\mathbb{H} \triangleq \{0, 1\}$, $\mathbb{H}_0 \triangleq \{1\}$
- $(q, \varphi) \in \mathfrak{H}_{\mathbb{H}}$ iff $(q = 0 \wedge \varphi \notin \mathfrak{P}) \vee (q = 1 \wedge \varphi \in \mathfrak{P})$
- $(p, \exists \mathbf{z} \,.\, \Sigma : \Pi, q_1 \ldots q_n) \in \Delta[\![\Phi]\!]$ iff $p = \max\{[\exists x \in Var \,.\, x = \mathtt{nil} \in \Pi], q_1, \ldots, q_n\}$

A Refinement Theorem

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

Theorem

For each compositional heap property $\mathfrak{H}$, $\varphi \in \mathit{SLF}$ and $\Phi \in \mathit{SRD}$ one can effectively construct $\Psi \in \mathit{SRD}$ and $\psi \in \mathit{SLF}$ such that for each unfolding tree t of φ in Φ:

$$[\![t]\!] \in \mathfrak{H} \text{ iff } \exists \text{ unfolding tree } t' \text{ of } \psi \text{ in } \Psi \text{ such that } [\![t]\!] \equiv [\![t']\!].$$

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

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

A Refinement Theorem

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

Theorem

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

$$[\![t]\!] \in \mathfrak{H} \text{ iff } \exists \text{ unfolding tree } t' \text{ of } \psi \text{ in } \Psi \text{ such that } [\![t]\!] \equiv [\![t']\!].$$

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

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

The refinement of Φ by $\mathfrak{H}_{\mathbb{H}}$ is the least set $\Phi \upharpoonright \mathfrak{H}_Q$ satisfying

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

where each predicate call $P_i\,\mathbf{z}$ is replaced by $(P_i, \mathbf{q}_i)\,\mathbf{z}$ in $\varphi[\mathbf{q}]$.

A Refinement Theorem

Theorem

Compositional heap properties are closed under Boolean operations.

Theorem

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

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

$$\mathcal{O}\left(\left[\mathbb{H}^{cw+1} \cdot (|\Phi| + |\varphi|) + |\mathbb{H}_0| \cdot |\varphi|\right] \cdot T(\Delta[\![\Phi]\!])\right)$$

Outline

1. Motivation
2. Symbolic Heaps with Recursive Definitions
3. A Refinement Theorem
4. Applications
5. Conclusion

Satisfiability

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

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

where k bounds the number of free variables.

Satisfiability

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

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

where k bounds the number of free variables.

Theorem

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

Satisfiability

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

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

where k bounds the number of free variables.

Theorem

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

Theorem

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

Model-Checking

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

$$\mathfrak{M}_k \triangleq \{\varphi\mathbf{x} \in TSLF_k \mid s, h \models \varphi\mathbf{x}\}$$

where k bounds the number of free variables.

Theorem

$\mathfrak{M}_k$ is a compositional heap property.

Theorem

- $\texttt{SL-MC}$ *is* $\textsc{ExpTime}$*–complete (Brotherstone, 2016)*
- $\texttt{SL-MC}_k$ *is* NP*–complete for* $k \geq 3$ *(Brotherstone, 2016)*
- $\texttt{SL-MC}_{k,cw}$ *is* $\textsc{PTime}$*–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.

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

where k bounds the number of free variables.

Theorem

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

Theorem

- `Establishment` *is in* $\textsc{ExpTime}$
- $\texttt{Establishment}_k$ *is in* NP *for* $k \geq 3$
- $\texttt{Establishment}_{k,cw}$ *is* $\textsc{PTime}$*–complete for* $cw \geq 2$, $k \geq 0$

Equality of Program Variables

`EqVar`: Given $\Phi \in SRD$, $\varphi\mathbf{x} \in SLF$ and i, j, decide whether $\mathbf{x}_i = \mathbf{x}_j$ holds for some unfolding.

$$\mathfrak{V}_k \triangleq \{\varphi \in TSLF_k \mid \mathbf{x}_i = \mathbf{x}_j \in \Pi^*(\varphi) \text{ and } \mathbf{x}_i \neq \mathbf{x}_j \notin \Pi^*(\varphi)\}$$

where k bounds the number of free variables.

Theorem

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

Theorem

- `EqVar` *is in* $\textsc{ExpTime}$*–complete.*
- $\texttt{EqVar}_k$ *is in* NP*–complete for* $k \geq 3$
- $\texttt{EqVar}_{k,cw}$ *is* $\mathrm{P}\textsc{Time}$*–complete for* $cw \geq 2$, $k \geq 0$

Reachability

`Reachability`: Given $\Phi \in SRD$, $\varphi\mathbf{x} \in SLF$ and i, j, decide whether $\mathbf{x}_i \rightsquigarrow \mathbf{x}_j$ holds for some unfolding.

$$\mathfrak{R}_k \triangleq \{\varphi \in TSLF_k \mid \mathbf{x}_i \rightsquigarrow \mathbf{x}_j\}$$

where k bounds the number of free variables.

Theorem

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

Theorem

- `Reachability` *is in* $\textsc{ExpTime}$*–complete.*
- $\texttt{Reachability}_k$ *is in* NP*–complete for* $k \geq 3$
- $\texttt{Reachability}_{k,cw}$ *is* $\mathrm{P}\textsc{Time}$*–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 $\mathfrak{H}$–congruent if for each $\vartheta \in SLF$ with one predicate call P,

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

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 $\mathfrak{H}$–congruent if for each $\vartheta \in SLF$ with one predicate call P,

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

Theorem

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

Entailment

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

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

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

Definition

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

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

Theorem

$\mathfrak{H}$ is compositional if the $\mathfrak{H}$–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

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

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

Definition

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

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

Theorem

$\mathfrak{H}$ is compositional if the $\mathfrak{H}$–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 $\mathfrak{H}$–congruences for restricted fragments?

Conclusion

1. Symbolic Heaps with Recursive Definitions

   $\varphi\mathbf{x} \triangleq \exists\mathbf{z} \,.\, \Sigma : \Pi, \quad \Phi \triangleq \{P \Rightarrow \varphi\mathbf{x} \mid \ldots\},$ fragment used by most tools

2. A Refinement Theorem

   Refine Φ and $\varphi\mathbf{x}$ such that each unfolding satisfies property $\mathfrak{H}$.

3. Applications

   Satisfiability, Model-Checking, Establishment, Reachability, ... are compositional

4. Future Work

   Apply compositional heap properties to the entailment problem for restricted fragments