The Complexity of Abduction for Separated Heap Abstractions

Nikos Gorogiannis Max Kanovich Peter O'Hearn

Queen Mary University of London

July 13th, 2011

Motivation

Motivation

- Calcagno, Distefano, O'Hearn, Yang propose (POPL'09) *Compositional shape analysis by means of bi-abduction.*

Motivation

- Calcagno, Distefano, O'Hearn, Yang propose (POPL'09) *Compositional shape analysis by means of bi-abduction.*
- Further papers extend the analysis, apply it to other domains.

Motivation

- Calcagno, Distefano, O'Hearn, Yang propose (POPL'09) *Compositional shape analysis by means of bi-abduction.*
- Further papers extend the analysis, apply it to other domains.
- The published algorithms for abduction are incomplete.

Motivation

- Calcagno, Distefano, O'Hearn, Yang propose (POPL'09) *Compositional shape analysis by means of bi-abduction.*
- Further papers extend the analysis, apply it to other domains.
- The published algorithms for abduction are incomplete.
- Is there a complete algorithm? (is the problem decidable?).

Motivation

- Calcagno, Distefano, O'Hearn, Yang propose (POPL'09) *Compositional shape analysis by means of bi-abduction.*
- Further papers extend the analysis, apply it to other domains.
- The published algorithms for abduction are incomplete.
- Is there a complete algorithm? (is the problem decidable?).
- If yes, what is the complexity for a common abstract domain?

A Heap of Problems

A Heap of Problems

$$\{\quad \mathsf{ls}(x,0) \wedge \mathsf{ls}(y,0) \quad\}\,\texttt{append(x,y)}\,\{\quad \mathsf{ls}(x,0) \quad\}$$

A Heap of Problems

$$\{\quad \mathsf{ls}(x,0) \wedge \mathsf{ls}(y,0) \quad\}\, \texttt{append(x,y)}\, \{\quad \mathsf{ls}(x,0) \quad\}$$

How do we prevent sharing in the precondition?

A Heap of Problems

$$\{ \quad \mathsf{ls}(x,0) \wedge \mathsf{ls}(y,0) \quad \} \, \texttt{append(x,y)} \, \{ \quad \mathsf{ls}(x,0) \quad \}$$

How do we prevent sharing in the precondition?

- Reachability?

$$\left\{ \begin{array}{c} \forall z.\mathsf{reach}(x,z) \Rightarrow \neg\mathsf{reach}(y,z) \wedge \\ \forall w.\mathsf{reach}(y,w) \Rightarrow \neg\mathsf{reach}(x,w) \wedge \\ \mathsf{ls}(x,0) \wedge \mathsf{ls}(y,0) \end{array} \right\}$$

A Heap of Problems

$$\{\quad \mathsf{ls}(x,0) \wedge \mathsf{ls}(y,0) \quad\}\ \texttt{append(x,y)}\ \{\quad \mathsf{ls}(x,0) \quad\}$$

How do we prevent sharing in the precondition?

- Reachability?

$$\left\{\begin{array}{c} \forall z.\mathsf{reach}(x,z) \Rightarrow \neg\mathsf{reach}(y,z) \wedge \\ \forall w.\mathsf{reach}(y,w) \Rightarrow \neg\mathsf{reach}(x,w) \wedge \\ \mathsf{ls}(x,0) \wedge \mathsf{ls}(y,0) \end{array}\right\}$$

- Separation Logic?

$$\{\quad \mathsf{ls}(x,0) * \mathsf{ls}(y,0) \quad\}\ \texttt{append(x,y)}\ \{\quad \mathsf{ls}(x,0) \quad\}$$

Heaps and Stars

Heaps and Stars

$$\{\quad \mathsf{ls}(x,0) * \mathsf{ls}(y,0) \quad\}\,\texttt{append(x,y)}\,\{\quad \mathsf{ls}(x,0) \quad\}$$

Heaps and Stars

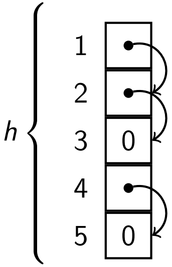
$$\{\quad \mathsf{ls}(x,0) * \mathsf{ls}(y,0) \quad\}\,\texttt{append(x,y)}\,\{\quad \mathsf{ls}(x,0) \quad\}$$

Suppose for a model, $x = 1$ and $y = 4$.

Heaps and Stars

$$\{ \quad \mathsf{ls}(x,0) * \mathsf{ls}(y,0) \quad \}\, \texttt{append(x,y)}\, \{ \quad \mathsf{ls}(x,0) \quad \}$$

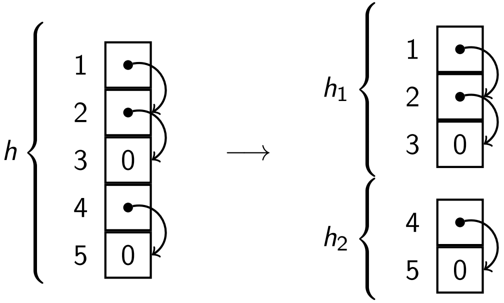
Suppose for a model, $x = 1$ and $y = 4$.



Heaps and Stars

$$\{\quad \mathsf{ls}(x,0) * \mathsf{ls}(y,0) \quad\}\,\texttt{append(x,y)}\,\{\quad \mathsf{ls}(x,0) \quad\}$$

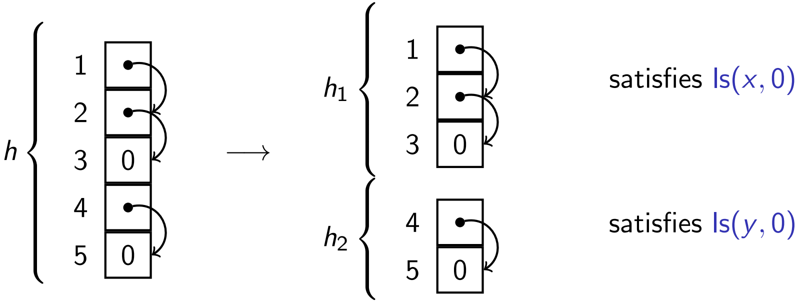
Suppose for a model, $x = 1$ and $y = 4$.



Heaps and Stars

$$\{\ \mathsf{ls}(x,0) * \mathsf{ls}(y,0)\ \}\ \texttt{append(x,y)}\ \{\ \mathsf{ls}(x,0)\ \}$$

Suppose for a model, $x = 1$ and $y = 4$.



Semantics

Semantics

- Stack $s : Var \rightarrow Values$.

Semantics

- Stack $s : Var \to Values$.
- Heap $h : Addresses \to_f Values$.

Semantics

- Stack $s : \mathit{Var} \to \mathit{Values}$.
- Heap $h : \mathit{Addresses} \to_f \mathit{Values}$.
- $(s, h) \models$ true always.

Semantics

- Stack $s : \mathit{Var} \to \mathit{Values}$.
- Heap $h : \mathit{Addresses} \to_f \mathit{Values}$.
- $(s, h) \models \mathsf{true}$ always.
- $(s, h) \models x = y$ iff $s(x) = s(y)$ (same for $\neq$).

Semantics

- Stack $s : \mathit{Var} \to \mathit{Values}$.
- Heap $h : \mathit{Addresses} \to_f \mathit{Values}$.
- $(s, h) \models$ true always.
- $(s, h) \models x = y$ iff $s(x) = s(y)$ (same for $\neq$).
- $(s, h) \models A \wedge B$ as usual.

Semantics

- Stack $s : Var \to Values$.
- Heap $h : Addresses \to_f Values$.
- $(s, h) \models$ true always.
- $(s, h) \models x = y$ iff $s(x) = s(y)$ (same for $\neq$).
- $(s, h) \models A \wedge B$ as usual.
- $(s, h) \models$ emp iff $h = \emptyset$.

Semantics

- Stack $s : \mathit{Var} \to \mathit{Values}$.
- Heap $h : \mathit{Addresses} \to_f \mathit{Values}$.
- $(s, h) \models$ true always.
- $(s, h) \models x = y$ iff $s(x) = s(y)$ (same for $\neq$).
- $(s, h) \models A \wedge B$ as usual.
- $(s, h) \models$ emp iff $h = \emptyset$.
- $(s, h) \models x \mapsto y$ iff

Semantics

- Stack $s : Var \rightarrow Values$.
- Heap $h : Addresses \rightarrow_f Values$.
- $(s, h) \models$ true always.
- $(s, h) \models x = y$ iff $s(x) = s(y)$ (same for $\neq$).
- $(s, h) \models A \wedge B$ as usual.
- $(s, h) \models$ emp iff $h = \emptyset$.
- $(s, h) \models x \mapsto y$ iff
  - $s(x) = u,\ s(y) = v$

Semantics

- Stack $s : Var \to Values$.
- Heap $h : Addresses \to_f Values$.
- $(s, h) \models$ true always.
- $(s, h) \models x = y$ iff $s(x) = s(y)$ (same for $\neq$).
- $(s, h) \models A \wedge B$ as usual.
- $(s, h) \models$ emp iff $h = \emptyset$.
- $(s, h) \models x \mapsto y$ iff
  - $s(x) = u,\ s(y) = v$
  - $h = \{(u, v)\}$.

More semantics

More semantics

- $(s, h) \models A * B$ iff there are h_A, h_B such that

More semantics

- $(s, h) \models A * B$ iff there are h_A, h_B such that
  - $(s, h_A) \models A$

More semantics

- $(s, h) \models A * B$ iff there are h_A, h_B such that
    - $(s, h_A) \models A$
    - $(s, h_B) \models B$

More semantics

- $(s, h) \models A * B$ iff there are h_A, h_B such that
  - $(s, h_A) \models A$
  - $(s, h_B) \models B$
  - h_A and h_B are domain-disjoint and $h = h_A \cup h_B$.

More semantics

- $(s, h) \models A * B$ iff there are h_A, h_B such that
  - $(s, h_A) \models A$
  - $(s, h_B) \models B$
  - h_A and h_B are domain-disjoint and $h = h_A \cup h_B$.
- $(s, h) \models \mathsf{ls}(x, y)$ iff

More semantics

- $(s, h) \models A * B$ iff there are h_A, h_B such that
  - $(s, h_A) \models A$
  - $(s, h_B) \models B$
  - h_A and h_B are domain-disjoint and $h = h_A \cup h_B$.
- $(s, h) \models \mathsf{ls}(x, y)$ iff
  - $(s, h) \models x \neq y \wedge x \mapsto y$, or,

More semantics

- $(s, h) \models A * B$ iff there are h_A, h_B such that
  - $(s, h_A) \models A$
  - $(s, h_B) \models B$
  - h_A and h_B are domain-disjoint and $h = h_A \cup h_B$.
- $(s, h) \models \mathsf{ls}(x, y)$ iff
  - $(s, h) \models x \neq y \wedge x \mapsto y$, or,
  - $(s, h) \models x \neq y \wedge \exists z. (x \mapsto z * \mathsf{ls}(z, y))$.

More semantics

- $(s, h) \models A * B$ iff there are h_A, h_B such that
  - $(s, h_A) \models A$
  - $(s, h_B) \models B$
  - h_A and h_B are domain-disjoint and $h = h_A \cup h_B$.
- $(s, h) \models \mathsf{ls}(x, y)$ iff
  - $(s, h) \models x \neq y \wedge x \mapsto y$, or,
  - $(s, h) \models x \neq y \wedge \exists z. (x \mapsto z * \mathsf{ls}(z, y))$.

  I.e., non-empty, acyclic list segments.

A Bit More on the Semantics

What does it mean for $(s, h) \models A * \mathsf{true}$ to be true?

A Bit More on the Semantics

What does it mean for $(s, h) \models A * \mathsf{true}$ to be true?
That there is a heap $h_A \subseteq h$ such that $(s, h_A) \models A$.

A Bit More on the Semantics

What does it mean for $(s, h) \models A * \mathsf{true}$ to be true?
That there is a heap $h_A \subseteq h$ such that $(s, h_A) \models A$.

We work with *symbolic heaps*, e.g.,

A Bit More on the Semantics

What does it mean for $(s, h) \models A * \mathsf{true}$ to be true?
That there is a heap $h_A \subseteq h$ such that $(s, h_A) \models A$.

We work with *symbolic heaps*, e.g.,

$$x \neq y \wedge w \neq z \wedge x \mapsto y * \mathsf{ls}(y, x)$$

Extracting Preconditions from Code

$\{\mathsf{emp} \qquad\qquad \}$
```
    *x = 0;
```

Extracting Preconditions from Code

$\{\mathsf{emp} \qquad\qquad\}$

```
    *x = 0;
```

- Suppose the current state is emp.

Extracting Preconditions from Code

```
{emp        }
    *x = 0;
```

- Suppose the current state is emp.
- The next command is `*x = 0` .

Extracting Preconditions from Code

$\{\mathsf{emp} \qquad\qquad \}$
```
    *x = 0;
```

- Suppose the current state is emp.
- The next command is `*x = 0` .
- Its precondition is $x \mapsto x' * \mathsf{true}$.

Extracting Preconditions from Code

```
{emp          }
    *x = 0;
```

- Suppose the current state is emp.
- The next command is `*x = 0` .
- Its precondition is $x \mapsto x' * \mathsf{true}$.
- Is it true that $\mathsf{emp} \models x \mapsto x' * \mathsf{true}$?

Extracting Preconditions from Code

$\{\mathsf{emp} \qquad\qquad \}$

```
    *x = 0;
```

- Suppose the current state is emp.
- The next command is `*x = 0` .
- Its precondition is $x \mapsto x' * \mathsf{true}$.
- Is it true that $\mathsf{emp} \models x \mapsto x' * \mathsf{true}$?
- No, but $\mathsf{emp} * x \mapsto x' \models x \mapsto x' * \mathsf{true}$.

Extracting Preconditions from Code

$\{\mathrm{emp} * x \mapsto x'\}$

```
    *x = 0;
```

- Suppose the current state is emp.
- The next command is `*x = 0` .
- Its precondition is $x \mapsto x' * \mathrm{true}$.
- Is it true that $\mathrm{emp} \models x \mapsto x' * \mathrm{true}$?
- No, but $\mathrm{emp} * x \mapsto x' \models x \mapsto x' * \mathrm{true}$.

Abduction

What is abduction in AI?

Abduction

What is abduction in AI?

- Given A, B such that $A \not\models B$,

Abduction

What is abduction in AI?

- Given A, B such that $A \not\models B$,
- find X such that $A, X \models B$.

Abduction

What is abduction in AI?

- Given A, B such that $A \not\models B$,
- find X such that $A, X \models B$.
- But, ignore trivial solutions such as $\bot$ or $A \to B$.

Abduction

What is abduction in AI?

- Given A, B such that $A \not\models B$,
- find X such that $A, X \models B$.
- But, ignore trivial solutions such as $\perp$ or $A \to B$.

What is Abduction in Separation Logic?

Abduction

What is abduction in AI?

- Given A, B such that $A \not\models B$,
- find X such that $A, X \models B$.
- But, ignore trivial solutions such as $\bot$ or $A \to B$.

What is Abduction in Separation Logic?

- Given formulae A, B such that $A \not\models B$.

Abduction

What is abduction in AI?

- Given A, B such that $A \not\models B$,
- find X such that $A, X \models B$.
- But, ignore trivial solutions such as $\perp$ or $A \rightarrow B$.

What is Abduction in Separation Logic?

- Given formulae A, B such that $A \not\models B$.
- Find symbolic heap X such that

Abduction

What is abduction in AI?

- Given A, B such that $A \not\models B$,
- find X such that $A, X \models B$.
- But, ignore trivial solutions such as $\perp$ or $A \rightarrow B$.

What is Abduction in Separation Logic?

- Given formulae A, B such that $A \not\models B$.
- Find symbolic heap X such that
  - $A * X \models B$,

Abduction

What is abduction in AI?

- Given A, B such that $A \not\models B$,
- find X such that $A, X \models B$.
- But, ignore trivial solutions such as $\perp$ or $A \rightarrow B$.

What is Abduction in Separation Logic?

- Given formulae A, B such that $A \not\models B$.
- Find symbolic heap X such that
  - $A * X \models B$,
  - and $A * X$ is consistent.

Examples of Abduction

Examples of Abduction

$$\mathsf{emp} * \qquad\qquad \models \quad x \mapsto 0$$

Examples of Abduction

$$\mathsf{emp} * x \mapsto 0 \quad \models \quad x \mapsto 0$$

Examples of Abduction

$$\begin{array}{rcl} \mathsf{emp} * x \mapsto 0 & \models & x \mapsto 0 \\ y \mapsto 0 * & \models & x \mapsto 0 \end{array}$$

Examples of Abduction

$$\begin{aligned} \mathsf{emp} * x \mapsto 0 &\quad\models\quad x \mapsto 0 \\ y \mapsto 0 * x = y &\quad\models\quad x \mapsto 0 \end{aligned}$$

Examples of Abduction

$$\begin{array}{rcl}
\mathsf{emp} * x \mapsto 0 & \models & x \mapsto 0 \\
y \mapsto 0 * x = y & \models & x \mapsto 0 \\
y \mapsto 0 * & \models & x \mapsto 0 * \mathsf{true}
\end{array}$$

Examples of Abduction

$$\begin{aligned}
\mathsf{emp} * x \mapsto 0 &\models x \mapsto 0 \\
y \mapsto 0 * x = y &\models x \mapsto 0 \\
y \mapsto 0 * x \mapsto 0 &\models x \mapsto 0 * \mathsf{true}
\end{aligned}$$

Examples of Abduction

$$\begin{array}{rcl}
\mathsf{emp} * x \mapsto 0 & \models & x \mapsto 0 \\
y \mapsto 0 * x = y & \models & x \mapsto 0 \\
y \mapsto 0 * x \mapsto 0 & \models & x \mapsto 0 * \mathsf{true} \\
x \mapsto y * \qquad\qquad & \models & \mathsf{ls}(x, z)
\end{array}$$

Examples of Abduction

$$\begin{array}{rcl}
\mathsf{emp} * x \mapsto 0 & \models & x \mapsto 0 \\
y \mapsto 0 * x = y & \models & x \mapsto 0 \\
y \mapsto 0 * x \mapsto 0 & \models & x \mapsto 0 * \mathsf{true} \\
x \mapsto y * y = z \wedge z \neq x & \models & \mathsf{ls}(x, z)
\end{array}$$

Examples of Abduction

$$\begin{array}{rcl}
\mathsf{emp} * x \mapsto 0 & \models & x \mapsto 0 \\
y \mapsto 0 * x = y & \models & x \mapsto 0 \\
y \mapsto 0 * x \mapsto 0 & \models & x \mapsto 0 * \mathsf{true} \\
x \mapsto y * y = z \wedge z \neq x & \models & \mathsf{ls}(x, z) \\
x \mapsto y * \qquad\qquad\qquad\qquad & \models & \mathsf{ls}(x, z)
\end{array}$$

Examples of Abduction

$$\begin{aligned}
\mathsf{emp} * x \mapsto 0 &\models x \mapsto 0 \\
y \mapsto 0 * x = y &\models x \mapsto 0 \\
y \mapsto 0 * x \mapsto 0 &\models x \mapsto 0 * \mathsf{true} \\
x \mapsto y * y = z \wedge z \neq x &\models \mathsf{ls}(x, z) \\
x \mapsto y * (z \neq x \wedge \mathsf{ls}(y, z)) &\models \mathsf{ls}(x, z)
\end{aligned}$$

Examples of Abduction

$$\begin{array}{rcl}
\mathsf{emp} * x \mapsto 0 & \models & x \mapsto 0 \\
y \mapsto 0 * x = y & \models & x \mapsto 0 \\
y \mapsto 0 * x \mapsto 0 & \models & x \mapsto 0 * \mathsf{true} \\
x \mapsto y * y = z \land z \neq x & \models & \mathsf{ls}(x, z) \\
x \mapsto y * (z \neq x \land \mathsf{ls}(y, z)) & \models & \mathsf{ls}(x, z) \\
\mathsf{ls}(x, z) * \mathsf{ls}(y, z) * \qquad\qquad & \models & \mathsf{ls}(x, w) * \mathsf{ls}(y, w)
\end{array}$$

Examples of Abduction

$$\begin{array}{rcl}
\mathsf{emp} * x \mapsto 0 & \models & x \mapsto 0 \\
y \mapsto 0 * x = y & \models & x \mapsto 0 \\
y \mapsto 0 * x \mapsto 0 & \models & x \mapsto 0 * \mathsf{true} \\
x \mapsto y * y = z \wedge z \neq x & \models & \mathsf{ls}(x, z) \\
x \mapsto y * (z \neq x \wedge \mathsf{ls}(y, z)) & \models & \mathsf{ls}(x, z) \\
\mathsf{ls}(x, z) * \mathsf{ls}(y, z) * z = w & \models & \mathsf{ls}(x, w) * \mathsf{ls}(y, w)
\end{array}$$

Results

Results

Abduction is decidable (interpolation-like result).

Results

Abduction is decidable (interpolation-like result).

Domain	"$*$true" $\notin$ RHS	"$*$true" $\in$ RHS
$\mapsto$		
$\mapsto$, ls		

Results

Abduction is decidable (interpolation-like result).

Domain	"$*$true" $\notin$ RHS	"$*$true" $\in$ RHS
$\mapsto$	NP-complete	
$\mapsto$, ls		

Results

Abduction is decidable (interpolation-like result).

Domain	"$*$true" $\notin$ RHS	"$*$true" $\in$ RHS
$\mapsto$	NP-complete	PTIME
$\mapsto$, ls		

Results

Abduction is decidable (interpolation-like result).

Domain	“$*$true” $\notin$ RHS	“$*$true” $\in$ RHS
$\mapsto$	NP-complete	PTIME
$\mapsto$, ls	NP-complete	

Results

Abduction is decidable (interpolation-like result).

Domain	“$*$true” $\notin$ RHS	“$*$true” $\in$ RHS
$\mapsto$	NP-complete	PTIME
$\mapsto$, ls	NP-complete	NP-complete

Conclusions

- The abduction problem is NP-complete.

Conclusions

- The abduction problem is NP-complete.
- Lower bounds should carry over to other heap abstractions.

Conclusions

- The abduction problem is NP-complete.
- Lower bounds should carry over to other heap abstractions.
- Cases occuring in practice can be usually treated in polytime.

Conclusions

- The abduction problem is NP-complete.
- Lower bounds should carry over to other heap abstractions.
- Cases occuring in practice can be usually treated in polytime.
- There is a polytime algorithm for a fixed number of lists.