Shape Analysis via Monotonic Abstraction

Parosh Aziz Abdulla

Uppsala University

February 9, 2010

(Joint work with Ahmed Bouajjani, Jonathan Cederberg, Fédéric Haziza and Ahmed Rezine.)
1. Background
2. Monotonic Transition Systems
3. Monotonic Abstraction
4. Singly-Linked Lists
5. Ordering
6. Bad Configurations
7. Computing Predecessors
8. Termination
9. Experimental Results
Model Checking + Abstraction

Infinite-state System → Abstraction → Finite-State System → Model Checking
Model Checking + Abstraction

Infinite-state System → Abstraction → Infinite-State System → Model Checking
Monotonic Transition Systems

Monotonic Transition System

- $\mathcal{T} = (S, \rightarrow, \preceq)$
- $S$: (infinite) set of configurations
- $\rightarrow$: transition relation
- $\preceq$: preorder on $S$
Monotonic Transition System

- $\mathcal{T} = (S, \rightarrow, \preceq)$
- $S$: (infinite) set of configurations
- $\rightarrow$: transition relation
- $\preceq$: preorder on $S$

Monotonicity

$C_1 \rightarrow C_2$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Monotonic Transition Systems

Monotonic Transition System

- $\mathcal{T} = (S, \rightarrow, \preceq)$
- $S$: (infinite) set of configurations
- $\rightarrow$: transition relation
- $\preceq$: preorder on $S$

Monotonicity

$C_1 \rightarrow C_2$

$|\text{人}| \rightarrow |\text{人}|$

$C_3 \rightarrow C_4$
Monotonic Transition Systems

Monotonic Transition System

- $\mathcal{T} = (S, \longrightarrow, \preceq)$
- $S$: (infinite) set of configurations
- $\longrightarrow$: transition relation
- $\preceq$: preorder on $S$

Monotonicity

$\mathcal{C}_1 \xrightarrow{} \mathcal{C}_2$

$\mathcal{C}_3 \xrightarrow{} \mathcal{C}_4$

Examples

- Petri Nets.
- Lossy Channel Systems.
- Timed Petri Nets.
- Multiset Rewriting Systems.
- Broadcast Protocols.
- etc.
Upward-Closed Sets (UC)

Why UC?
- Bad sets of states are UC
- Safety properties = reachability of UC
- Uniquely characterized by generator
  - Simple representation = minimal element
Upward-Closed Sets (UC)

Why UC?
- Bad sets of states are UC
  - safety properties = reachability of UC
- Uniquely characterized by generator
  - simple representation = minimal element
Monotonicity implies UC is closed under $Pre$
Monotonicity and Upward Closedness

Monotonicity implies UC is closed under $Pre$

$Pre(U)$: Upward Closed?

$U$: Upward Closed
Monotonicity and Upward Closedness

Monotonicity implies UC is closed under $Pre$

$Pre(U)$: Upward Closed?

$U$: Upward Closed
Monotonicity and Upward Closedness

Monotonicity implies UC is closed under $Pre$

$Pre(U)$: Upward Closed?

$U$: Upward Closed
Monotonicity and Upward Closedness

Monotonicity implies UC is closed under $Pre$

$Pre(U)$: Upward Closed?

$U$: Upward Closed
Monotonicity and Upward Closedness

Monotonicity implies UC is closed under $\text{Pre}$

$\text{Pre}(U)$: Upward Closed?  \quad \quad \quad \quad \quad U$: Upward Closed
Monotonicity implies UC is closed under $Pre$.

$Pre(U)$: Upward Closed? Yes

$U$: Upward Closed
Problem

- When transition system not monotonic
Monotonic Abstraction

Problem
- When transition system not monotonic

Solution: Monotonic Abstraction
- Force monotonicity!
- Over-Approximation of non-monotonic transitions
Monotonic Abstraction

**Problem**
- When transition system not monotonic

**Solution:** Monotonic Abstraction
- Force monotonicity!
- Over-Approximation of non-monotonic transitions
Monotonic Abstraction

Problem

- When transition system not monotonic

Solution: Monotonic Abstraction

- Force monotonicity!
- Over-Approximation of non-monotonic transitions

\[ c_1 \preceq c_2 \]

\[ \gamma | c_3 \to c_2 \]
Monotonic Abstraction

Problem
- When transition system not monotonic

Solution: Monotonic Abstraction
- Force monotonicity!
- Over-Approximation of non-monotonic transitions

\[ c_1 \preceq c_2 \]

\[ \gamma | \]

\[ c_3 \]
Monotonic Abstraction

Problem
- When transition system not monotonic

Solution: Monotonic Abstraction
- Force monotonicity!
- Over-Approximation of non-monotonic transitions

Examples
- Parameterized Systems.
- Shape Analysis.
Shape Analysis: Singly Linked Lists

Transition System = (S, →, ⪯)

Configuration graph
- node: cell
- edge: successor
- pointers: x, y, z, #
Transitions

\[ x = y? \]

\[ z \]

\[ t \]
Transitions

\[ x = y? \]

\[
\begin{array}{c}
\text{z} \\
\text{t}
\end{array}
\]

Parosh Aziz Abdulla (Uppsala University)
Shape Analysis via Monotonic Abstraction
February 9, 2010 10 / 33
Transitions

\[ x = y? \]

\[ x \rightarrow y \]

\[ z \rightarrow z \]

\[ t \]

\[ x = y? \]

\[ x \rightarrow y \]

\[ z \rightarrow z \]
Transitions

\[ x = y? \]

\[ x \quad y \]

\[ z \]

\[ \rightarrow \]

\[ t \]

\[ x = y? \]

\[ x \quad y \]

\[ z \]

\[ \rightarrow \]

\[ \cdot \cdot \cdot \quad \times \quad \rightarrow \]
Transitions

$x \neq y$?
Transitions

\[ x \neq y? \]

\[ x \quad y \]

\[ z \]

\[ x \quad y \]

\[ z \]
Transitions

\[ x \neq y? \]

[Diagram]

\[ x \neq y? \]

[Diagram]
Transitions

\[ x \neq y? \]

\[ x \rightarrow y \]

\[ z \]

\[ x \rightarrow y \]

\[ z \]

\[ x \neq y? \]

\[ x \rightarrow y \]

\[ z \]

\[ x \rightarrow y \]

\[ z \]
Transitions

\[ y := x \]

\[ x \quad y \]

\[ z \]
Transitions

\[ y := x \]

\[ x \quad y \]

\[ z \]

\[ x \quad y \]

\[ z \]
Transitions

\[ y := x \]

\[ x \rightarrow y \]

\[ z \]

\[ y := x \cdot \text{next} \]

\[ x \rightarrow y \]

\[ z \]

\[ x \rightarrow y \rightarrow z \]

\[ x \rightarrow y \rightarrow z \]
Transitions

\[ y := x \]

\[ y := x \cdot \text{next} \]
Transitions

\[ x \cdot \text{next} := y \]
Transitions

\[ x \cdot \text{next} := y \]
Ordering on Graphs

Variable Deletion
Ordering on Graphs

Variable Deletion

Variable Deletion

x \rightarrow y

z
Ordering on Graphs

Variable Deletion

Variable Deletion

\[
\begin{array}{c}
\text{Variable Deletion} \\
\begin{array}{c}
\text{ } \quad \text{ } \quad \text{ } \\
x \quad y \quad \text{ } \\
\downarrow \quad \downarrow \\
z \quad \text{ } \\
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{ } \quad \text{ } \quad \text{ } \\
\text{ } \quad \text{ } \quad \text{ } \\
\downarrow \quad \downarrow \\
z \quad \text{ } \\
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{ } \quad \text{ } \quad \text{ } \\
\text{ } \quad \text{ } \quad \text{ } \\
\downarrow \quad \downarrow \\
z \quad \text{ } \\
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{ } \quad \text{ } \quad \text{ } \\
\text{ } \quad \text{ } \quad \text{ } \\
\downarrow \quad \downarrow \\
z \quad \text{ } \\
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{ } \quad \text{ } \quad \text{ } \\
\text{ } \quad \text{ } \quad \text{ } \\
\downarrow \quad \downarrow \\
z \quad \text{ } \\
\end{array}
\end{array}
\]
Ordering on Graphs

Edge Deletion

Edge Deletion

x \rightarrow y

z \rightarrow x \leftrightarrow y

z

Parosh Aziz Abdulla (Uppsala University)

Shape Analysis via Monotonic Abstraction

February 9, 2010
Ordering on Graphs

Edge Deletion

\[ x \rightarrow z \rightarrow y \]

\[ x \rightarrow z \rightarrow y \]

\[ \succ \]
Ordering on Graphs

Vertex Deletion

**Isolated Vertex**
- no label
- no incoming/outgoing arcs

**Vertex Deletion**

\[ x \quad y \]

\[ \begin{array}{c}
\bullet \\
\downarrow \\
\bullet
\end{array} \quad \begin{array}{c}
\bullet \\
\downarrow \\
\bullet
\end{array} \]

\[ \begin{array}{c}
\bullet \\
\downarrow \\
\bullet
\end{array} \quad \begin{array}{c}
\bullet
\end{array} \]
Ordering on Graphs

Vertex Deletion

Isolated Vertex
- no label
- no incoming/outgoing arcs

Vertex Deletion

\[
\begin{align*}
\text{Before:} & \quad \text{After:} \\
\begin{array}{c}
\bullet x \\
\downarrow \\
\bullet z \\
\end{array} & \quad \begin{array}{c}
\bullet x \\
\downarrow \\
\bullet z \\
\end{array} \\
\begin{array}{c}
\bullet y \\
\end{array} & \quad \begin{array}{c}
\bullet y \\
\end{array}
\end{align*}
\]
Ordering on Graphs

SimpleVertex
- no label
- one incoming arc
- one outgoing arc

Contraction

\[ x \trianglerighteq y \]

\[
\begin{array}{c}
\text{SimpleVertex} \\
\text{no label} \\
\text{one incoming arc} \\
\text{one outgoing arc}
\end{array}
\]
Ordering on Graphs

Contraction

SimpleVertex
- no label
- one incoming arc
- one outgoing arc

Contraction

$\leq$

$x$ $y$

$z$

$x$ $y$

$z$
Bad Configurations

Well-formed Lists

Well-Formed List:

Badly-Formed Lists:
Bad Configurations

Well-formed Lists

Bad Patterns:
- minimal elements
- finitely many
- upward closure = all badly-formed lists
Bad Configurations

Well-formed Lists
Bad Configurations

Well-formed Lists

Bad pattern

\[ \leq \]

Bad configuration
Bad Configurations

Well-formed Lists

Bad pattern

Bad configuration

<
Bad Configurations

Well-formed Lists

Bad pattern

\( \preceq \)

Bad configuration
Bad Configurations

Well-formed Lists
Bad Configurations

Well-formed Lists

Bad pattern

Bad configuration

Parosh Aziz Abdulla (Uppsala University)
Shape Analysis via Monotonic Abstraction
February 9, 2010 21 / 33
Bad Configurations

Well-formed Lists

Bad pattern

\[ \times \rightarrow \# \]

Bad configuration

\[ \times \rightarrow \# \rightarrow \bullet \]
Bad Configurations

Well-formed Lists

Bad pattern

Bad configuration
Bad Configurations

Backward Reachability Analysis

$G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow G_3 \Rightarrow G_4 \Rightarrow G_5$

symbolic representation = graphs

WQO implies termination
Bad Configurations

Backward Reachability Analysis

\[ G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow G_0 \]

symbolic representation = graphs

WQO implies termination
Backward Reachability Analysis

\[ G_5 \] \[ \Rightarrow \] \[ G_2 \] \\
[ \Rightarrow ] \[ G_4 \] \[ \Rightarrow \] \[ G_0 \] \\
[ \Rightarrow ] \[ G_3 \] \[ \Rightarrow \] \[ G_1 \] \\

\( \leq \) symbolic representation = graphs \\
\( WQO \) implies termination
Backward Reachability Analysis

\[ G_5 \Rightarrow G_2 \Rightarrow G_1 \Rightarrow G_0 \]

Symbolic representation = graphs

WQO implies termination
Backward Reachability Analysis

\[ G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow G_3 \Rightarrow G_4 \Rightarrow G_5 \]

symbolic representation = graphs

WQO implies termination
Backward Reachability Analysis

\[
G_5 \Rightarrow G_4 \Rightarrow G_3 \Rightarrow G_2 \Rightarrow G_1 \Rightarrow G_0
\]

\[\text{symbolic representation} = \text{graphs}\]

WQO implies termination
Backward Reachability Analysis

\[ G_5 \xrightarrow{\text{}} G_2 \xleftarrow{\text{}} G_1 \xrightarrow{\text{}} G_0 \]

\[ G_4 \xrightarrow{\text{}} G_2 \xleftarrow{\text{}} G_0 \]

\[ G_3 \xrightarrow{\text{}} G_2 \xleftarrow{\text{}} G_0 \]

Symbolic representation = graphs

WQO implies termination
Backward Reachability Analysis

symbolic representation = graphs

\[ G_3 \preceq G_4 \preceq G_5 \]

\[ G_1 \preceq G_2 \preceq G_0 \]

\[ \leq \] WQO implies termination
Computing predecessors

Testing Equality: $x = y$?
Computing predecessors

Testing Equality: \( x = y \)?
Computing predecessors

Testing Equality: $x = y$?
Computing predecessors

Testing Equality: $x = y$?
Computing predecessors

Testing Equality: $x = y$?
Computing predecessors

Testing Equality: $x = y$?
Computing predecessors

Testing Equality: $x = y$?
Computing predecessors

Testing Equality: $x = y$?
Computing predecessors

Testing Equality: $x = y$?
Computing predecessors

Testing Equality: $x = y$?
Computing predecessors

\[ x := y \cdot \text{next} \]
Computing predecessors

\[ x := y \cdot \text{next} \]
Computing predecessors

\[ x := y \cdot \text{next} \]
Computing predecessors

\( x := y \cdot next \)
Computing predecessors

\[ x := y \cdot \text{next} \]
Computing predecessors

\[ x := y \cdot next \]
Computing predecessors

\[ x := y \cdot \text{next} \]
Computing predecessors

\[ x := y \cdot \text{next} \]
Degree

\[ \text{deg}(G) := \# \text{ unlabeled leafs} \]
WQO

Degree

Degree

$deg(G) := \# \text{ unlabeled leafs}$

Example: $deg(G) = 4$
Block

maximal subgraph which is connected
WQO
Block

Block
maximal subgraph which is connected

Example: Two blocks

\[
\begin{align*}
\text{Example: Two blocks} \\
\end{align*}
\]
WQO

Proof

\( \preceq \) WQO:

- \( g_1 \rightsquigarrow g_2 \) implies \( \deg(g_1) \geq \deg(g_2) \)
- In back reachability scheme:
  - generated graphs have bounded degree
  - contain finitely many types of blocks (modulo contraction)
  - each graph can be encoded by a vector of multisets of vectors of natural numbers!
- \( \preceq \) WQO by Higman’s lemma.
### Experiments

<table>
<thead>
<tr>
<th>Prog.</th>
<th>Prop.</th>
<th>Time</th>
<th>#Cons.</th>
<th>#Iter.</th>
<th>Prog.</th>
<th>Prop.</th>
<th>Time</th>
<th>#Cons.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concat</td>
<td>Deref</td>
<td>0.4 s</td>
<td>7</td>
<td>3</td>
<td>Delete</td>
<td>Deref</td>
<td>0.4 s</td>
<td>8</td>
</tr>
<tr>
<td>Fumble</td>
<td>Deref</td>
<td>0.3 s</td>
<td>3</td>
<td>2</td>
<td>Reverse</td>
<td>Deref</td>
<td>0.3 s</td>
<td>2</td>
</tr>
<tr>
<td>Walk</td>
<td>Deref</td>
<td>0.4 s</td>
<td>9</td>
<td>3</td>
<td>Zip</td>
<td>Deref</td>
<td>1.9 s</td>
<td>206</td>
</tr>
<tr>
<td>Fumble</td>
<td>Garbage</td>
<td>0.7 s</td>
<td>38</td>
<td>14</td>
<td>Reverse</td>
<td>Garbage</td>
<td>0.8 s</td>
<td>55</td>
</tr>
<tr>
<td>Reverse</td>
<td>Well-form.</td>
<td>1.7 s</td>
<td>48</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>