Simplifying Verification of Unbounded Structures

Alexander Dominik Weinert

RWTH Aachen University

September 3, 2012
Model Checking on Unbounded Structures

Model Checking

Verification by exploration of states

<table>
<thead>
<tr>
<th>Require: int $i_1$, int $i_2$</th>
<th>Require: List $l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{while } i_1 \neq 0 \text{ do}$</td>
<td>$\text{Element } e := l \cdot \text{first}()$</td>
</tr>
<tr>
<td>$i_1 := i_1 - 1$</td>
<td></td>
</tr>
<tr>
<td>$i_2 := i_2 + 1$</td>
<td></td>
</tr>
<tr>
<td>$\text{end while}$</td>
<td>$\text{while } e \neq \text{null do}$</td>
</tr>
<tr>
<td></td>
<td>$e := l \cdot \text{next}(e)$</td>
</tr>
<tr>
<td></td>
<td>$\text{end while}$</td>
</tr>
</tbody>
</table>

Finite number of states $\checkmark$  Infinite number of states $\times$

Idea [Heinen et al., 2009]

Model heap configurations as graphs

Use **Heap Abstraction Grammars (HAG)** $\Rightarrow$ Finitely many heap states
Workflow in the Juggrnaut-Framework

Current Workflow

Desired Workflow

Program

Specification

HAG

Verifier

Yes/No

Program

Inputs

Specification

Inference

HAG

Verifier

Yes/No
Inference

Program → Inputs

Heap Configurations

Execute program

Summarize recursion

Intermediate HAG

Minimization

HAG
Greedy Summarization

**Minimization Problem**

Given: Set $S$ of rules $X \rightarrow Y$.
Task: Find “cheaper” set that describes the same set of graphs.

**Solution: Minimum Description Length [Rissanen, 1978]**

Define gain function for each subgraphs and cost function for rules:

$$gain(G) = gain(S \mid Y \rightarrow G) - cost(Y \rightarrow G)$$

Finding optimal graph for minimization

$\Leftrightarrow$

Maximization of gain function over all subgraphs

Cost- and Gain-functions are heuristics
Problem: Minimum Description Length: enumeration over all subgraphs
⇒ Exponential runtime

Solution [Jonyer et al., 2002]: Grow only connected subgraphs.
Inference (Reminder)

Program → Inputs → Execute program → Heap Configurations → Summarize recursion → Intermediate HAG → Minimization → HAG
Recursive Data Structures

Conditions for Recursion

A data structure is recursive, if

- it is found in at least two places,
- these two embeddings overlap and
- a concatenation of these embeddings is itself a subgraph.
Representation of Recursive Data Structures

When finding a recursive data structure:

- Add rule for concatenations of arbitrary length
- Add rule for stopping concatenation
- Remove concatenated structure from original graph and replace it with new nonterminal

Example in String Case

String under consideration:

\[xyzabcabcyxx\]

\[
\begin{align*}
\text{Rule for concatenations:} & \quad X &\rightarrow& \ abcX \\
\text{Rule for stopping:} & \quad X &\rightarrow& \ abc \\
\text{Resulting string:} & \quad xyzXzxx
\end{align*}
\]
Results for Singly Linked List

Input: Singly linked lists with 25 to 200 nodes

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Subgraphs [ms]</th>
<th>Complete [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>90</td>
<td>102</td>
</tr>
<tr>
<td>50</td>
<td>285</td>
<td>305</td>
</tr>
<tr>
<td>75</td>
<td>437</td>
<td>500</td>
</tr>
<tr>
<td>100</td>
<td>642</td>
<td>682</td>
</tr>
<tr>
<td>125</td>
<td>1 001</td>
<td>1 040</td>
</tr>
<tr>
<td>150</td>
<td>1 455</td>
<td>1 526</td>
</tr>
<tr>
<td>175</td>
<td>1 884</td>
<td>2 000</td>
</tr>
<tr>
<td>200</td>
<td>2 895</td>
<td>3 028</td>
</tr>
</tbody>
</table>
Results for Singly Linked Nested Lists

Input: Singly linked nested lists

<table>
<thead>
<tr>
<th>Outer</th>
<th>Inner</th>
<th>Inner [ms]</th>
<th>Outer [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>31</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>394</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1280</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>96</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>2701</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>51608</td>
<td>23</td>
</tr>
</tbody>
</table>
Thank you for your attention

www.mpi-sws.org/~aweinert
alexander.weinert@rwth-aachen.de
Juggrnaut: Graph Grammar Abstraction for Unbounded Heap Structures. 

Jonyer, I., Holder, L. B., and Cook, D. J. (2002). 
Concept Formation Using Graph Grammars. 
In Proceedings of the KDD Workshop on Multi-Relational Data Mining.

Modeling by shortest data description. 

Inferring Heap Abstraction Grammars. 
Bachelor’s Thesis, RWTH Aachen University, Aachen.