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

Require: int i_1, int i_2
while $i_1 \neq 0$ **do**
$i_1 := i_1 - 1$
$i_2 := i_2 + 1$
end while

Finite number of states ✓

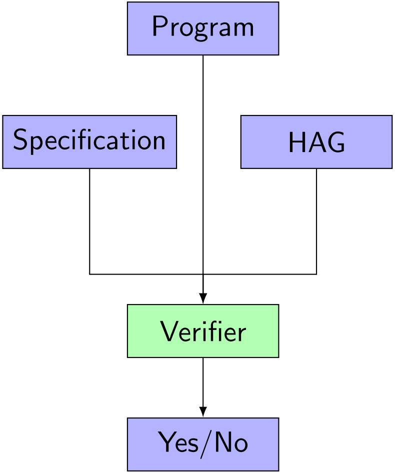
Require: List l
Element $e := l.\mathit{first}()$
while $e \neq \mathit{null}$ **do**
$e := l.\mathit{next}(e)$
end while

Infinite number of states χ

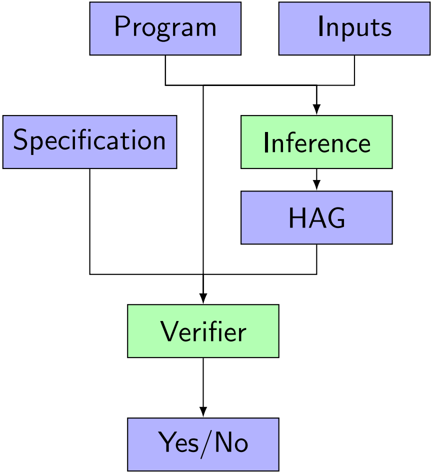
Idea [Heinen et al., 2009]

Model heap configurations as graphs
Use *Heap Abstraction Grammars (HAG)* $\Rightarrow$ Finitely many heap states

Workflow in the Juggrnaut-Framework

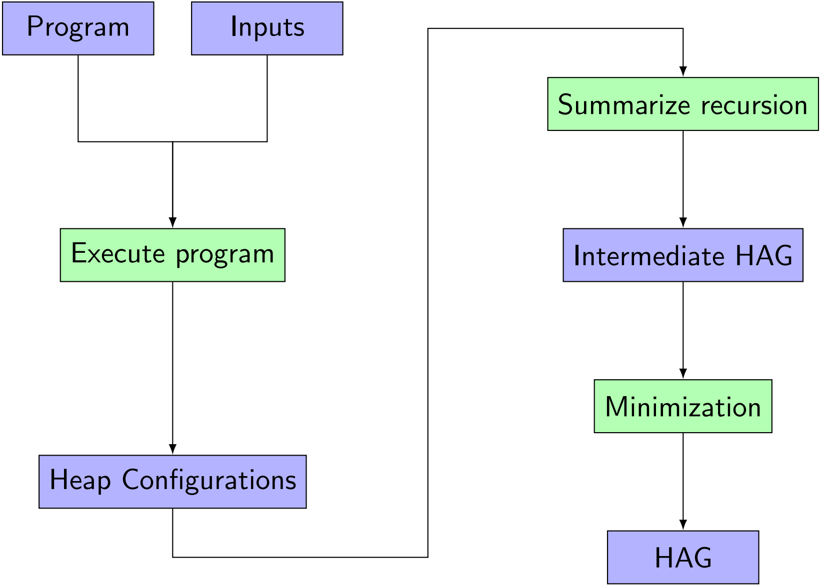


Current Workflow



Desired Workflow

Inference


Program
Inputs
Execute program
Heap Configurations
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: Mimimum 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

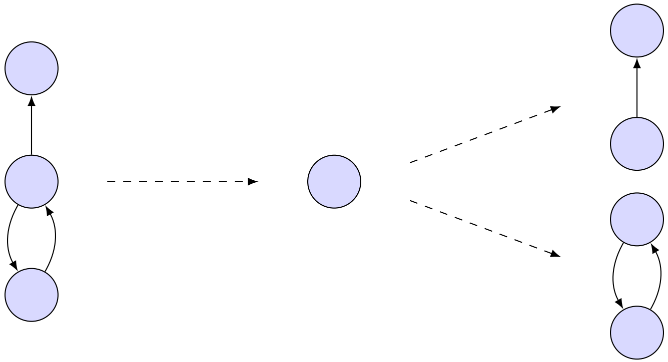
Cost- and Gain-functions are heuristics

Enumeration of Subgraphs

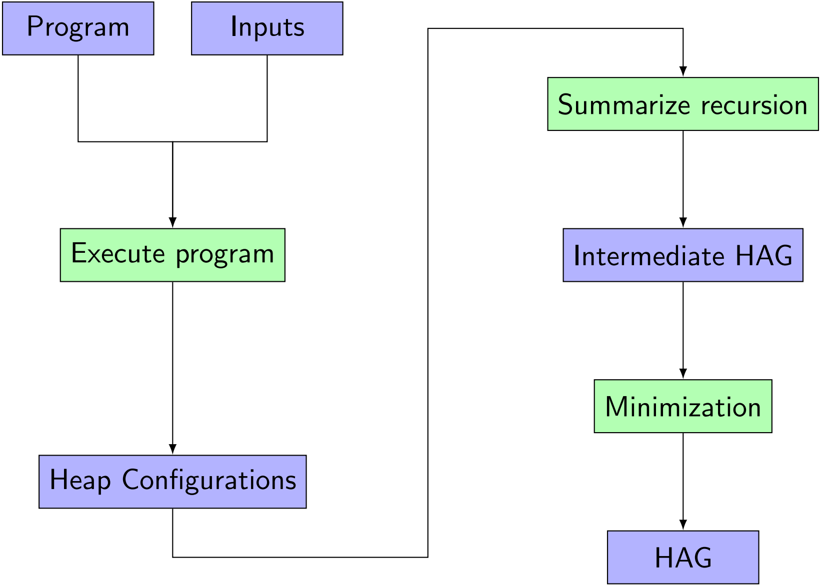
Miminum-Description-Length-approach

Problem: Minimum Description Length: enumeration over all subgraphs
$\Rightarrow$ Exponential runtime

Solution [Jonyer et al., 2002]: Grow only connected subgraphs.



Inference (Reminder)

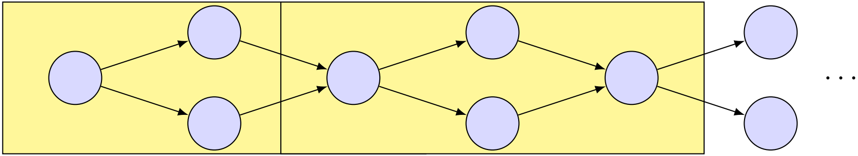


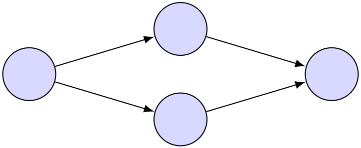
Recursive Data Structures

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.





Recursive Data Structures (cont.)

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:

$$xyzabcabcyxx \begin{cases} \text{Rule for concatenations:} & X \rightarrow abcX \\ \text{Rule for stopping:} & X \rightarrow abc \\ \text{Resulting string:} & xyzXzxx \end{cases}$$

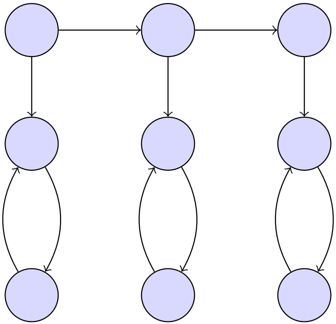
Results for Singly Linked List

Input: Singly linked lists with 25 to 200 nodes

Nodes	Subgraphs [ms]	Complete [ms]
25	90	102
50	285	305
75	437	500
100	642	682
125	1 001	1 040
150	1 455	1 526
175	1 884	2 000
200	2 895	3 028

Results for Singly Linked Nested Lists

Input: Singly linked nested lists



Outer	Inner	Inner [ms]	Outer [ms]
2	2	9	8
2	4	19	5
4	2	31	16
4	4	394	11
4	5	1 280	20
5	2	96	23
5	4	2 701	22
5	5	51 608	23

Thank you for your attention

www.mpi-sws.org/∼aweinert

Heinen, J., Noll, T., and Rieger, S. (2009).
Juggrnaut: Graph Grammar Abstraction for Unbounded Heap Structures.
In Johnsen, E. B. and Stolz, V., editors, *Harnessing Theories for Tool Support in Software, Preliminary Proceedings*, pages 53–67. United Nations University - International Institute for Software Technology.

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*.

Rissanen, J. (1978).
Modeling by shortest data description.
Automatica, 14(5):465–471.

Weinert, A. (2012).
Inferring Heap Abstraction Grammars.
Bachelor's Thesis, RWTH Aachen University, Aachen.