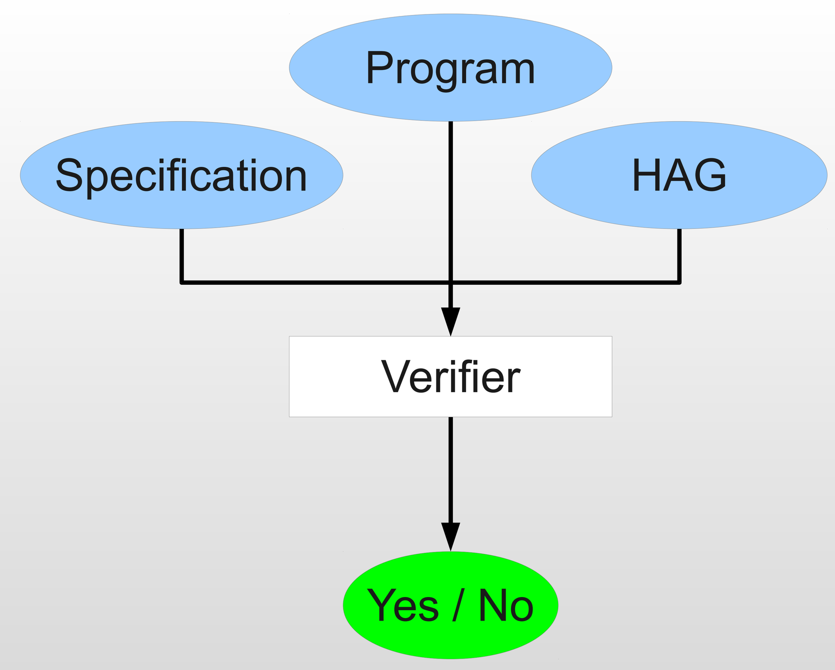
Simplifying Verification of Unbounded Structures

Alexander Weinert

RWTH Aachen University

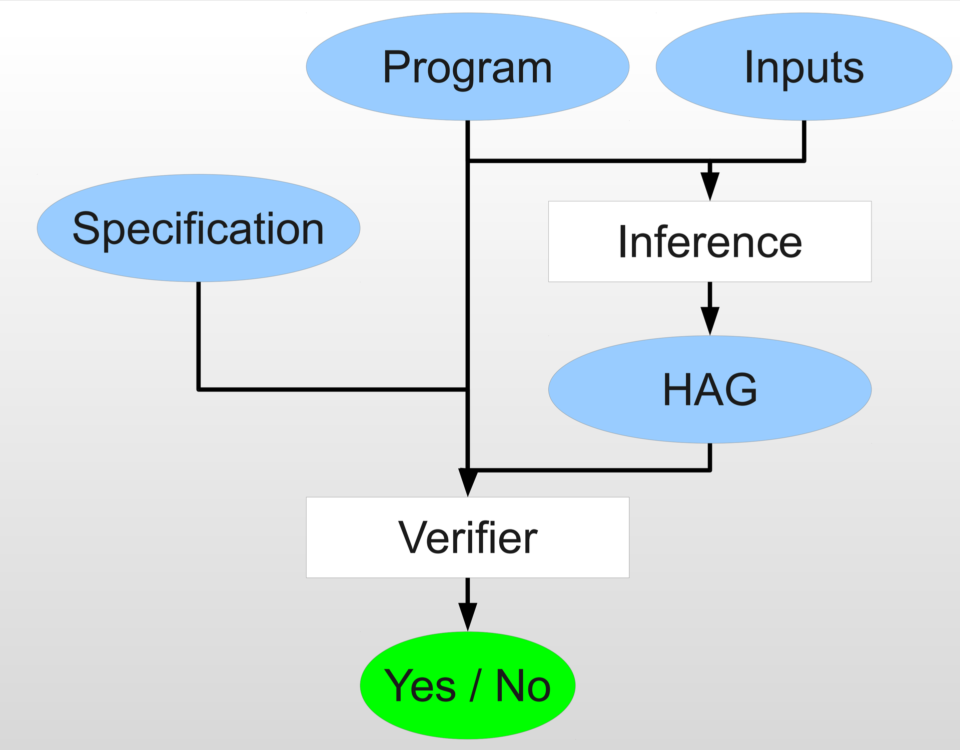
Motivation

- Traditional Model Checking enumerated all states of a computation.
- **Problem:** Fails when operating on unbounded heap memory.
- **Solution:** Describe heap states as graphs, describe set of heap states as finite Heap Abstraction Grammar (HAG).



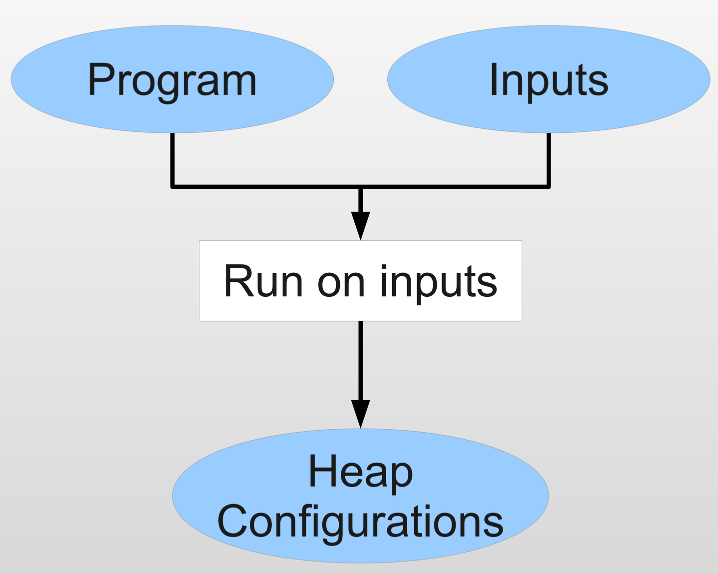
Current workflow for verification of unbounded heap structures

- **New Problem:** Human verifier has to provide HAG. Can be unintuitive and cumbersome.
- **Goal of this work:** Given a program and some inputs, automatically produce (*infer*) the HAG.



Desired workflow for verification of unbounded heap structures

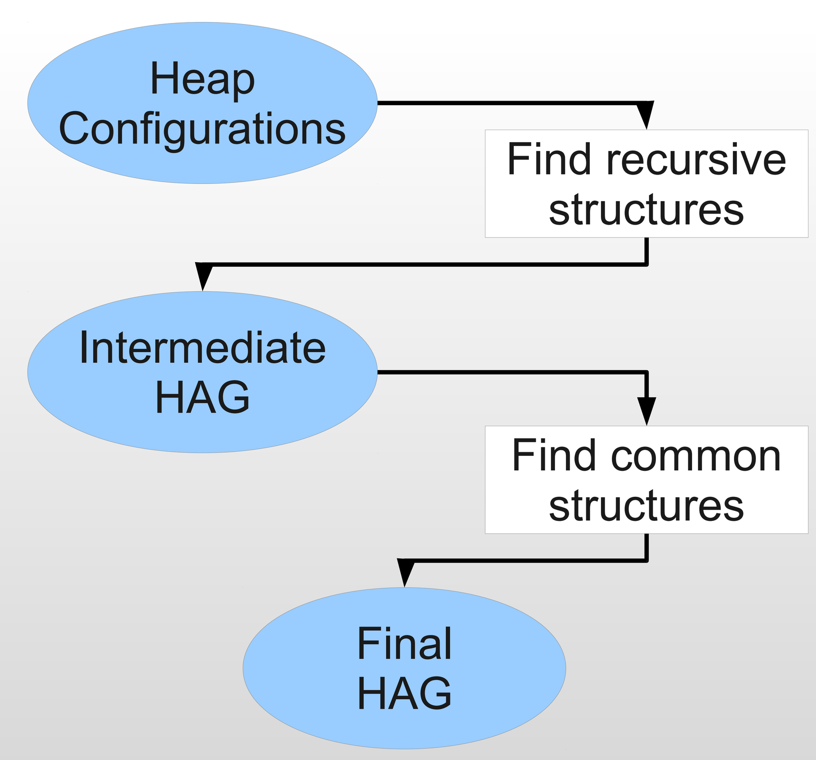
Inference



Producing a set of heap configurations

From a Program to Heap Configurations

- First step: Produce a set of heap configurations from the given program and its inputs.
  1. Execute program on each of its inputs.
  2. Save Heap Configuration after every step of execution.



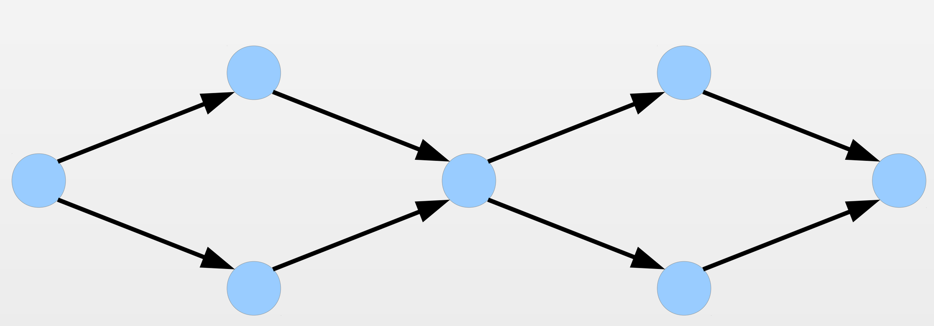
Producing a HAG from a set of heap configurations

From Heap Configurations to Grammar

- From a set of heap configurations, produce a grammar that describes *at least* the given set.
  1. Summarize substructures that serve as evidence for a recursive datastructure.
  2. Summarize structures that occur at multiple points in the configurations.

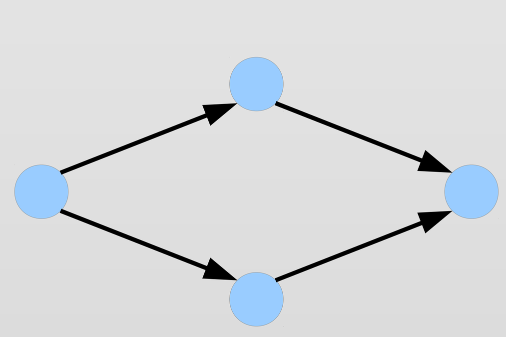
Recursion

- Generalize observed behaviour of program by finding evidence for recursive data structure.
  1. Find the same structure at two *overlapping* points in the graph.



Complete Structure

2. Check that complete structure is a *concatenation* of the substructure.

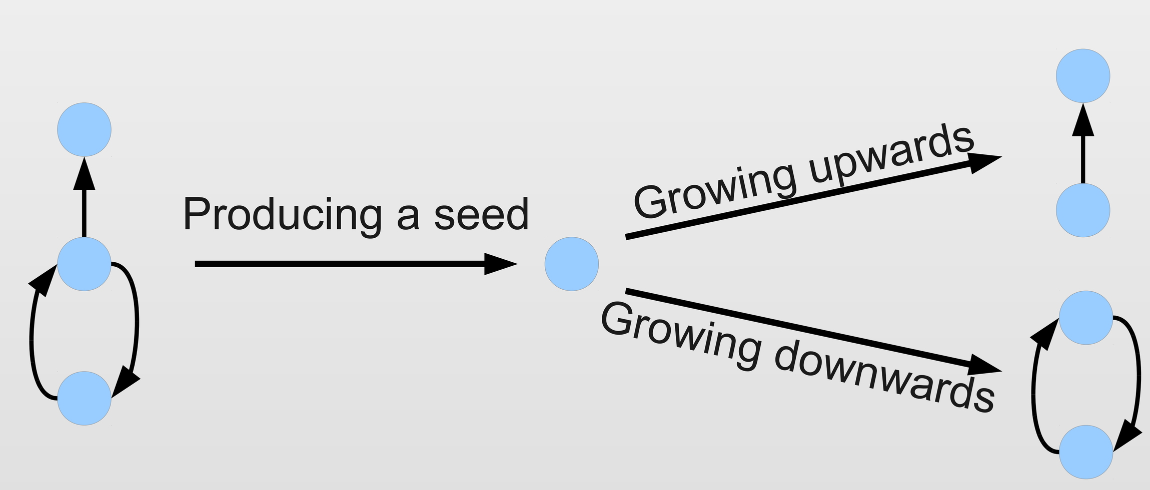


Substructure

3. Summarize all other occurrences of this structure in the graphs.
4. Add rules for concatenation of the structure of arbitrary length, analogous to string case.

Common Structures

- Minimize intermediate HAG by finding structures occurring at multiple points in the grammar
- **Problem:** Checking *all* subgraphs takes exponential time.
- **Solution:** Grow only connected subgraphs



First steps of growing subgraphs

- **Problem:** How to choose the subgraph to be summarized?
- **Solution:**
  1. Define gain function that describes the space saved by a single summary.
  2. Maximize that function.

Results

The resulting grammars for singly linked lists, circular singly linked lists and nested lists are the same as those written by a human verifier.

Nodes	Growing [ms]	Complete [ms]
50	285	305
100	642	682
200	2895	3028

Time needed for inferring a grammar for singly linked, non-circular lists

Out	In	Inner [ms]	Outer [ms]
2	4	31	16
4	4	394	11
5	4	2701	22

Time needed for inferring a grammar for nested lists. Out and In denote the number of outer and inner nodes, respectively

Contact

Alexander Dominik Weinert,
Email: alexander.weinert@rwth-aachen.de
Online: www.mpi-sws.org/~aweinert

References

- A. Weinert (2012). Inferring Heap Abstraction Grammars. Unpublished Bachelor's Thesis. RWTH Aachen University, Aachen.
- I. Jonyer, L. Holder, D. Cook (2004). MDL-Based Context-Free Graph Grammar Induction And Applications. International Journal on Artificial Intelligence Tools, 13(1):65 – 79.
- J. Heinen, T. Noll, S. Rieger (2009). Juggrnaut: Graph Grammar Abstraction for Unbounded Heap Structures. Harnessing Theories for Tool Support in Software, 53 - 67.
- J. Heinen, C. Jansen, J.-P. Katoen, T. Noll (tbp). Juggrnaut: Using Graph Grammars for Abstracting Unbounded Heap Structures