Static Analysis of Dynamically Allocated Data

Vini Kanvar

under the guidance of

Prof. Uday P. Khedker

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

April 2016

Static Analysis of Dynamically Allocated Data

- Some questions that my work can help answer:
  - Is there a memory leak in the program?
  - Is there a null dereference?
  - What is the shape (tree, DAG, linked list) of a heap data structure?
    - for program understanding, verification, debugging.

Challenges in Heap Analysis

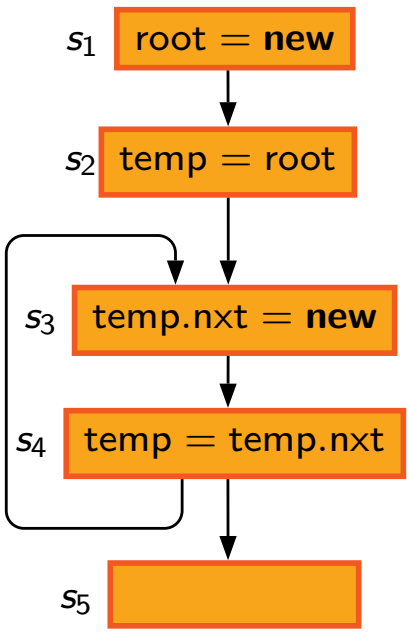
- Challenges in the analysis of pointer variables
  - Undecidable [Chakaravarthy 2003][Ramalingam 1994]
  - Features: dynamic typing, implicit casting, pointer arithmetic, etc.

Challenges in Heap Analysis

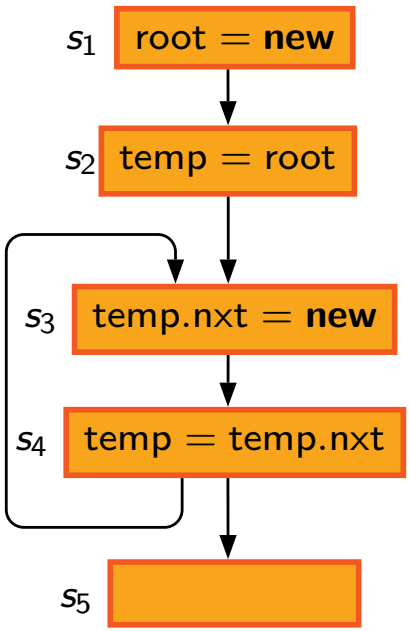
- Challenges in the analysis of pointer variables
  - Undecidable [Chakaravarthy 2003][Ramalingam 1994]
  - Features: dynamic typing, implicit casting, pointer arithmetic, etc.
- Challenges in the analysis of heap pointer variables
  - Unpredictable lifetime
  - Unbounded number of allocations
  - Unnamed locations

Conventional vs. Our Static Heap Analysis

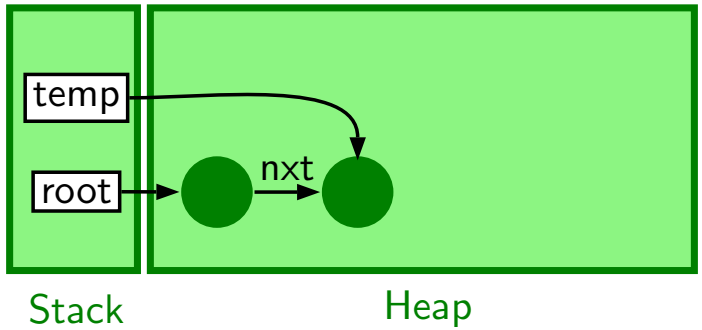
Program creates a linear linked list.



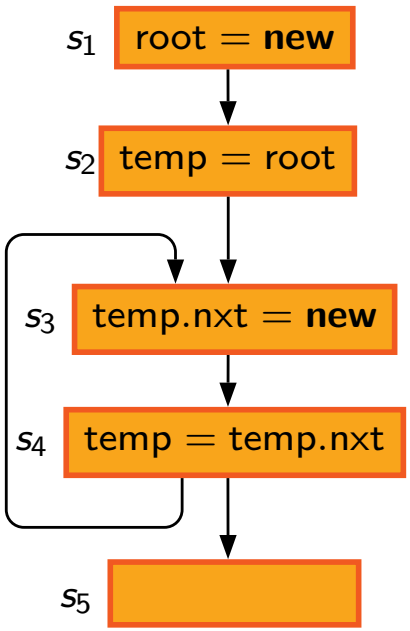
Conventional vs. Our Static Heap Analysis



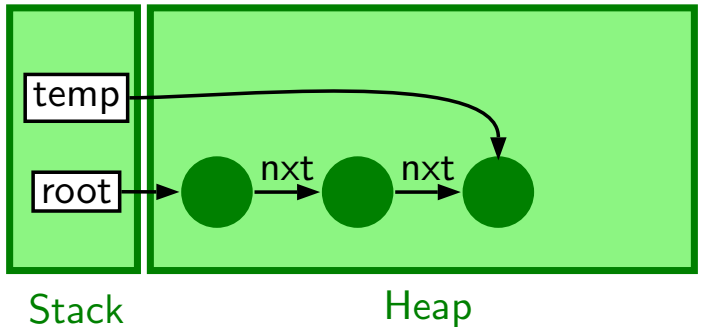
Possible runtime memory graphs at end of program



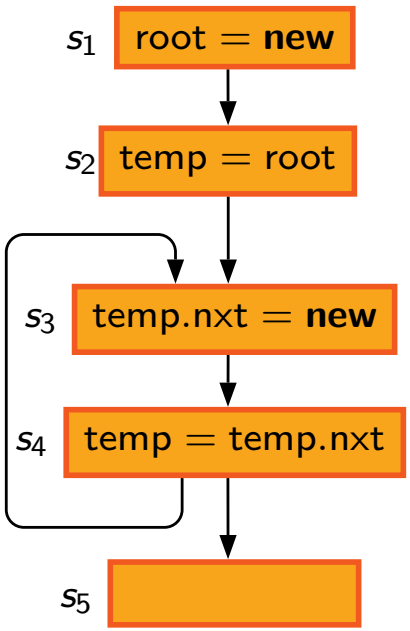
Conventional vs. Our Static Heap Analysis



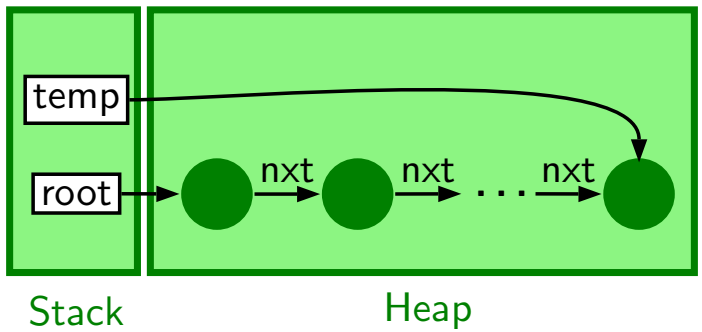
Possible runtime memory graphs at end of program



Conventional vs. Our Static Heap Analysis

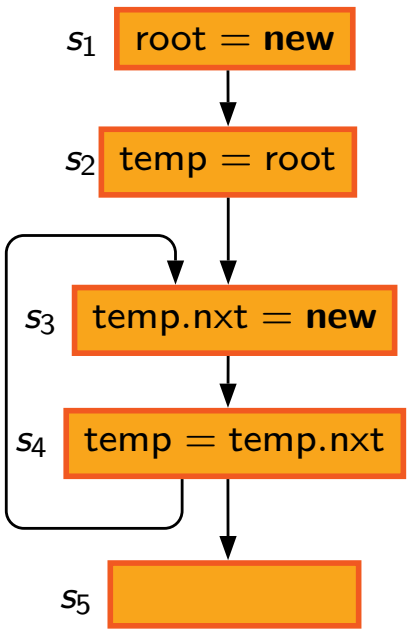


Possible runtime memory graphs at end of program

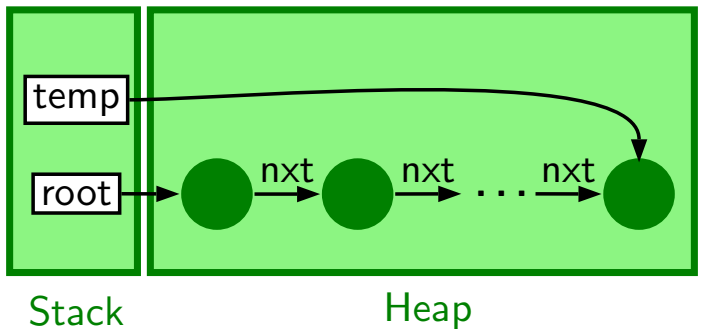


Conventional vs. Our Static Heap Analysis

root points to unbounded number of heap locations.
temp does not; it points to the end of the list.



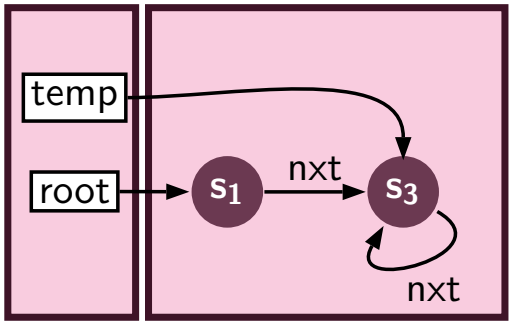
Possible runtime memory graphs at end of program

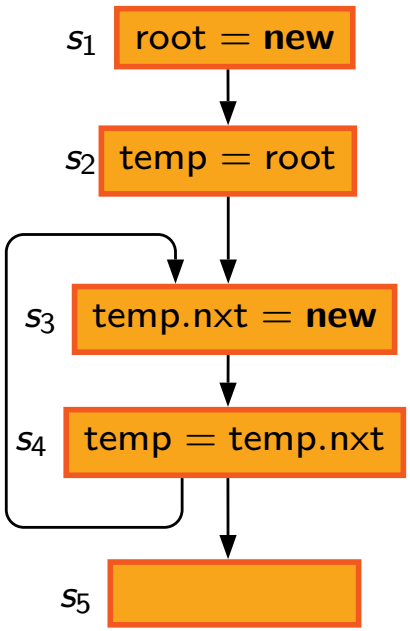


Conventional vs. Our Static Heap Analysis

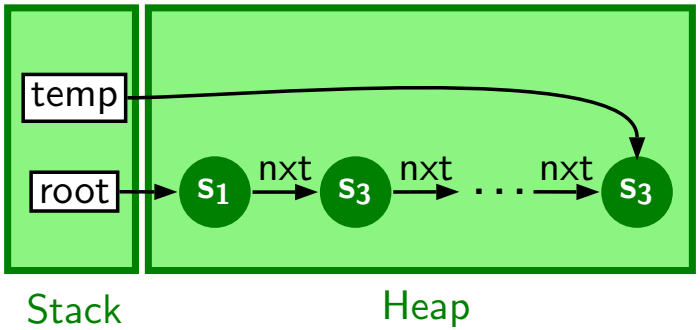
Conventional static analysis: merges all runtime memory graphs based on the allocation sites.

Static heap analysis
Conventional





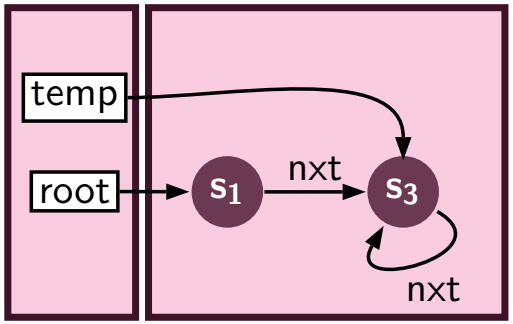
Possible runtime memory graphs at end of program

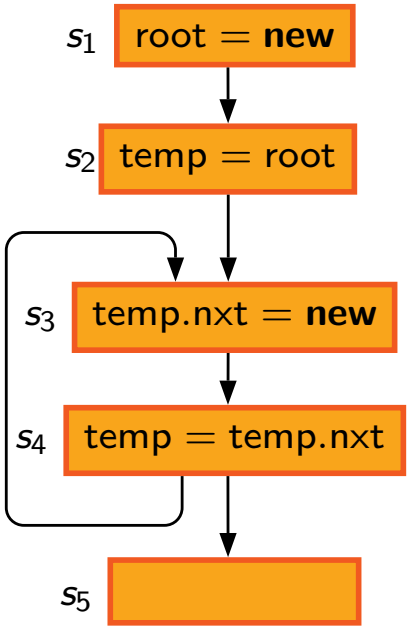


Conventional vs. Our Static Heap Analysis

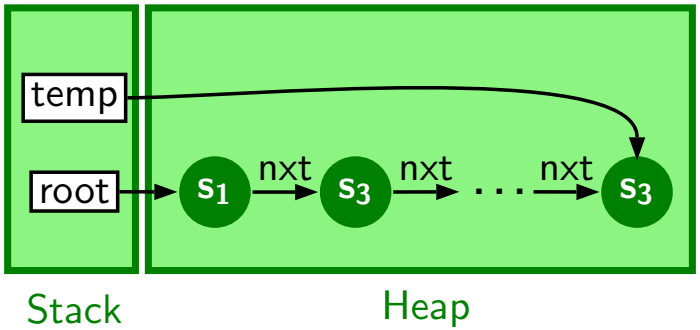
Cycle denotes unbounded number of heap locations.

Static heap analysis
Conventional



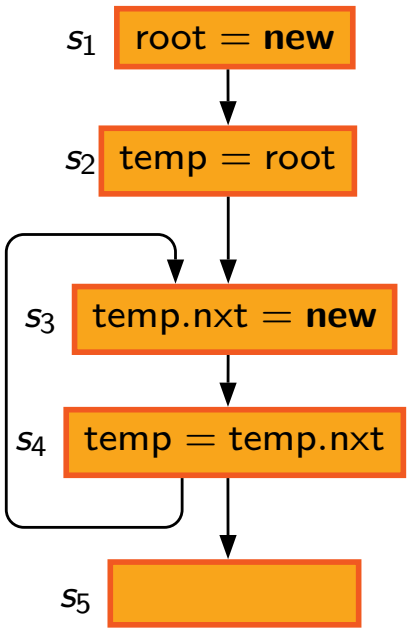


Possible runtime memory graphs at end of program

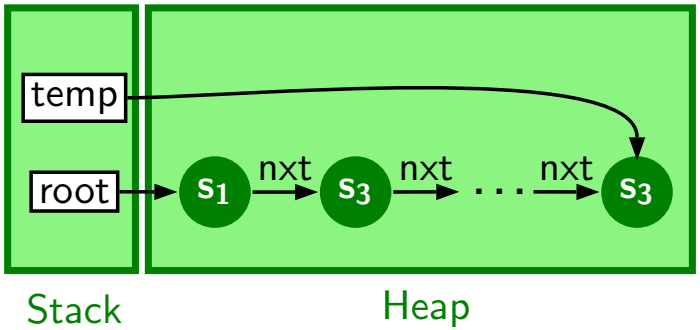


Conventional vs. Our Static Heap Analysis

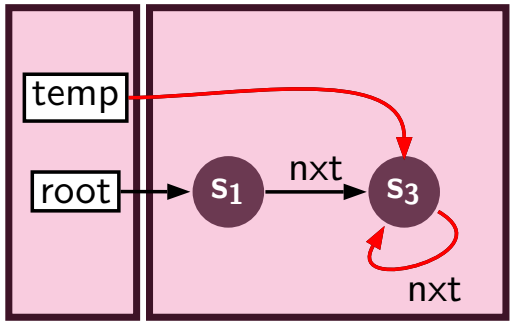
Conventional static analysis: spuriously computes that **temp** points to an unbounded number of heap locations.



Possible runtime memory graphs at end of program

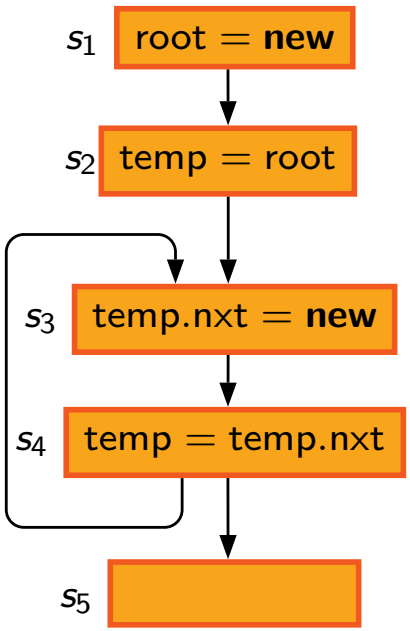


Static heap analysis
Conventional

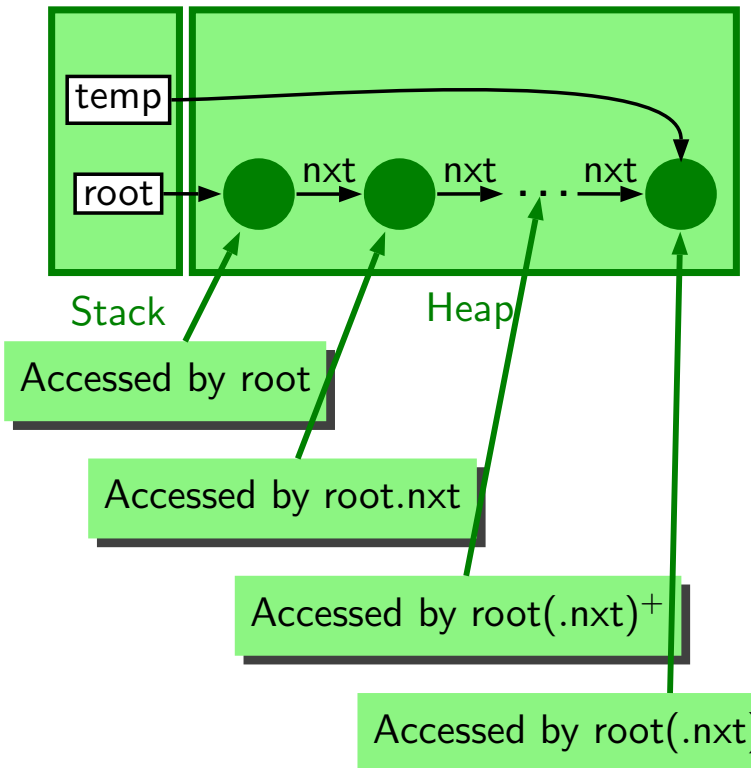


Conventional vs. Our Static Heap Analysis

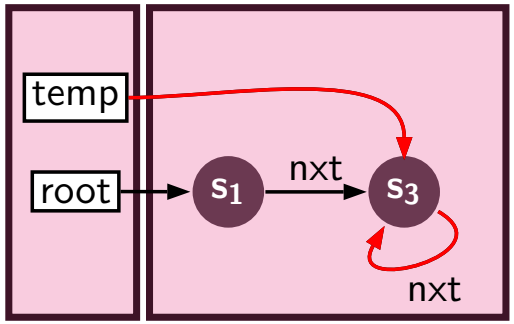
Our static analysis: we propose to merge all runtime memory graphs based on program accesses.



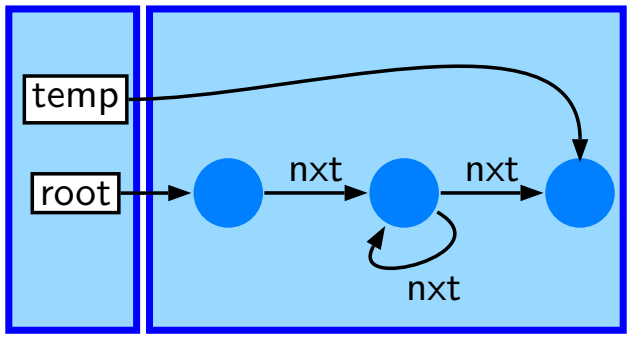
Possible runtime memory graphs at end of program



Static heap analysis
Conventional

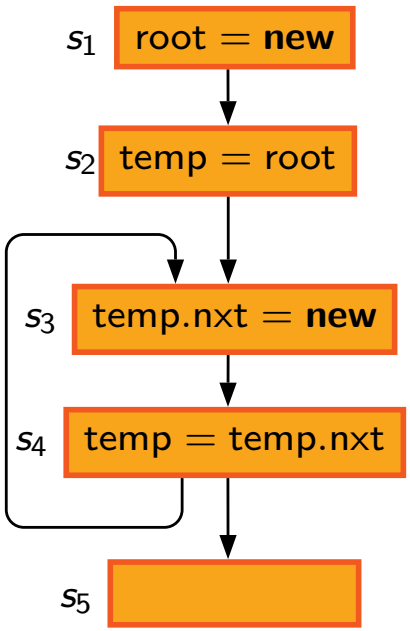


Static heap analysis
Our proposal

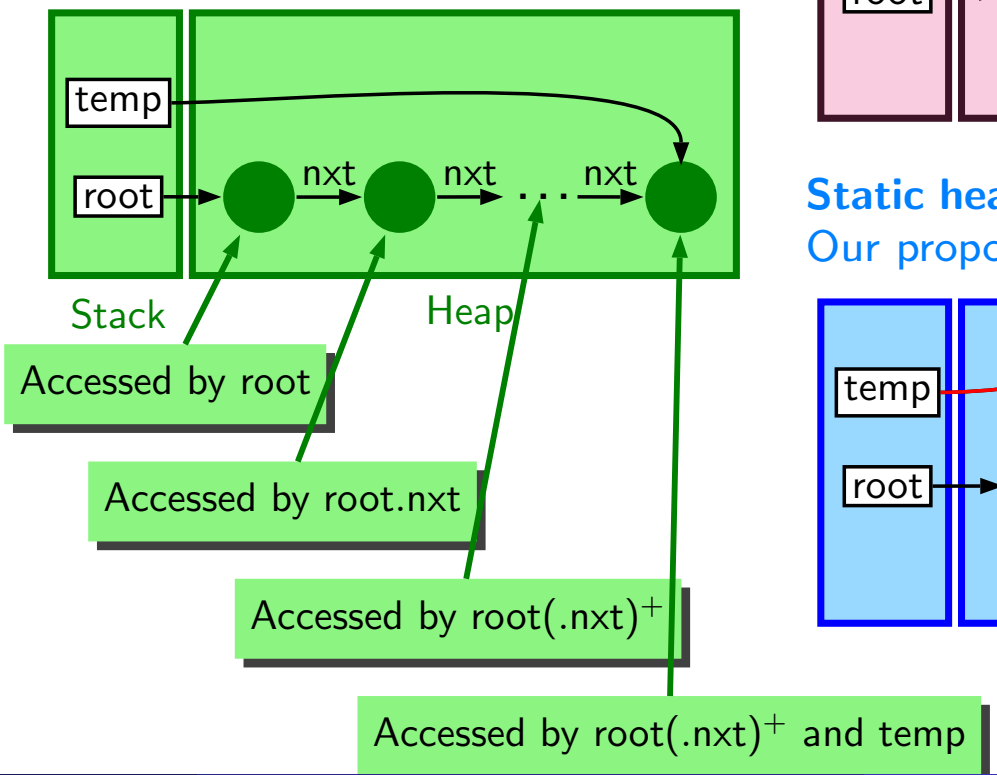


Conventional vs. Our Static Heap Analysis

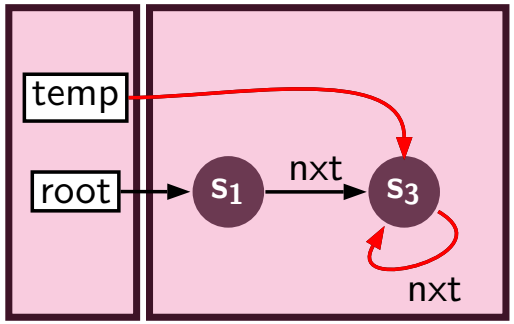
Our static analysis: we precisely compute that **temp** points to a single heap location only.



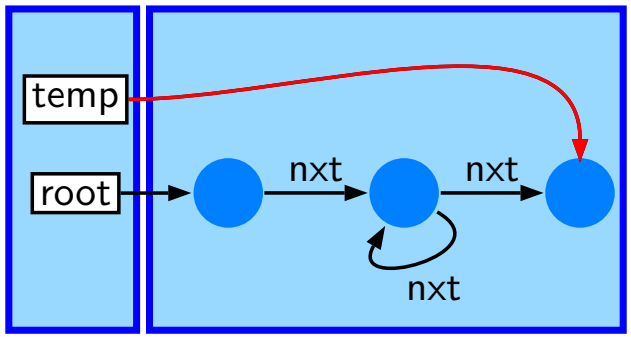
Possible runtime memory graphs at end of program



Static heap analysis
Conventional

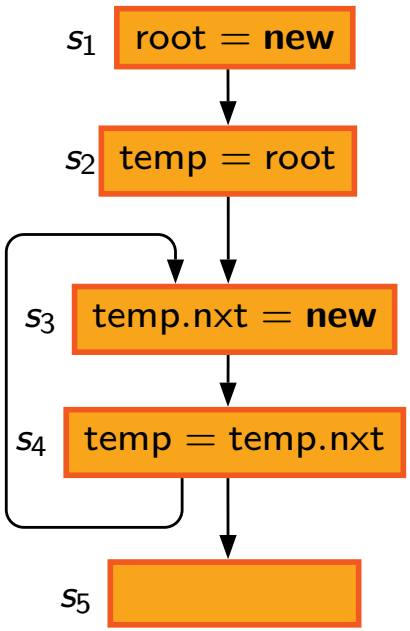


Static heap analysis
Our proposal

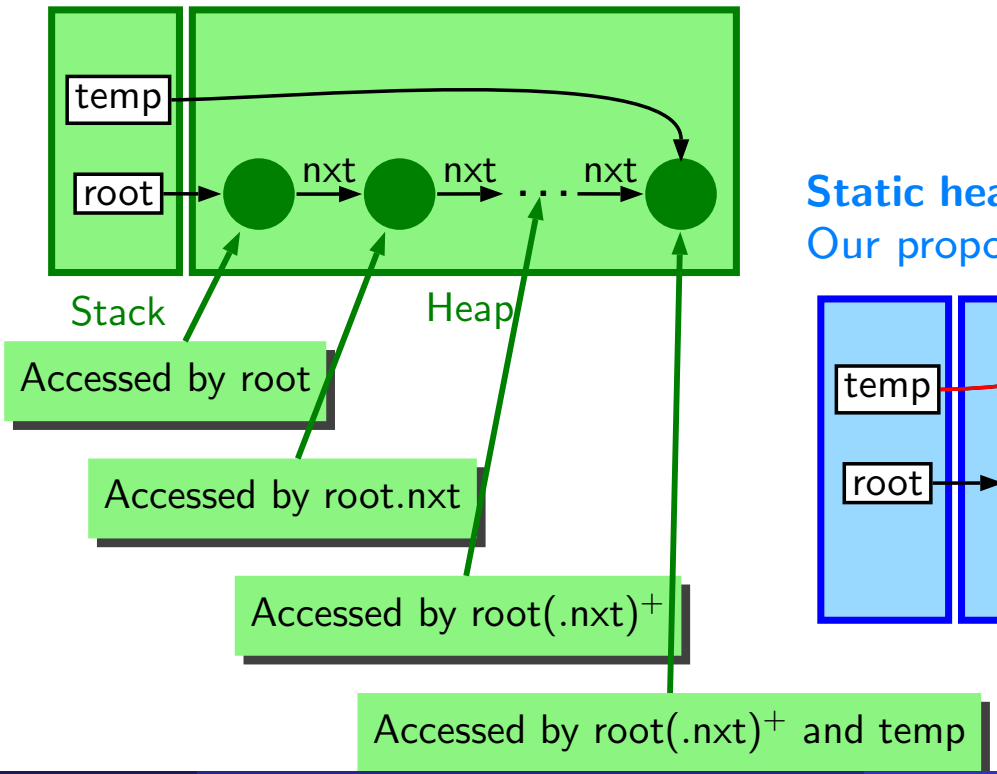


Challenges

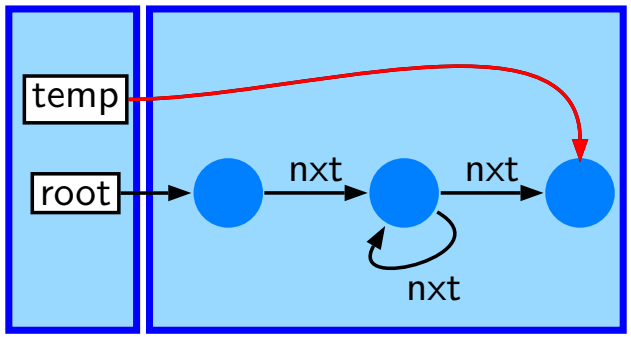
- Unbounded number of pointer expressions
  (e.g. root.nxt, root.nxt.nxt, root.nxt.nxt.nxt, etc.)
- Combinatorially large number of pointer expressions



Possible runtime memory graphs at end of program

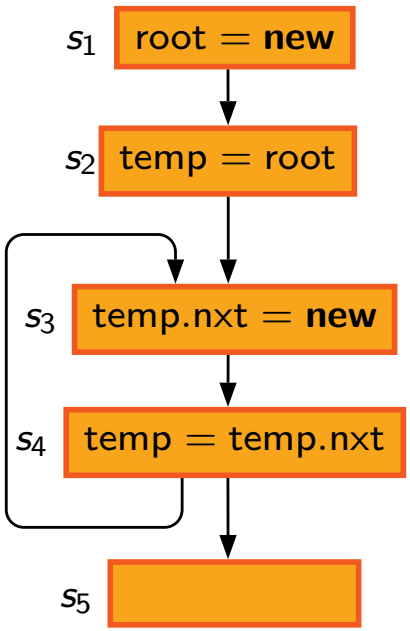


Static heap analysis
Our proposal

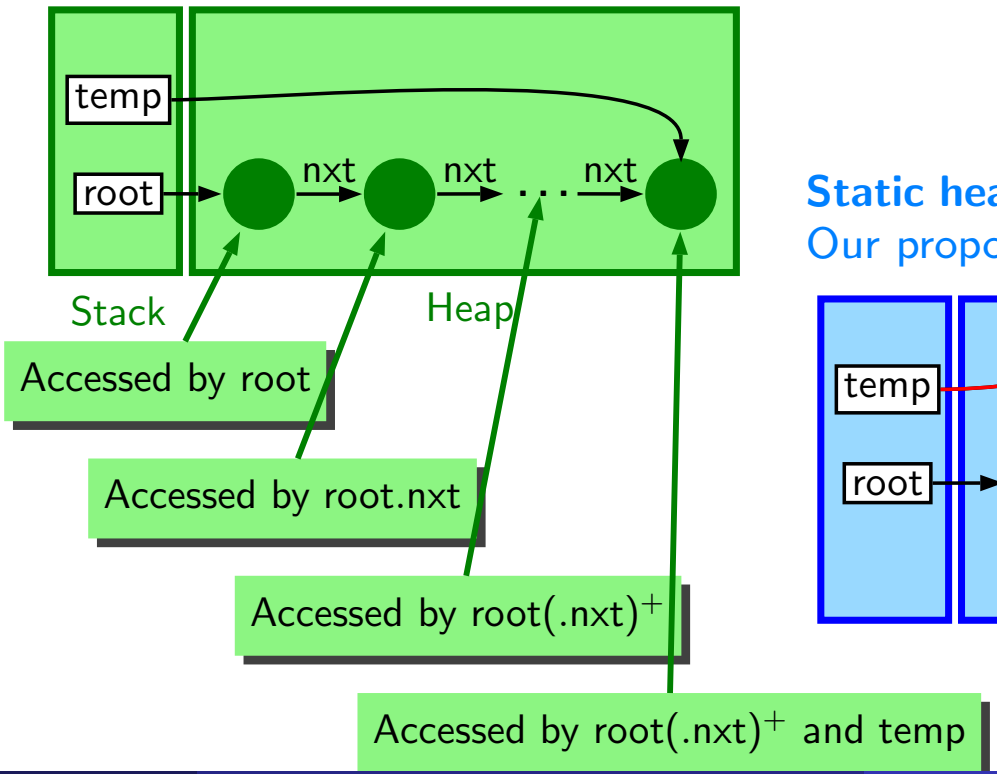


Challenges

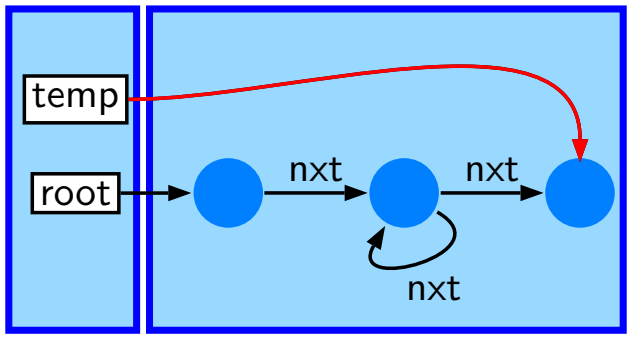
- Unbounded number of pointer expressions
  (e.g. root.nxt, root.nxt.nxt, root.nxt.nxt.nxt, etc.)
  - Distinguish betweeen expressions up to a fixed length L
- Combinatorially large number of pointer expressions



Possible runtime memory graphs at end of program

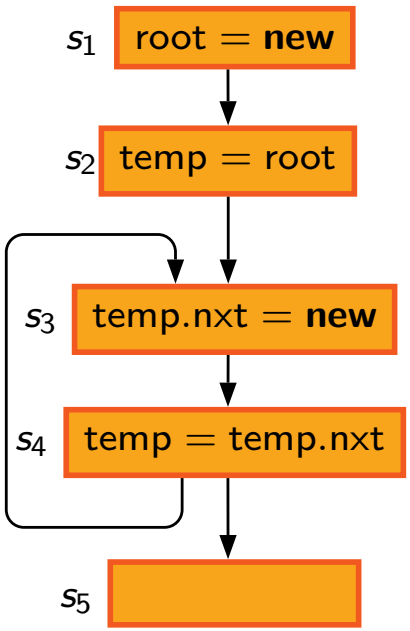


Static heap analysis
Our proposal

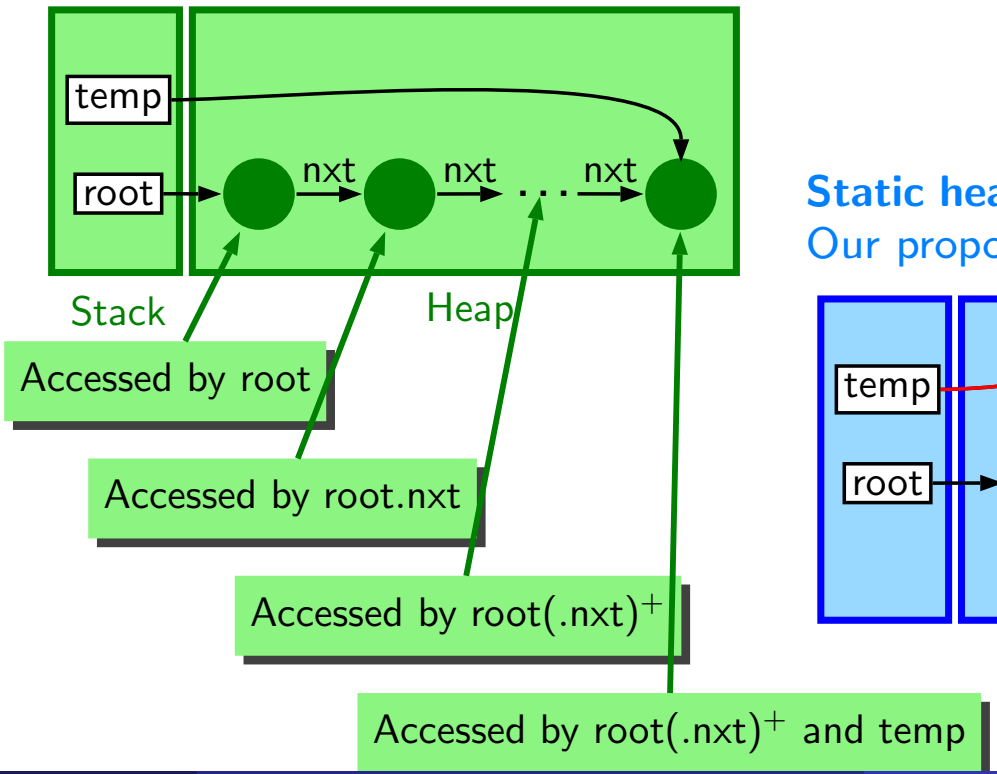


Challenges

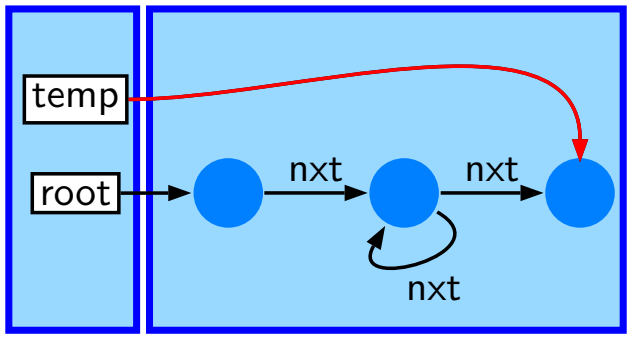
- Unbounded number of pointer expressions
  (e.g. root.nxt, root.nxt.nxt, root.nxt.nxt.nxt, etc.)
  - Distinguish betweeen expressions up to a fixed length L
- Combinatorially large number of pointer expressions
  - Memoize expressions used across program points



Possible runtime memory graphs at end of program



Static heap analysis
Our proposal

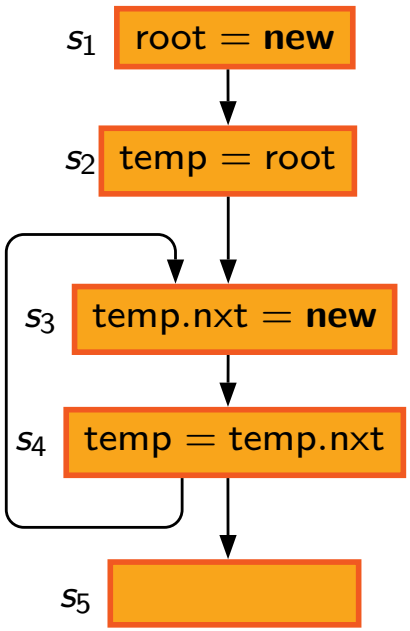


Challenges

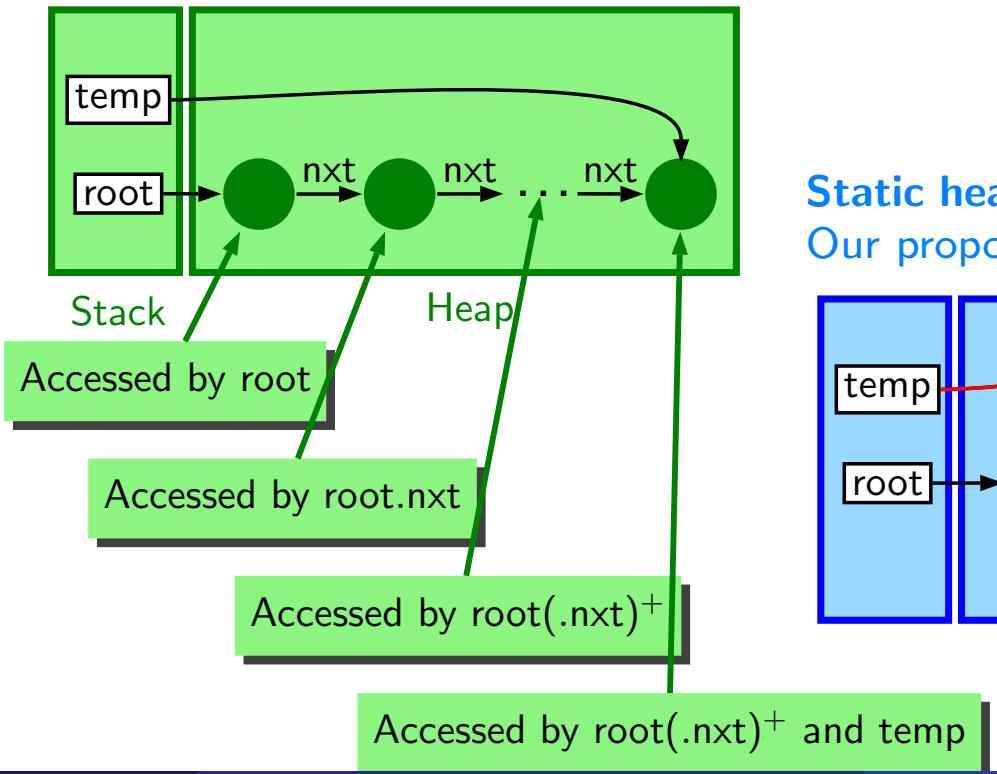
- Unbounded number of pointer expressions

  (e.g. root.nxt, root.nxt.nxt, root.nxt.nxt.nxt, etc.)

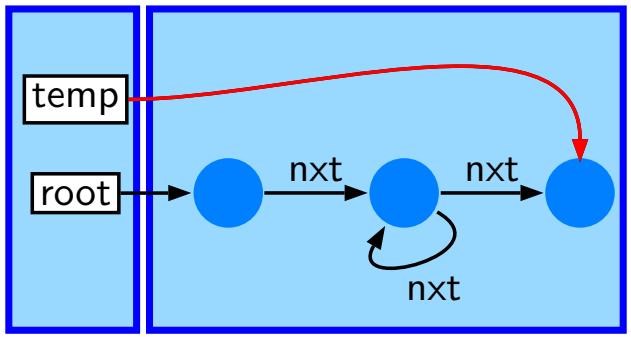
  - Distinguish betweeen expressions up to a fixed length L
- Combinatorially large number of pointer expressions
  - Memoize expressions used across program points
  - Save only live pointer expressions [Khedker et al. 2012]



Possible runtime memory graphs at end of program



Static heap analysis
Our proposal



Conclusion

- We aim to improve the precision of static heap analysis (including liveness analysis).
- We are in the process of measuring the effectiveness of our method.