Shape analysis

TVLA: a system for implementing static analyses
Tal Lev-Ami & Mooly Sagiv
Interprocedural shape analysis for cutpoint-free programs
Noam Rinetzky, Mooly Sagiv and Eran Yahav
A local shape analysis based on separation logic
Dino Distefano, Peter W. O'Hearn & Hongseok Yang

Presented by Tal Zelmanovich
4.5.2014

Shape analysis is a family of static analysis methods attempting to identify the general shape of the memory structures occurring through the program's run. These methods usually try to present the memory states of the program using a pointing graphs, or equivalent logical expressions. This advanced analysis can deal with non-trivial questions such as "is heap location x pointed by stack-pointer y?" and as "is heap location x a part of a cyclic list?", and therefore requires the graph representation to be in-depth.

One of the main issues in these algorithms is the ensuring of termination, this is commonly achieved by means of abstraction: Collapsing a sub-graph into a singular presentation or loosen properties that were concluded to apply in the graph.

Three shape-analysis methods will be presented:
TVLA: An approach utilizing three-valued logic, oriented by defining attributes as predicates
Cutpoint-free: TVLA-based method to allow interprocedural analysis
Separation logic: Representing the state as logical formula, gain speed up using locality.

The TVLA method relies on attributes (predicates) creating relations between memory locations. The user of the program can define predicates on his own as well as on which of them he is willing to compromise in the abstraction process. Three valued logic values 0, 1 or ½ are assigned to each predicate telling whether we know if it applies.

The cutpoint-free method handles only cases where function calls separates its arguments from the rest of the heap. Having this assumption allows us to cut on runtime by analyzing each procedure separately taking into account only the heap accessible from the function's arguments.

The separation logic method represent abstracted memory states as logical statements, using core properties such as: aliasing, points to, indirectly points to (ls). The power of this method lies in the locality principle: the heap is kept as a series of separated parts, and hence updates can be done locally.

At the end of the talk, I presented the approach I expected the algorithms to take prior to reading them. In this approach common subgraphs appearing throught the analysis are kept as memory templates, and can be reused to shrink the representation of the pointing graphs. This method leaves many open questions that should be handled, but will hopefully gain us the advantage of calculating attributes for each template only once. Such methods were already investigated in the past, as a possible extension to separation logic oriented analysis.

During the discussion after the presentation the following points came up:

- Can these tools be used for practical usage today?
  Although separation logic method was successfully run on the Linux kernel, the road to wide usage is still far. These methods require a lot of fine tuning (which properties we wish to analyze?, which level of abstraction to use?), and do not scale up well unless many demands are loosen.
- When should these methods be applied?
  The strong point of the algorithms is identifying heap-properties, and therefore they’re well suited for verifying data-structure handling code (for example: a code implementing a dictionary as a binary tree)
- Also during the lecture itself one of the students had the idea to bound analysis pointing-depth instead of summarization, that idea is also mentioned in the TVLA article.