Heap Typability is NP-Complete

Matt Elder
elder@cs.wisc.edu

Ben Liblit
liblit@cs.wisc.edu

Computer Sciences Department
University of Wisconsin–Madison

October 4, 2007

Abstract

Given a snapshot of a running program's memory heap, and a set of types representing data in the program, *dynamic heap type inference* attempts to assign types to memory locations such that certain global consistency constraints are satisfied. Previous work has used brute-force searches to solve heap typing problems. We prove that a problem derived from dynamic heap type inference is NP-complete by reduction from 3-colorability. Thus, it is unlikely that a polynomial-time algorithm for heap type inference exists.

1 Introduction

When debugging a program, it can be useful to examine the values and data structures in a snapshot of the program heap. This can be especially important when diagnosing memory corruption bugs in low-level, non-type-safe languages such as C. Deciphering the heap requires determining the types of allocated blocks of memory, a task which recent work has defined as the *dynamic heap type inference* problem [2, 3].

Dynamic heap type inference can be thought of as a constraint-resolution problem where the key constraints are agreement between the types of pointers and pointed-to data. For example, if one memory location holds a pointer to integer, the memory location to which it points must hold an integer or some structural subtype thereof.

Due to the heap-spanning (global) nature of these constraints, manual heap typing is tedious even for tiny programs. It quickly becomes impractical for programs of realistic size. Therefore, we would like to automate heap typing and integrate it into a debugger. WHATSAT is a working implementation of heap typing for the heaps of C programs [3]. This implementation is fundamentally a brute-force, heuristically-guided, iterative search through the solution space with backtracking when a partial solution can no longer make progress. WHATSAT works well in some experiments but performs slowly in others. In one reported case, the search neither succeeds nor terminates after many hours of execution [2].

Perhaps, then, an improved constraint resolution algorithm could solve the heap typing problem more directly, without resorting to heuristics and backtracking. Unfortunately, as this report demonstrates, a simplified problem derived from dynamic heap type inference is NP-complete. Thus, it is unlikely that any polynomial-time dynamic heap type inference exists.

2 Heap Typability

Dynamic heap type inference may involve sophisticated type systems, programmer hints, and information available at run time. We abstract away these implementation details and instead analyze the following decision subproblem, called *Heap Typability*:

A *site* is a single continuous portion of the heap that must take a single type. A site may contain several subsites if, for example, the site is a `struct`. We can properly assign a type to a site if and only if:

- the size of the site equals the size of the type;
- the values in the site are consistent with the allowed values for the type;
- the subsites of the site are typed as the type demands; and
- if the type is a pointer, then the referenced site is assigned the pointer type's referent type.

For a given heap and set of types, the Heap Typability problem asks whether all sites in the heap can be properly assigned types from the given type set.

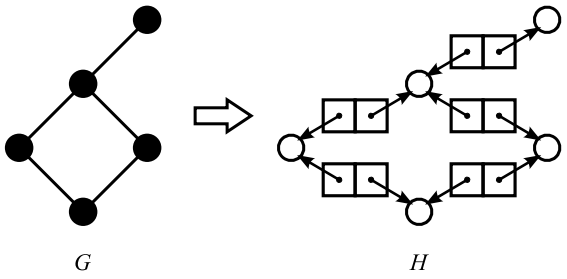

Figure 1: Reducing graph G to heap H. In H, circles represent vertex sites, and pairs of squares represent edge sites.

3 Reduction From 3-Colorability

The 3-colorability problem asks, for a given undirected graph G, whether one of three colors can be assigned to each vertex so that no two adjacent vertices have the same color. 3-colorability is a known NP-complete problem [1], so we can show that Heap Typing is NP-complete by reduction from 3-colorability.

Our reduction transforms a graph G into a heap H and the following set of types:

```
enum tiny { ZERO };

typedef struct blue { tiny t; } blue;
typedef struct gray { tiny t; } gray;
typedef struct pink { tiny t; } pink;

struct bg_edge { blue *x; gray *y; };
struct bp_edge { blue *x; pink *y; };
struct gb_edge { gray *x; blue *y; };
struct gp_edge { gray *x; pink *y; };
struct pb_edge { pink *x; blue *y; };
struct pg_edge { pink *x; gray *y; };

/* no bb_edge, gg_edge, or pp_edge. */
```

Assume that the graph G is a connected graph with at least two vertices. For every vertex v in the graph G, we place into H the *vertex site* v', which contains just the value `ZERO`. For each edge $e = (u, v)$ in G, we place into H the *edge site* e', which is composed of two pointers, one to site u' and one to site v'.

Because each vertex site is of the size of one `enum`, it can be typed only with `tiny`, `blue`, `gray`, or `pink`. Similarly, the edge sites can be typed only with the six edge struct types. Since each vertex site is referenced by at least one edge site, no vertex site may be typed by `tiny`.

Edge sites may point to node sites of any two distinct node types, but cannot point to two node sites of the same type. So, given a possible typing of all of the node sites, the edge sites can be properly typed if and only if the corresponding coloring of G has no adjacent nodes of the same color. Thus, H can be properly typed if and only if G is 3-colorable.

This reduction from G to H requires only polynomial time. Thus, if a polynomial time algorithm could tell whether H has a proper typing, then that algorithm could also determine whether G has a 3-coloring in polynomial time. 3-colorability is known to be NP-hard, so Heap Typability is NP-hard. Heap Typability is trivially in NP, since any proposed heap typing can easily be checked for validity. Therefore, Heap Typability is NP-complete.

4 Conclusions

We have shown that Heap Typability is NP-complete by reduction from 3-colorability. A polynomial-time algorithm for dynamic heap type inference would trivially provide Heap Typability decisions in polynomial time as well. Thus, if P $\neq$ NP, then no polynomial-time algorithm for dynamic heap type inference can exist.

While this does not prove that the current brute-force search strategy used by WHATSAT is the best possible al-

gorithm, it does put limits on how much better a refined approach could be. Future work in dynamic heap type inference may instead explore efficient approximation algorithms for NP-complete problems. It is an open question whether approximate heap typings can be as useful as exact typings for real debugging tasks.

References

[1] Michael R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., January 1990.

[2] Marina Polishchuk, Ben Liblit, and Chloë Schulze. WHATSAT: Dynamic heap type inference for program understanding and debugging. Technical Report 1583, University of Wisconsin–Madison, December 2006.

[3] Marina Polishchuk, Ben Liblit, and Chloë Schulze. Dynamic heap type inference for program understanding and debugging. In *Proceedings of the 34th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages*, Nice, France, January 17–19 2007. Association for Computing Machinery.