Diplomarbeitspräsentationen der Fakultät für Informatik

Shape-Based Alias Analysis for Object-Oriented Languages

Viktor Pavlu

FAKULTÄT FÜR !NFORMATIK

Masterstudium
Software Engineering / Internet Computing

Technische Universität Wien
Institut für Computersprachen
Arbeitsbereich: Programmiersprachen und Übersetzerbau
Betreuer: Dipl.-Ing. Dr. Markus Schordan

OVERVIEW

we statically approximate the heap to find aliases required for analyses and the generation of efficient code

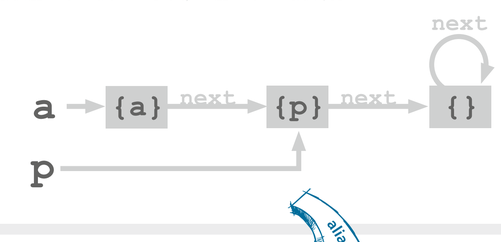
SOURCE CODE

```
#define NULL ((List*)0)

class List {
  public:
    List(int pValue) {
        next  = NULL;
        value = pValue;
    }
    List* next;
    int value;
};

int main(int argc, char **argv) {

    List *a = new List(1);
    List *t = new List(2);
    a->next = t;
    t->next = new List(3);
    t = NULL;

    List *p = a->next;

    p->next->value       = 0;   // Stmt A
    a->next->next->value = 42;  // Stmt B

    int x = p->next->value;     // Stmt C

    return 0;
}
```

at compile-time, shape analyses discover the structure of objects later allocated in the heap

shape analysis

SHAPE GRAPHS



alias analysis

interprets shapegraph to find expressions that refer to the same location

ALIAS SETS

```
(a->next) = (p)
(a->next->next) = (p->next)
(a->next->next->next) = (p->next->next)
```

ANALYSIS RESULTS
- are NULL-pointers possibly being dereferenced?
  -> *compiler issues a warning*
- are p->next->next and a->next->next->next must-aliases?
  -> *Stmt A is a definition without use and can be eliminated*
- are p->next->next and a->next->next->next no aliases?
  -> *Stmt B is independent of Stmt A and C, and could be parallelized or reordered, possibly helping with register-allocation, for example*

QUESTION

APPLICATION

- to create efficient code, a compiler needs precise information about the source program
- program analyses gather this information at compile-time
- using this information a program can be made faster, smaller, or less power-consuming

PROBLEM

- pointers are very common in object-oriented languages, but they greatly reduce what can be found out by program analyse -> *good pointer alias analyses are required*
- shape analyses are the most precise pointer analyses available
- relative quality of two shape analyses SRW* & NNH** is yet unknown

OUR WORK

- implemented parametrized versions of both shape analyses for C++
- derived 32 variations of shape-based alias analyses
- experimentally found sweet spots in runtime/precision tradeoff
- recommends two variations: for speed, for precision

COMPARISON

SRW

The shape analysis by Sagiv, Reps and Wilhelm*) was the first shape analysis to achieve strong updates for languages with destructive updating. It uses finite static shape graphs to approximate the structure of the heap.

SOURCE CODE

```
int main(int argc, char **argv) {

    List *a = new List(1);
    List *t = new List(2);
    a->next = t;
    t->next = new List(3);
    t = NULL;
```

NNH

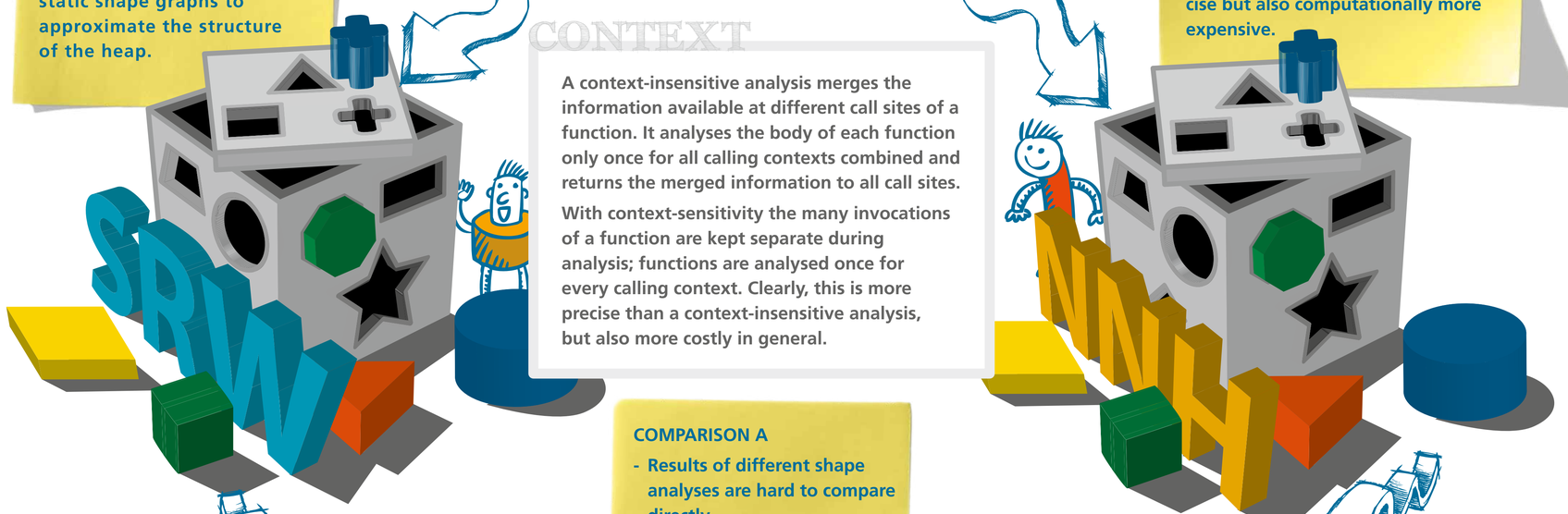
The shape analysis described by Nielson, Nielson, and Hankin in „Principles of Program Analysis“**) is based on the SRW analysis but uses sets of compatible shape graphs instead of a single graph.

This makes the analysis more precise but also computationally more expensive.

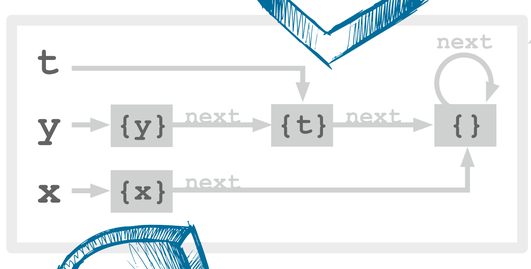
CONTEXT

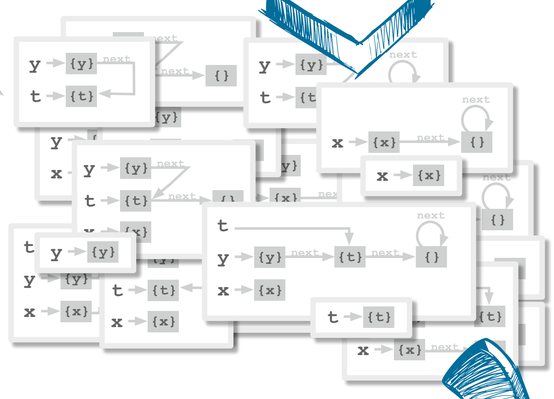
A context-insensitive analysis merges the information available at different call sites of a function. It analyses the body of each function only once for all calling contexts combined and returns the merged information to all call sites.

With context-sensitivity the many invocations of a function are kept separate during analysis; functions are analysed once for every calling context. Clearly, this is more precise than a context-insensitive analysis, but also more costly in general.



COMPARISON A
- Results of different shape analyses are hard to compare directly
- When one representation is converted to the other, graphs are often equal, though the precision of the analyses may not be equal





- don't convert indeterministic SRW static shape graph into NNH set of compatible shape graphs, but
- perform alias test directly on SRW static shape graph
- only one test instead of one per graph in the set
- for analysis results already in SRW form, there is no loss of precision
- for NNH shape graph sets, the precision is lost during conversion

lazy alias test

COMMON TAILS

Interpreting shape graphs

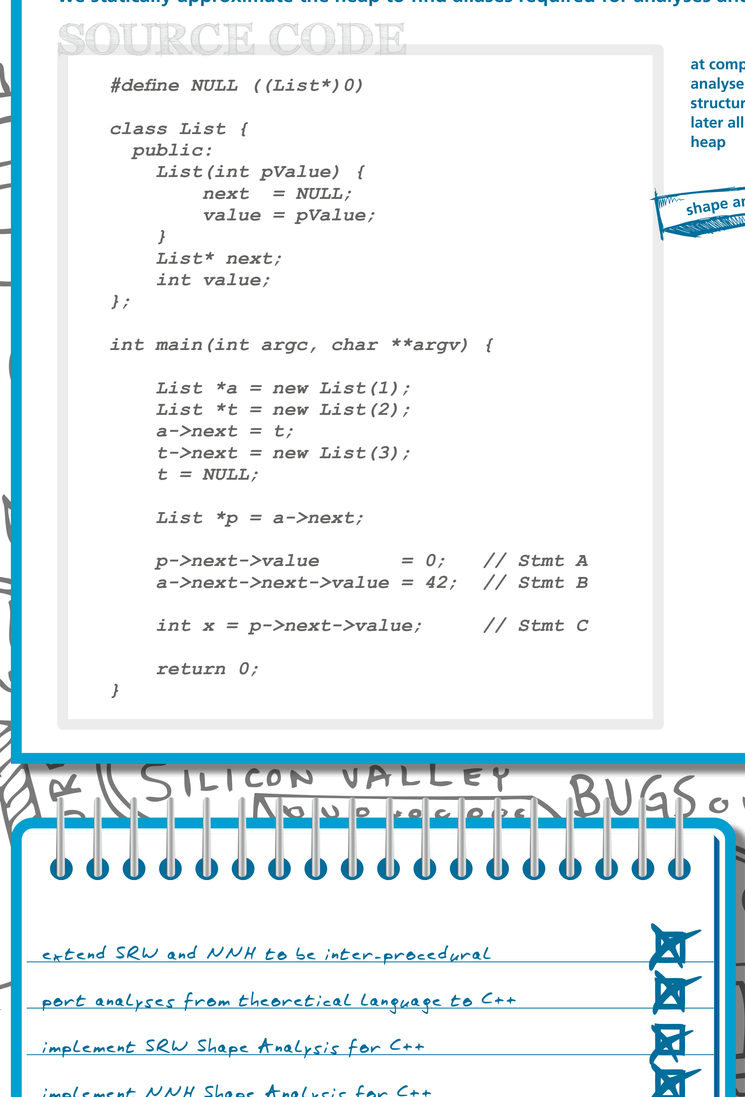
naive (as described in RWS02***)):
- expressions leading to the same named node are must-aliases
- expressions leading to the summary location are may-aliases

our common tails test:
- when expressions leading to the summary location share a common tail of selectors that starts at a common named node, they too are must-aliases
- all other expressions leading to the unshared summary location are not aliased

- tests aliasing of expressions in *every* single shape graph
- only if they are must-aliases in every member shape graph are they actually must-aliases
- conversely, only if there is no alias in any member shape graph are the expressions not aliased
- in all other cases: may-alias

extensive alias test

COMPARISON B
- Interpreting Shape Graphs to obtain alias sets makes different shape analyses comparable by the size of derived may-alias sets
- Smaller may-alias sets are the better results and indicate a more precise underlying shape analysis



SUMMARY

The impact of five orthogonal analysis parameters has been studied:
- shape analysis algorithm: SRW vs. NNH
- context-sensitivity: with vs. without
- retaining temporary variables for additional names vs. removing them
- alias test: lazy vs. extensive
- alias test: compare final node vs. common tail of selectors

-> *comparison of 32 variations of shape-based alias analyses*

SRW

context	extensive	lazy
naive		T F
common tails		T F

NNH

context	extensive	lazy
naive	T F	T F
common tails	T F	T F

context	extensive	lazy
naive		
common tails		

context	extensive	lazy
naive		
common tails		

Results
- ignoring context information produced the worst results *and* took the most time – always use context information!
- SRW shape analysis + extensive test cannot be more precise than the lazy test on SRW graphs – always perform the lazy test to save time!
- precise NNH + fast lazy alias test is bad: always slower than SRW but only in one case more precise
- without tempvar or common tail extension, SRW and NNH have comparable precision, but SRW is roughly 5x faster
- retaining temporary variables increased precision for both SRW and NNH (23% and 21% smaller alias sets) but also increased graph sizes and therefore anlysis runtime (3x) – expensive precision
- common tails test increased precision only in combination with NNH, but then at no measurable increased cost – cheap precision for expensive analysis

-> fastest: SRW + context + lazy - common tails - tempvars
-> most precise: NNH + context + extensive + common tails (time x36, precision +28%)

*) Mooly Sagiv, Thomas W. Reps, and Reinhard Wilhelm. Solving shape-analysis problems in languages with destructive updating. ACM Transactions on Programming Languages and Systems (TOPLAS), 20(1):1–50, January 1998.

**) Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. Principles of Program Analysis, chapter Shape Analysis, pages 102–129. Springer, 1999.

***) Thomas W. Reps, Mooly Sagiv, and Reinhard Wilhelm. Shape analysis and applications. In The Compiler Design Handbook: Optimizations and Machine Code Generation, pages 175–218. CRC Press, 2002.

Kontakt: e0425543@student.tuwien.ac.at