Query-Directed Adaptive Heap Cloning for Optimizing Compilers

Yulei Sui, Yue Li, Jingling Xue

Programming Languages and Compilers Group
School of Computer Science and Engineering
University of New South Wales, Australia

February 25, 2013

Heap cloning

Statically distinguishing heap objects by call paths

Heap cloning

Statically distinguishing heap objects by call paths

```
int main(){
 int* buffer1 = getMem();
 int* buffer2 = getMem();
}
```

```
int* getMem(){
 return malloc(10);
}
```

Heap cloning

Statically distinguishing heap objects by call paths

```
int main(){
  int* buffer1 = getMem();
  int* buffer2 = getMem();
}
```

```
int* getMem(){
  return malloc(10);
}
```

main

buffer1=getMem();

Program execution

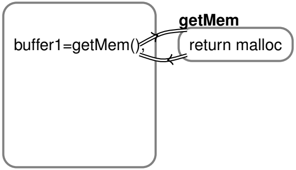
Heap cloning

Statically distinguishing heap objects by call paths

```
int main(){
  int* buffer1 = getMem();
  int* buffer2 = getMem();
}
```

```
int* getMem(){
  return malloc(10);
}
```

main



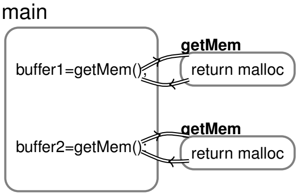
Program execution

Heap cloning

Statically distinguishing heap objects by call paths

```
int main(){
  int* buffer1 = getMem();
  int* buffer2 = getMem();
}
```

```
int* getMem(){
  return malloc(10);
}
```



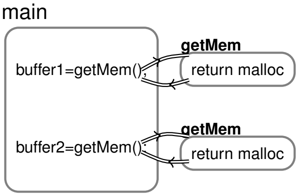
Program execution

Heap cloning

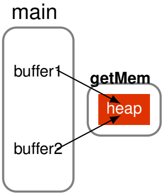
Statically distinguishing heap objects by call paths

```
int main(){
  int* buffer1 = getMem();
  int* buffer2 = getMem();
}
```

```
int* getMem(){
  return malloc(10);
}
```



Program execution



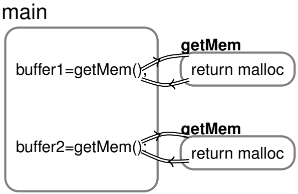
Points-to relations without heap cloning

@ Yulei Sui, CGO 2013, 25/02/2013
PLC, UNSW

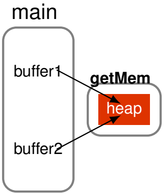
Heap cloning

Statically distinguishing heap objects by call paths

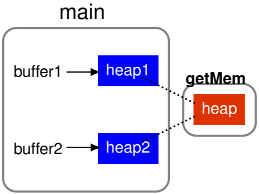
```
int main(){
  int* buffer1 = getMem();
  int* buffer2 = getMem();
}

int* getMem(){
  return malloc(10);
}
```



Program execution



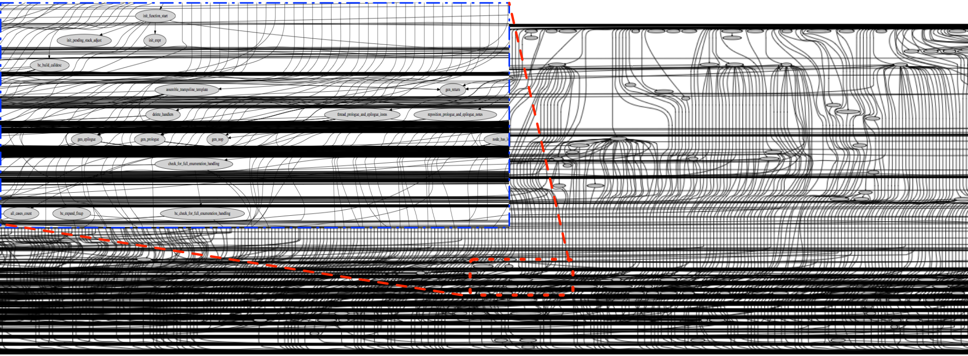
Points-to relations without heap cloning



Points-to relations with heap cloning

@ Yulei Sui, CGO 2013, 25/02/2013
PLC, UNSW

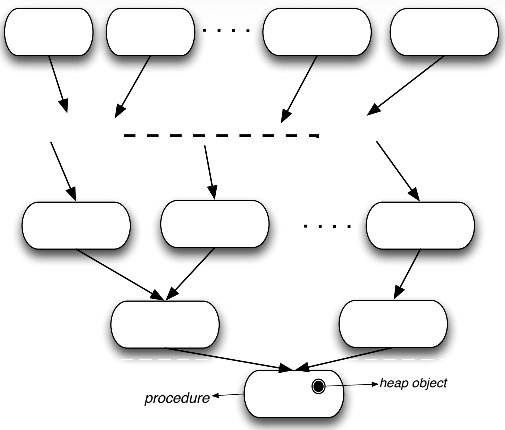
Call graph of 176.gcc (230.4KLOC)



#Procedures: 2256 #Pointers: 134380 #Calling Contexts: $1.2*10^5$

Context-sensitive heap cloning can be costly!

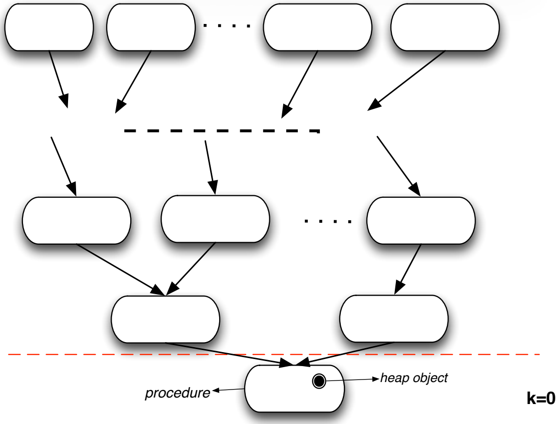
K-callsite-sensitive heap cloning



Call Graph with K-callsite-sensitive heap cloning

[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]

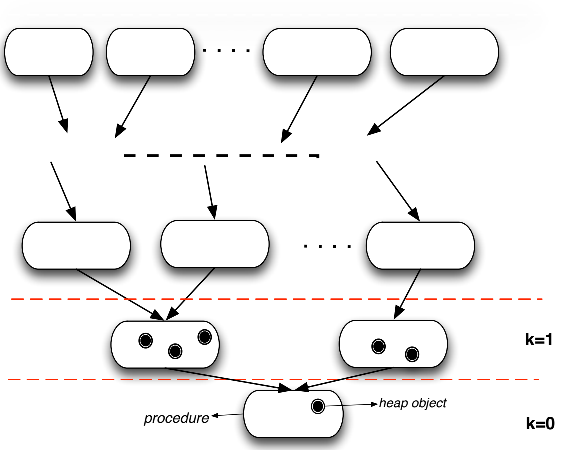
K-callsite-sensitive heap cloning



Call Graph with K-callsite-sensitive heap cloning

[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]

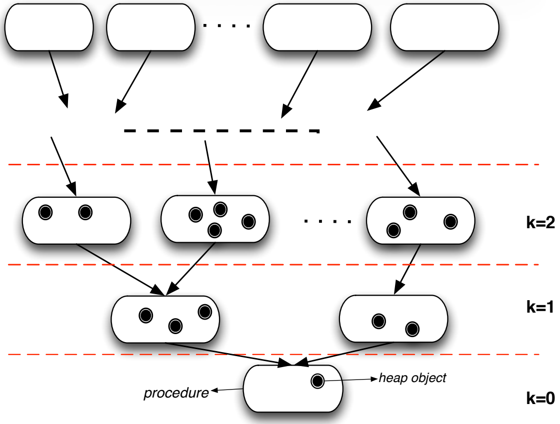
K-callsite-sensitive heap cloning



Call Graph with K-callsite-sensitive heap cloning

[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]

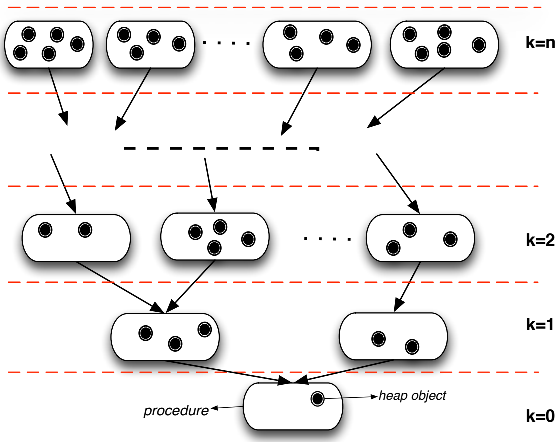
K-callsite-sensitive heap cloning



Call Graph with K-callsite-sensitive heap cloning

[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]

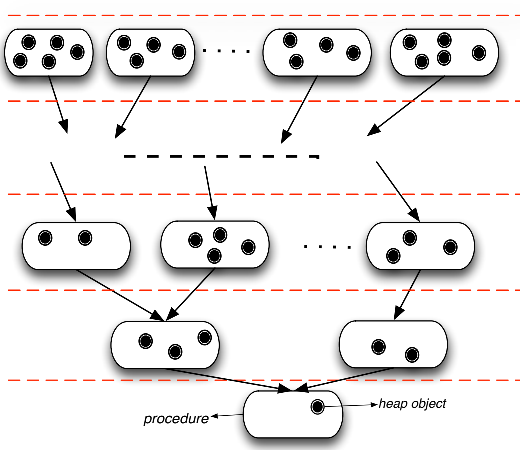
K-callsite-sensitive heap cloning



Call Graph with K-callsite-sensitive heap cloning

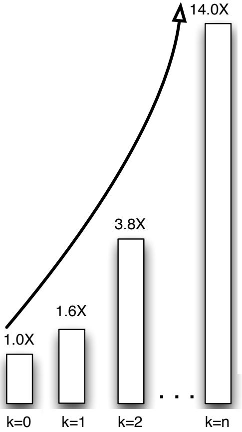
[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]

K-callsite-sensitive heap cloning



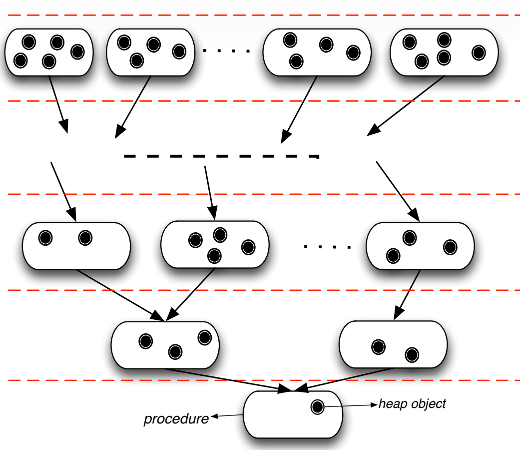
Call Graph with K-callsite-sensitive heap cloning

[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]



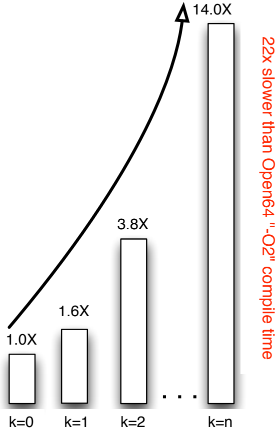
Analysis time of 176.gcc with Andersen-style context sensitive heap cloning

K-callsite-sensitive heap cloning



Call Graph with K-callsite-sensitive heap cloning

[Nystrom-SAS'04, Nystrom-PASTE'04, Lhotak-CC'06, Xu-ISSTA'08]



Analysis time of 176.gcc with Andersen-style context sensitive heap cloning

- Is *full heap cloning* overkill (relative to a client's needs)?

- Is *full heap cloning* overkill (relative to a client's needs)?

- Is *k-callsite sensitive cloning* the best solution?

Alias Query

- Whether two expressions may represent the same memory location.

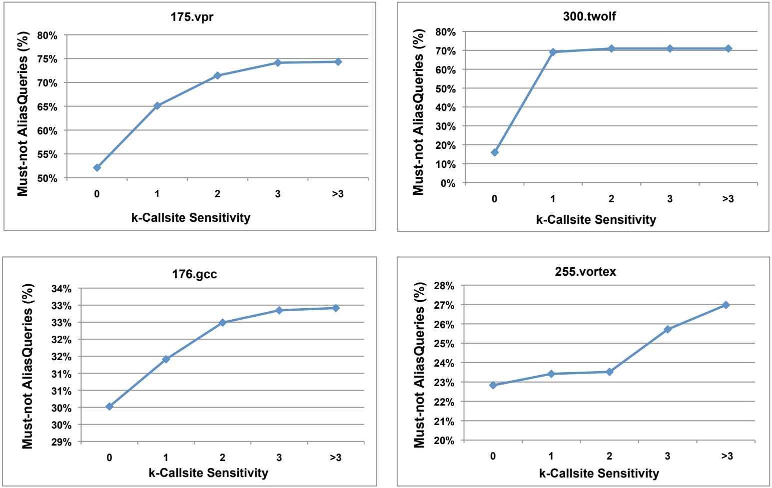
Alias Query

- Whether two expressions may represent the same memory location.
- For example: ⟨*buffer1,*buffer2⟩
    - Alias   Without Heap Cloning

Alias Query

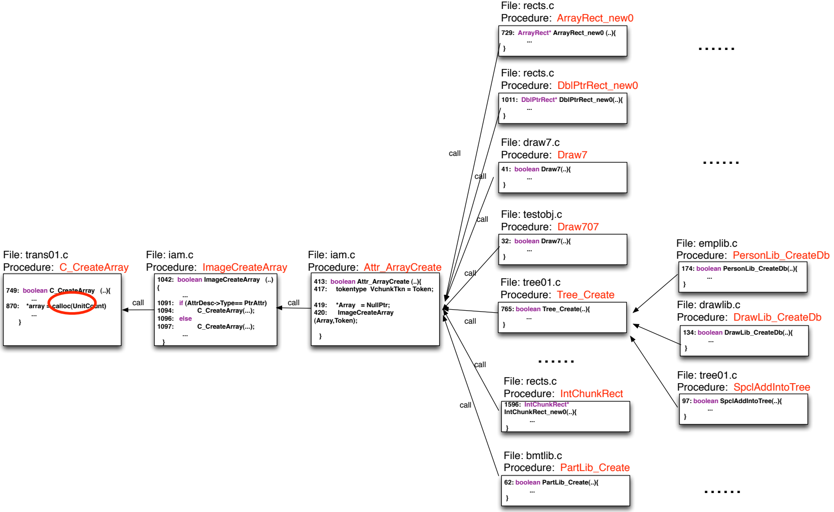
- Whether two expressions may represent the same memory location.
- For example: ⟨*buffer1,*buffer2⟩
  - Alias Without Heap Cloning
  - Not-Alias Heap Cloning

Analysis precision for answering alias queries



Percentage of must-not aliases disambiguated among the queries issued by WOPT with k-callsite-sensitive heap cloning

A close look at 255.vortex's call graph

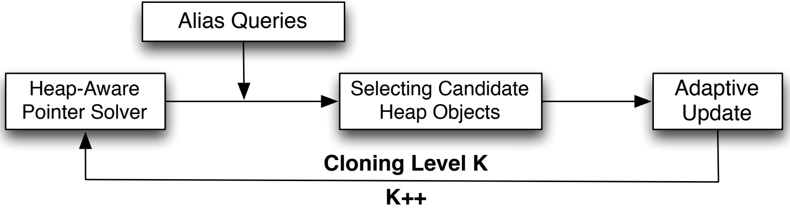


Goal

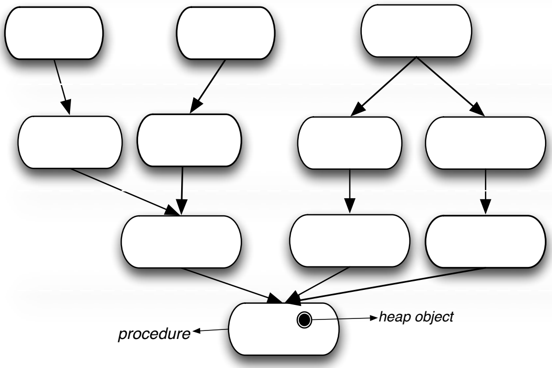
- Can we enable heap cloning only where it is necessary?
- Can we achieve the same precision as full heap cloning according to a client's needs?

Our QUDA framework

QUery-Directed Adaptive heap cloning

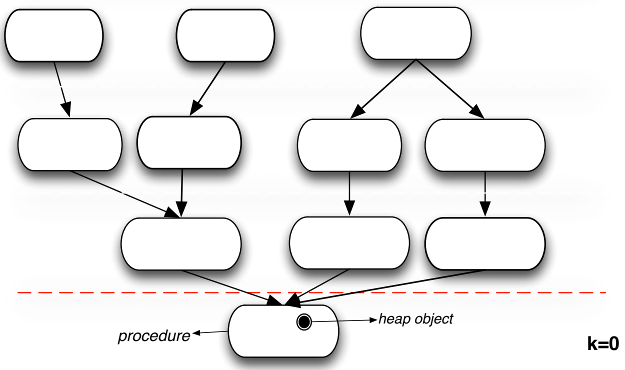


Query-directed adaptive heap cloning



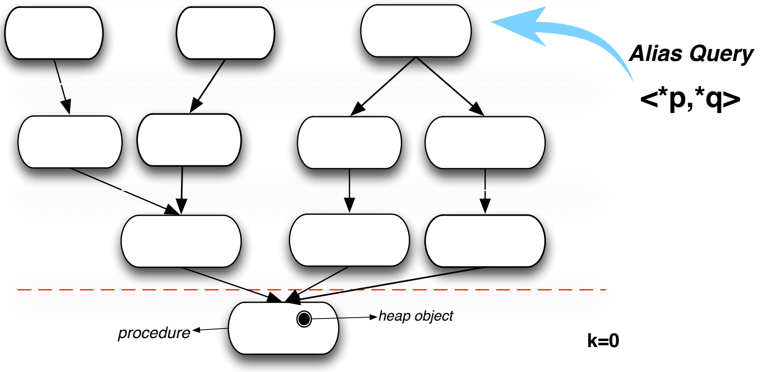
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



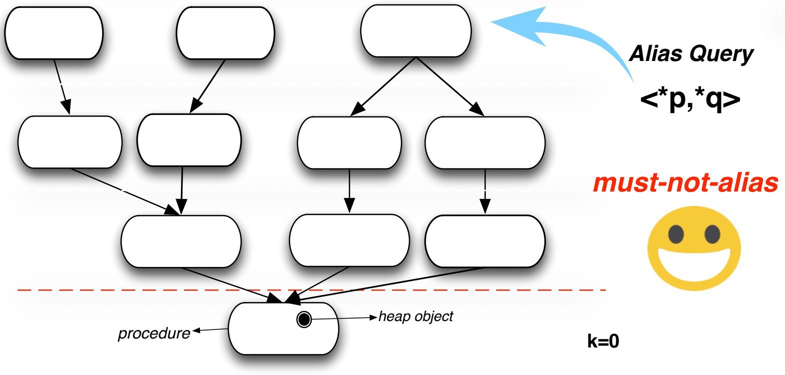
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



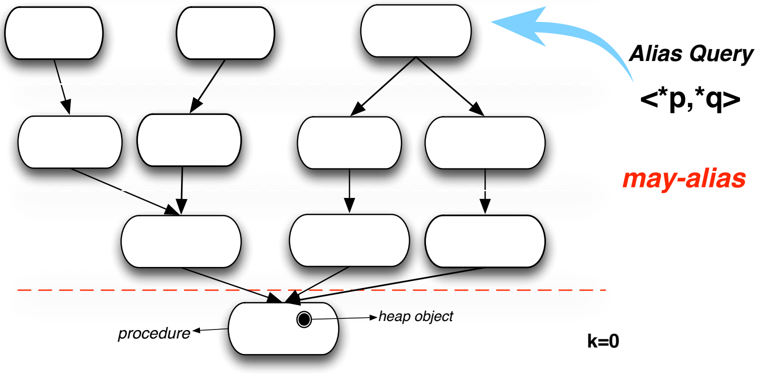
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



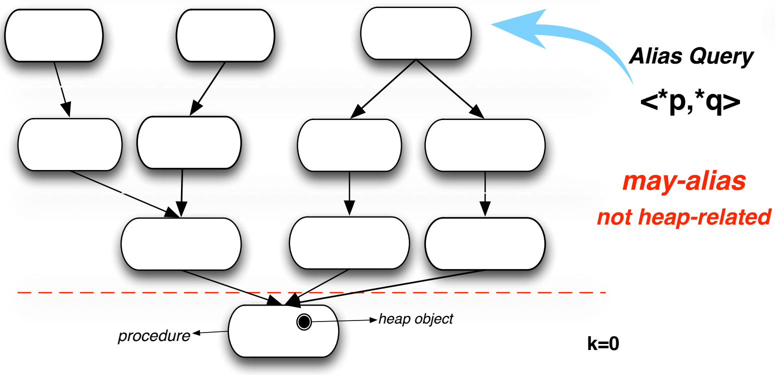
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



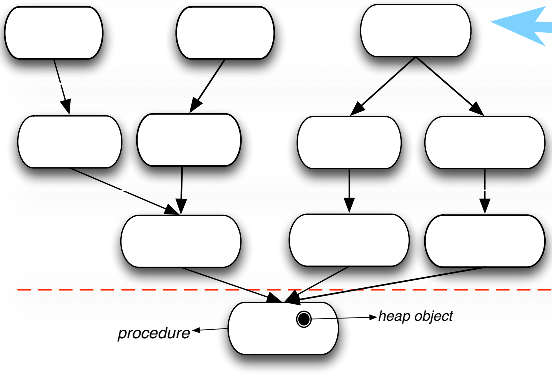
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



Query-Directed Adaptive Heap Cloning

Alias Query

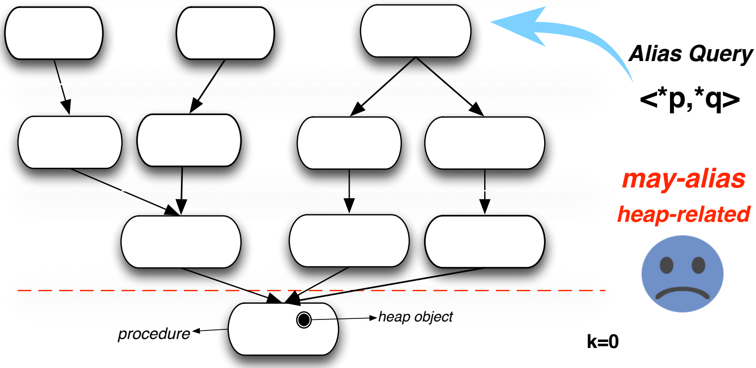
<*p,*q>

may-alias

not heap-related

k=0

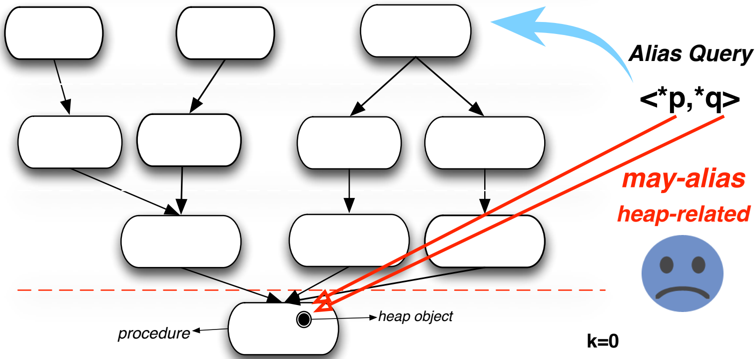
Query-directed adaptive heap cloning



Query-Directed Adaptive Heap Cloning

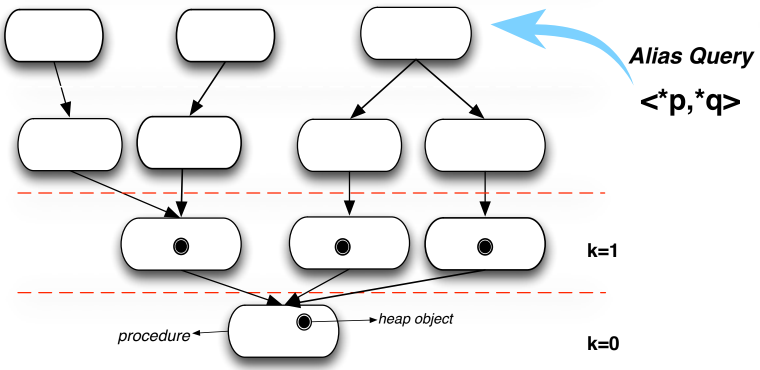
@ Yulei Sui, CGO 2013, 25/02/2013
PLC, UNSW

Query-directed adaptive heap cloning



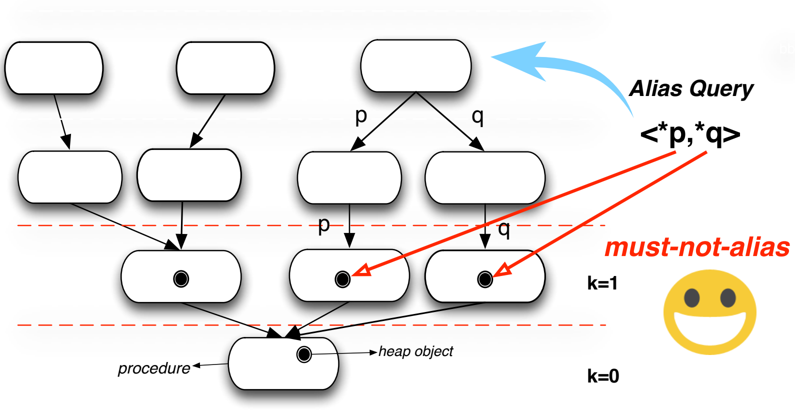
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



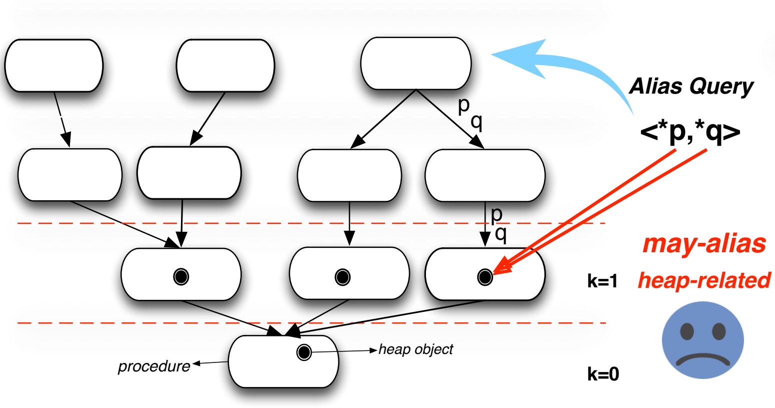
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



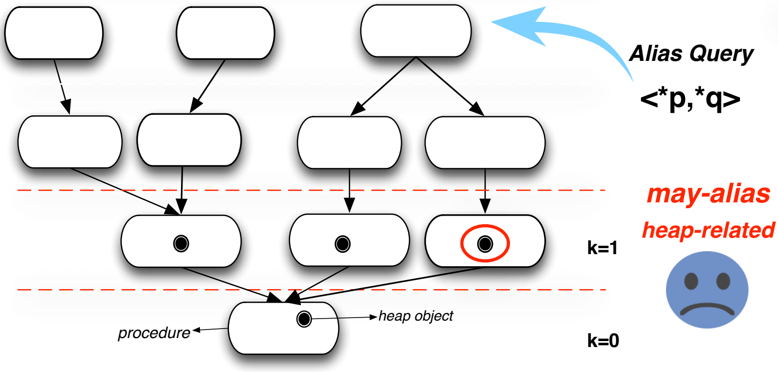
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



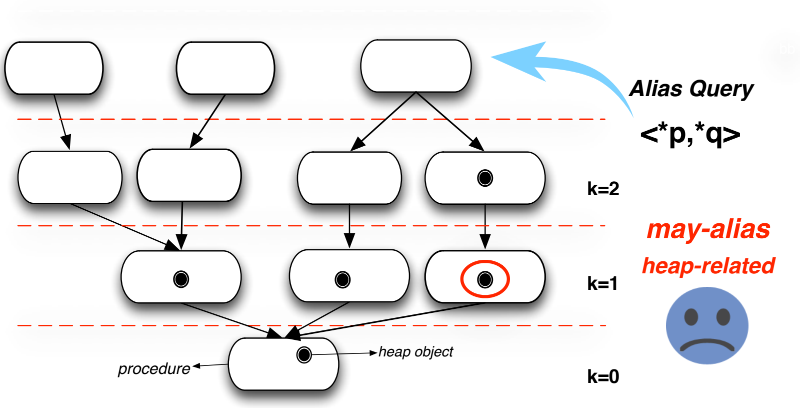
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



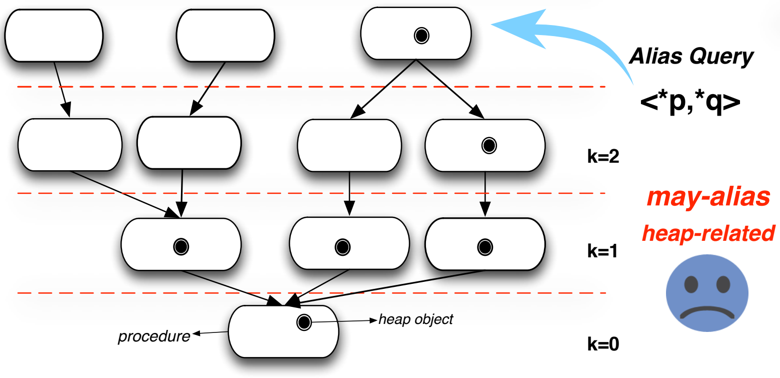
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



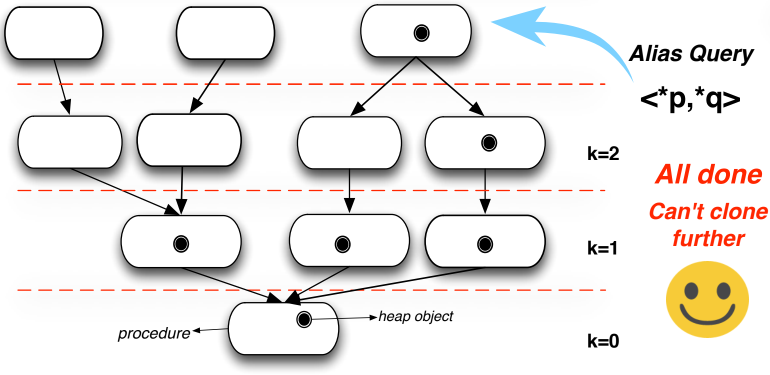
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



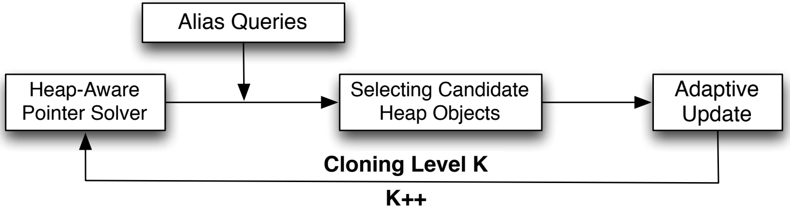
Query-Directed Adaptive Heap Cloning

Query-directed adaptive heap cloning



Query-Directed Adaptive Heap Cloning

QUDA:QUery-Directed Adaptive heap cloning



Heap-aware pointer analysis

x $\rightarrow g$
p $\rightarrow o$

Heap-aware pointer analysis

$$\begin{array}{lll} \mathrm{x} & \rightarrow (\mathrm{true}, & g) \\ \mathrm{p} & \rightarrow (h_o, & o) \end{array}$$

Heap-aware pointer analysis

$$
\begin{aligned}
x &\rightarrow (\text{true},\ g) \\
p &\rightarrow (h_o,\ \ o) \\
q &\rightarrow (h_o,\ \ o)
\end{aligned}
$$

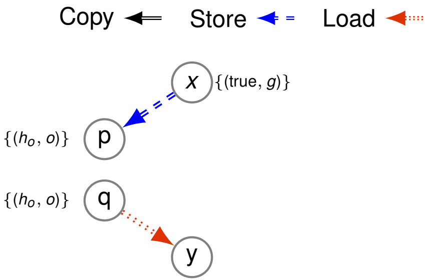
```
*p  =  x;
y   =  *q;
```

Heap-aware pointer analysis

$$
\begin{aligned}
x &\rightarrow (\text{true},\ g)\\
p &\rightarrow (h_o,\ o)\\
q &\rightarrow (h_o,\ o)
\end{aligned}
$$

```
*p = x;
y  = *q;
```

Constraint Graph:



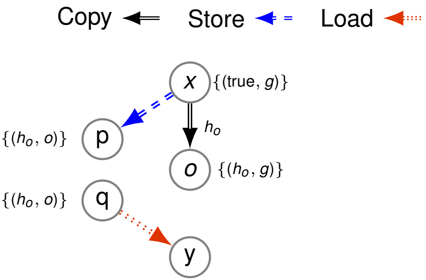
@ Yulei Sui, CGO 2013, 25/02/2013
PLC, UNSW

Heap-aware pointer analysis

$x \rightarrow (\text{true},\ g)$
$p \rightarrow (h_o,\ o)$
$q \rightarrow (h_o,\ o)$

*p = x;
y = *q;

Constraint Graph:

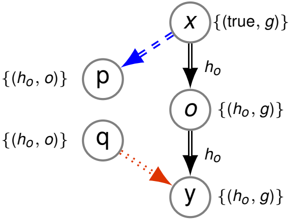


Heap-aware pointer analysis

Constraint Graph:

Copy $\Longleftarrow$ Store Load

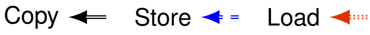
$$
\begin{aligned}
\mathrm{x} &\rightarrow (\mathrm{true},\ g)\\
\mathrm{p} &\rightarrow (h_o,\ \ o)\\
\mathrm{q} &\rightarrow (h_o,\ \ o)
\end{aligned}
$$




```
*p  =  x;
y   =  *q;
```

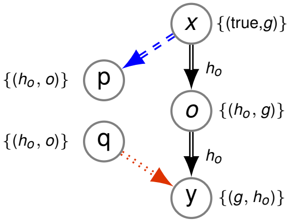
Candidate heap objects selection

Constraint Graph:



Alias Query

$\langle *x, *y \rangle$



@ Yulei Sui, CGO 2013, 25/02/2013
PLC, UNSW

Candidate heap objects selection

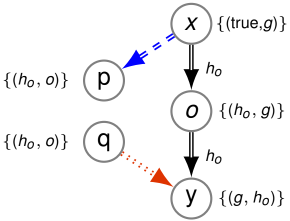
Constraint Graph:

Copy Store Load

Alias Query

$\langle *x, *y \rangle$

pts(x)={true,g}
pts(y)={$\boxed{h_o}$,g}



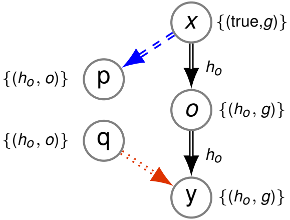
Candidate Heap Object $\{o\}$

Adaptive update

Constraint Graph:

Copy Store Load

Candidate Heap Object $\{o\}$

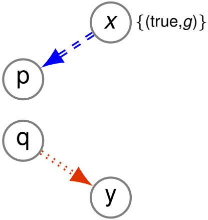


Adaptive update

Constraint Graph:

Copy ⟸ Store ⇠ Load ⇠

Candidate Heap Object $\{o\}$



Next round resolution

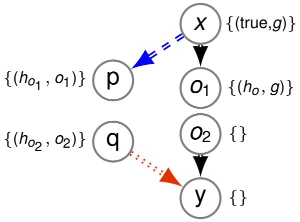
Constraint Graph:

Copy Store Load

Alias Query

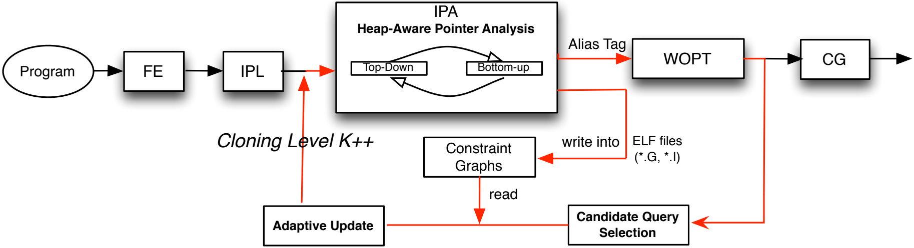
$\langle *x, *y \rangle$

Not-alias!

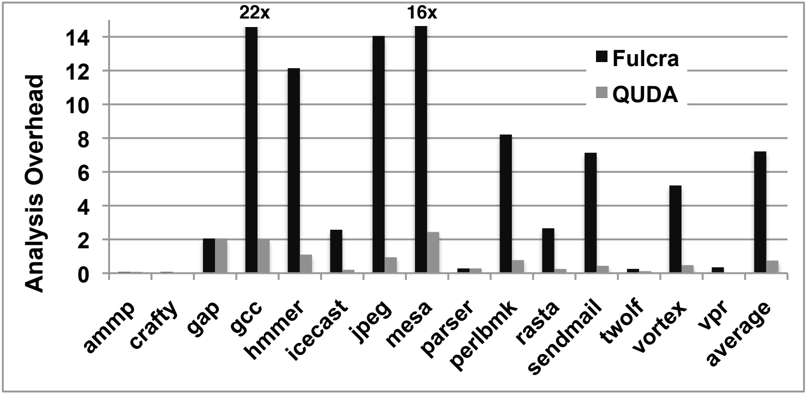


@ Yulei Sui, CGO 2013, 25/02/2013
PLC, UNSW

QUDA framework in Open64

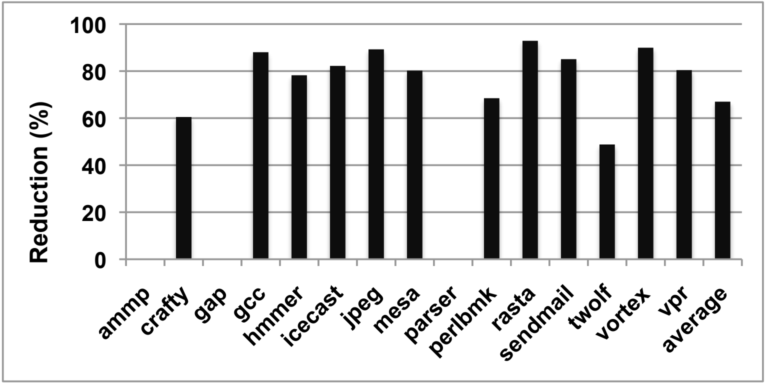


Analysis times of FULCRA and QUDA



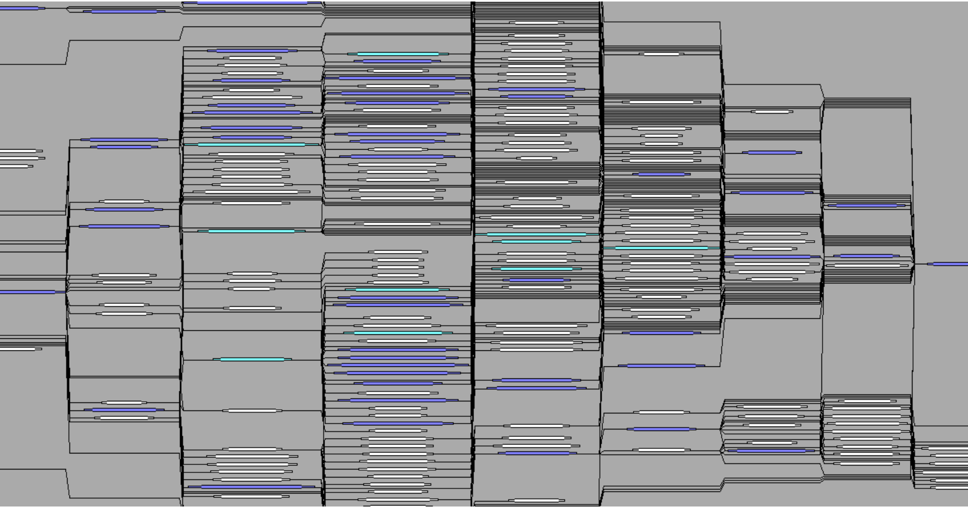
Analysis time normalized with respect to Open64's compile times (-O2)

Heap objects reduced by QUDA over FULCRA

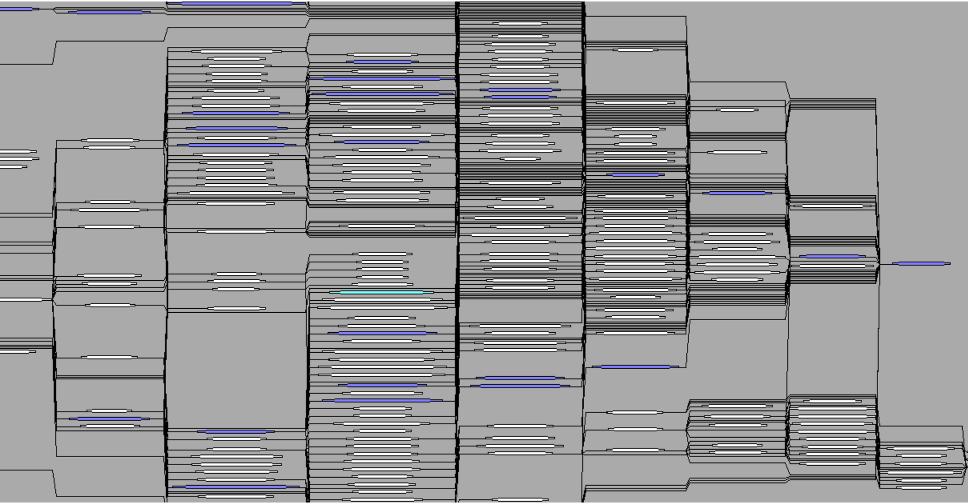


Number of heap objects reduced by QUDA over FULCRA in percentage terms

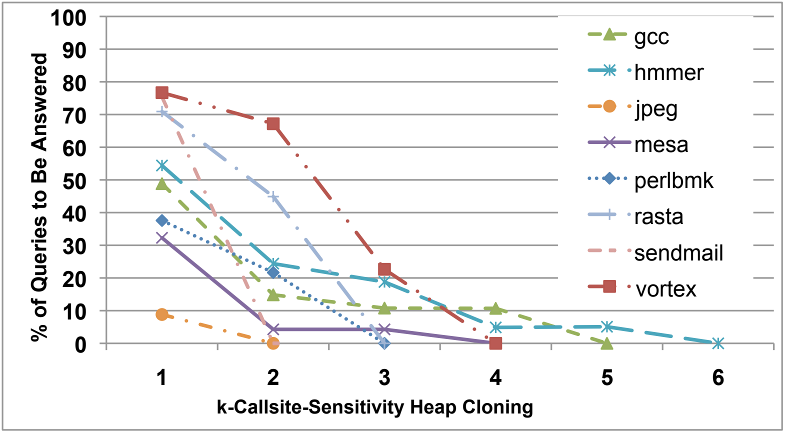
Heap distribution with full heap cloning (175.vpr)



Heap distribution with QUDA (175.vpr)

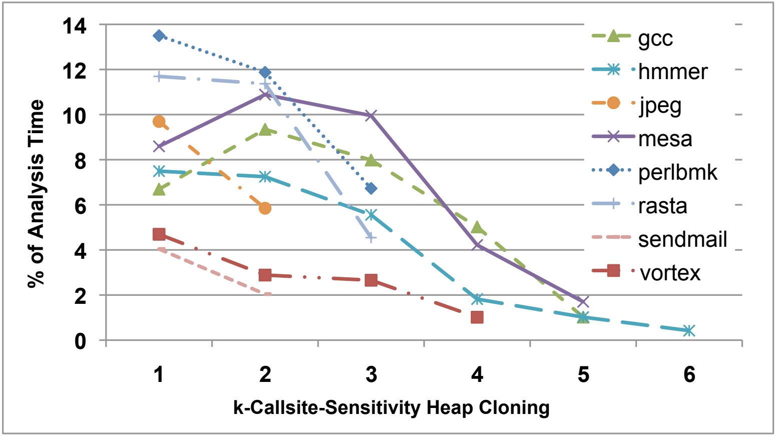


Alias queries to be answered at each iteration



UNSW THE UNIVERSITY OF NEW SOUTH WALES

Analysis time per iteration over the total



Conclusion

Novel heap cloning approach: same precision as full heap cloning but significantly more scalable

- Heap-aware analysis
- Query-directed
- Adaptive

Challenges and opportunities:

- Iterative compilation (prioritising queries in hot functions)
- Bug detection (scaling precise pointer analysis)