Script **generated by TTT**

Title: Seidl: Programmoptimierung (27.11.2013)

Date: Wed Nov 27 08:30:26 CET 2013

Duration: 74:26 min

Pages: 21

Simplification:

- We consider pointers to the beginning of blocks A which allow indexed accesses $A[i]$:-)
- We ignore well-typedness of the blocks.
- New statements:

$x = \text{new}();$ // allocation of a new block

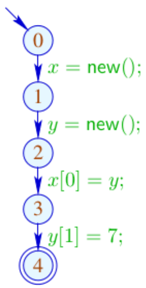
$x = y[e];$ // indexed read access to a block

$y[e_1] = e_2;$ // indexed write access to a block

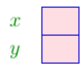
- Blocks are possibly infinite :-)
- For simplicity, all pointers point to the beginning of a block.

Simple Example:

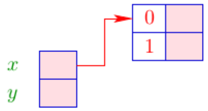
$x = \text{new}();$
$y = \text{new}();$
$x[0] = y;$
$y[1] = 7;$



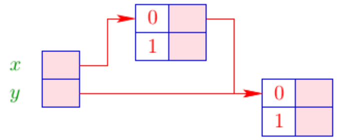
The Semantics:



The Semantics:

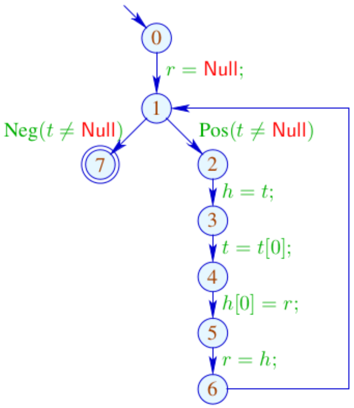


The Semantics:



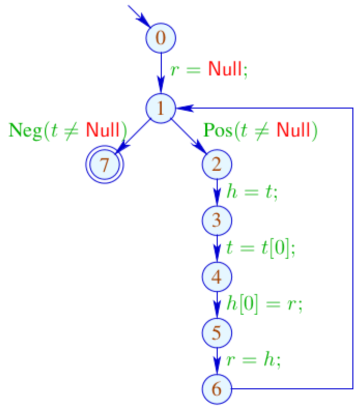
More Complex Example:

$r = $ Null;
while $(t \neq$ Null$)$ {
$\quad h = t;$
$\quad t = t[0];$
$\quad h[0] = r;$
$\quad r = h;$
}



More Complex Example:

$r = $ Null;
while $(t \neq$ Null$)$ {
$\quad h = t;$
$\quad t = t[0];$
$\quad h[0] = r;$
$\quad r = h;$
}



Concrete Semantics:

A store consists of a finite collection of blocks.

After h new-operations we obtain:

$$\begin{array}{lcll} Addr_h & = & \{\text{ref } a \mid 0 \leq a < h\} & // \text{ addresses} \\ Val_h & = & Addr_h \cup \mathbb{Z} & // \text{ values} \\ Store_h & = & (Addr_h \times \mathbb{N}_0) \to Val_h & // \text{ store} \\ State_h & = & (Vars \to Val_h) \times Store_h & // \text{ states} \end{array}$$

For simplicity, we set: $0 = \text{Null}$

Let $(\rho, \mu) \in State_h$. Then we obtain for the new edges:

$$\begin{array}{lcl} [\![x = \text{new}();]\!]\, (\rho, \mu) & = & (\rho \oplus \{x \mapsto \text{ref } h\}, \\ & & \mu \oplus \{(\text{ref } h, i) \mapsto 0 \mid i \in \mathbb{N}_0\}) \\ [\![x = y[e];]\!]\, (\rho, \mu) & = & (\rho \oplus \{x \mapsto \mu\, (\rho\, y, [\![e]\!]\, \rho)\}, \mu) \\ [\![y[e_1] = e_2;]\!]\, (\rho, \mu) & = & (\rho, \mu \oplus \{(\rho\, y, [\![e_1]\!]\, \rho) \mapsto [\![e_2]\!]\, \rho\}) \end{array}$$

Caveat:

This semantics is too detailled in that it computes with absolute Addresses. Accordingly, the two programs:

$x = \text{new}();$
$y = \text{new}();$

$y = \text{new}();$
$x = \text{new}();$

are not considered as equivalent !!?

Possible Solution:

Define equivalence only up to permutation of addresses :-)

Alias Analysis 1. Idea:

- Distinguish finitely many classes of blocks.
- Collect all addresses of a block into one set!
- Use sets of addresses as abstract values!

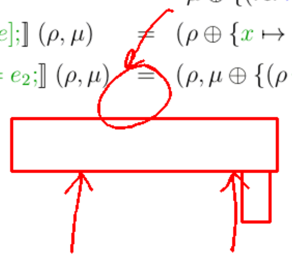
  $\Longrightarrow$ Points-to-Analysis

$$\begin{array}{lcll} Addr^\sharp & = & \boxed{Edges} & // \text{ creation edges} \\ Val^\sharp & = & 2^{Addr^\sharp} & // \text{ abstract values} \\ Store^\sharp & = & Addr^\sharp \to Val^\sharp & // \text{ abstract store} \\ State^\sharp & = & (Vars \to Val^\sharp) \times Store^\sharp & // \text{ abstract states} \end{array}$$

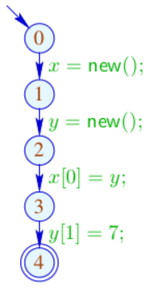
// complete lattice !!!

Let $(\rho, \mu) \in State_h$. Then we obtain for the new edges:

$$
\begin{array}{lll}
[\![x = \mathsf{new}();]\!]\,(\rho,\mu) & = & (\rho \oplus \{x \mapsto \mathsf{ref}\; h\}, \\
 & & \quad \mu \oplus \{(\mathsf{ref}\; h, i) \mapsto 0 \mid i \in \mathbb{N}_0\}) \\
[\![x = y[e];]\!]\,(\rho,\mu) & = & (\rho \oplus \{x \mapsto \mu\,(\rho\, y, [\![e]\!]\,\rho)\}, \mu) \\
[\![y[e_1] = e_2;]\!]\,(\rho,\mu) & = & (\rho, \mu \oplus \{(\rho\, y, [\![e_1]\!]\,\rho) \mapsto [\![e_2]\!]\,\rho\})
\end{array}
$$



... in the Simple Example:



	x	y	$(0,1)$
0	$\emptyset$	$\emptyset$	$\emptyset$
1	$\{(0,1)\}$	$\emptyset$	$\emptyset$
2	$\{(0,1)\}$	$\{(1,2)\}$	$\emptyset$
3	$\{(0,1)\}$	$\{(1,2)\}$	$\{(1,2)\}$
4	$\{(0,1)\}$	$\{(1,2)\}$	$\{(1,2)\}$

The Effects of Edges:

destructive

$$
\begin{array}{lll}
[\![(\_, ;, \_)]\!]^\sharp\,(D,M) & = & (D,M) \\
[\![(\_, \mathrm{Pos}(e), \_)]\!]^\sharp\,(D,M) & = & (D,M) \\
[\![(\_, x = y;, \_)]\!]^\sharp\,(D,M) & = & (D \oplus \{x \mapsto D\,y\}, M) \\
[\![(\_, x = e;, \_)]\!]^\sharp\,(D,M) & = & (D \oplus \{x \mapsto \emptyset\}, M) \qquad , \qquad e \notin Vars \\
 & & \\
[\![(u, x = \mathsf{new}();, v)]\!]^\sharp\,(D,M) & = & (D \oplus \{x \mapsto \{(u,v)\}\}, M) \\
[\![(\_, x = y[e];, \_)]\!]^\sharp\,(D,M) & = & (D \oplus \{x \mapsto \bigcup\{M(f) \mid f \in D\,y\}\}, M) \\
[\![(\_, y[e_1] = x;, \_)]\!]^\sharp\,(D,M) & = & (D, M \oplus \{f \mapsto (M\,f \cup D\,x) \mid f \in D\,y\})
\end{array}
$$

non-destructive

Caveat:

- The value Null has been ignored. Dereferencing of Null or negative indices are not detected :-(
- Destructive updates are only possible for variables, not for blocks in storage!

  $\Longrightarrow$ no information, if not all block entries are initialized before use :-((
- The effects now depend on the edge itself.

  The analysis cannot be proven correct w.r.t. the reference semantics :-(

  In order to prove correctness, we first instrument the concrete semantics with extra information which records where a block has been created.

Caveat:

- The value Null has been ignored. Dereferencing of Null or negative indices are not detected :-(
- Destructive updates are only possible for variables, not for blocks in storage!

  $\Longrightarrow$ no information, if not all block entries are initialized before use :-((
- The effects now depend on the edge itself.

  The analysis cannot be proven correct w.r.t. the reference semantics :-(

  In order to prove correctness, we first instrument the concrete semantics with extra information which records where a block has been created.

...

- We compute possible points-to information.
- From that, we can extract may-alias information.
- The analysis can be rather expensive — without finding very much :-(
- Separate information for each program point can perhaps be abandoned ??

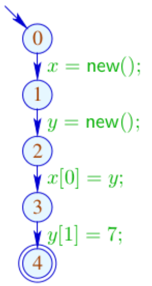
...

- We compute possible points-to information.
- From that, we can extract may-alias information.
- The analysis can be rather expensive — without finding very much :-(
- Separate information for each program point can perhaps be abandoned ??

Alias Analysis 2. Idea:

Compute for each variable and address a value which safely approximates the values at every program point simultaneously !

... in the Simple Example:



x	$\{(0,1)\}$
y	$\{(1,2)\}$
$(0,1)$	$\{(1,2)\}$
$(1,2)$	$\emptyset$

Each edge (u, lab, v) gives rise to constraints:

lab	$Constraint$
$x = y;$	$\mathcal{P}[x] \supseteq \mathcal{P}[y]$
$x = \mathsf{new}();$	$\mathcal{P}[x] \supseteq \{(u, v)\}$
$x = y[e];$	$\mathcal{P}[x] \supseteq \bigcup\{\mathcal{P}[f] \mid f \in \mathcal{P}[y]\}$
$y[e_1] = x;$	$\mathcal{P}[f] \supseteq (f \in \mathcal{P}[y])\,?\,\mathcal{P}[x] : \emptyset$
	for all $f \in Addr^\sharp$

Other edges have no effect :-)

Each edge (u, lab, v) gives rise to constraints:

lab	$Constraint$
$x = y;$	$\mathcal{P}[x] \supseteq \mathcal{P}[y]$
$x = \mathsf{new}();$	$\mathcal{P}[x] \supseteq \{(u, v)\}$
$x = y[e];$	$\mathcal{P}[x] \supseteq \bigcup\{\mathcal{P}[f] \mid f \in \mathcal{P}[y]\}$
$y[e_1] = x;$	$\mathcal{P}[f] \supseteq (f \in \mathcal{P}[y])\,?\,\mathcal{P}[x] : \emptyset$
	for all $f \in Addr^\sharp$

Other edges have no effect :-)