Script generated by TTT

Title: Seidl: Programmoptimierung (25.11.2015)

Date: Wed Nov 25 10:19:06 CET 2015

Duration: 90:59 min

Pages: 31

1.6 Pointer Analysis

Questions

→ Are two addresses possibly equal?

→ Are two addresses definitively equal?

1.6 Pointer Analysis

Questions

→ Are two addresses possibly equal? May Alias

→ Are two addresses definitively equal? Must Alias

⟹ Alias Analysis

The analyses so far without alias information

(1) Available Expressions:

- Extend the set *Expr* of expressions by occurring loads $M[e]$.
- Extend the Effects of Edges:

$$
\begin{array}{lll}
[\![x = e;]\!]^\sharp A & = & (A \cup \{e\}) \backslash Expr_x \\
[\![x = M[e];]\!]^\sharp A & = & (A \cup \{e, M[e]\}) \backslash Expr_x \\
[\![M[e_1] = e_2;]\!]^\sharp A & = & (A \cup \{e_1, e_2\}) \backslash Loads
\end{array}
$$

The analyses so far without alias information

(1) Available Expressions:

- Extend the set *Expr* of expressions by occurring loads $M[e]$.
- Extend the Effects of Edges:

$$\begin{array}{lll} [\![x = e;]\!]^\sharp\, A & = & (A \cup \{e\}) \backslash Expr_x \\ [\![x = M[e];]\!]^\sharp\, A & = & (A \cup \{e, M[e]\}) \backslash Expr_x \\ [\![M[e_1] = e_2;]\!]^\sharp\, A & = & (A \cup \{e_1, e_2\}) \backslash Loads \end{array}$$

(2) Values of Variables:

- Extend the set *Expr* of expressions by occurring loads $M[e]$.
- Extend the Effects of Edges:

$$[\![x = M[e];]\!]^\sharp\, V\, e' = \begin{cases} \{x\} & \text{if } e' = M[e] \\ \emptyset & \text{if } e' = e \\ V\, e' \backslash \{x\} & \text{otherwise} \end{cases}$$

$$[\![M[e_1] = e_2;]\!]^\sharp\, V\, e' = \begin{cases} \emptyset & \text{if } e' \in \{e_1, e_2\} \\ V\, e' & \text{otherwise} \end{cases}$$

(3) Constant Propagation:

- Extend the abstract state by an abstract store M
- Execute accesses to known memory locations!

$$[\![x = M[e];]\!]^\sharp\, (D, M) = \begin{cases} (D \oplus \{x \mapsto M\, a\}, M) & \text{if } [\![e]\!]^\sharp\, D = a \sqsubset \top \\ (D \oplus \{x \mapsto \top\}, M) & \text{otherwise} \end{cases}$$

$$[\![M[e_1] = e_2;]\!]^\sharp\, (D, M) = \begin{cases} (D, M \oplus \{a \mapsto [\![e_2]\!]^\sharp D\}) & \text{if } [\![e_1]\!]^\sharp\, D = a \sqsubset \top \\ (D, \underline{\top}) & \text{otherwise} \end{cases} \quad \text{where}$$

$$\underline{\top}\, a = \top \qquad (a \in \mathbb{N})$$

Problems

$\{1 \mapsto 5, 2 \mapsto 42\} \sqsubset \{2 \mapsto 42\}$

- Addresses are from $\mathbb{N}$.
  There are no infinite strictly ascending chains, but ...
- Exact addresses at compile-time are rarely known.
- At the same program point, typically different addresses are accessed ...
- Storing at an unknown address destroys all information M.

$\Longrightarrow$ constant propagation fails

$\Longrightarrow$ memory accesses/pointers kill precision

Problems

- Addresses are from $\mathbb{N}$.
  There are no infinite strictly ascending chains, but ...
- Exact addresses at compile-time are rarely known.
- At the same program point, typically different addresses are accessed ...
- Storing at an unknown address destroys all information M.

$\Longrightarrow$ constant propagation fails

$\Longrightarrow$ memory accesses/pointers kill precision

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.

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:

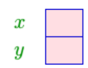
$x = \text{new}();$ // allocation of a new block

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

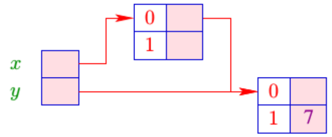
$y[e_1] = e_2;$ // indexed write access to a block

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

The Semantics

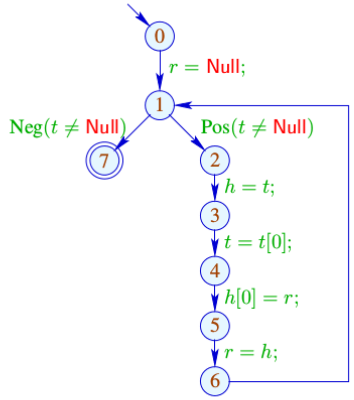


The Semantics



More Complex Example

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



Concrete Semantics

A store consists of a finite collection of blocks.

After h new-operations we obtain:

Addr$_h$	$=$	$\{\text{ref } a \mid 0 \leq a < h\}$	//	addresses
Val$_h$	$=$	$Addr_h \cup \mathbb{Z}$	//	values
Store$_h$	$=$	$(Addr_h \times \mathbb{N}_0) \to Val_h$	//	store
State$_h$	$=$	$(Vars \to Val_h) \times Store_h$	//	states

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

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

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

Let $(\rho, \mu) \in \mathit{State}_h$. Then we obtain for the new edges:

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

Caveat

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

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

are not considered as equivalent !!?

Possible Solution

Define equivalence only up to permutation of addresses !

Alias Analysis 1. Idea

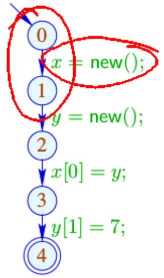
- Distinguish finitely many classes of blocks.
- Collect all addresses of a block into one set!
- Use sets of addresses as abstract values!

  $\Longrightarrow$ Points-to-Analysis

Addr$^\sharp$	$=$	Edges	//	creation edges
Val$^\sharp$	$=$	$2^{\mathit{Addr}^\sharp}$	//	abstract values
Store$^\sharp$	$=$	$\mathit{Addr}^\sharp \to \mathit{Val}^\sharp$	//	abstract store
State$^\sharp$	$=$	$(\mathit{Vars} \to \mathit{Val}^\sharp) \times \mathit{Store}^\sharp$	//	abstract states

// complete lattice !!!

... 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	$\{(0,1)\}$	$\{(1,2)\}$	$\{(1,2)\}$

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!

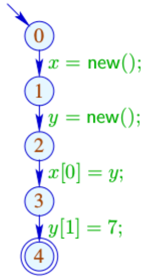
$\Longrightarrow$ Points-to-Analysis

$$\begin{array}{lll} Addr^\sharp & = & Edges \\ Val^\sharp & = & 2^{Addr^\sharp} \\ Store^\sharp & = & Addr^\sharp \to Val^\sharp \\ State^\sharp & = & (Vars \to Val^\sharp) \times Store^\sharp \end{array}$$

// creation edges
// abstract values
// abstract store
// abstract states

// complete lattice !!!

... 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

$$\begin{array}{lll} [\![(\_, ;, \_)]\!]^\sharp (D, M) & = & (D, M) \\ [\![(\_, \text{Pos}(e), \_)]\!]^\sharp (D, M) & = & (D, M) \\ [\![(\_, x = y;, \_)]\!]^\sharp (D, M) & = & (D \oplus \{x \mapsto D\, y\}, M) \\ [\![(\_, x = e;, \_)]\!]^\sharp (D, M) & = & (D \oplus \{x \mapsto \emptyset\}, M) \qquad , \quad e \notin Vars \end{array}$$

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

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.

The Effects of Edges

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

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.

375

...

- 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 ??

$x \mapsto \{(1,3),(5,7)\}$

$y \mapsto \{(4,6)\}(1,3)\}$

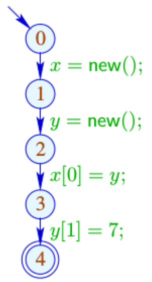
...

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

Other edges have no effect.

Discussion

- The resulting constraint system has size $\mathcal{O}(k \cdot n)$ for k abstract addresses and n edges.
- The number of necessary iterations is $\mathcal{O}(k(k + \#Vars))$ ...
- The computed information is perhaps still too zu precise !!?
- In order to prove correctness of a solution $s^\sharp \in States^\sharp$ we show: