Script **generated by TTT**

Title: Seidl: Programmoptimierung (02.12.2013)

Date: Mon Dec 02 14:16:36 CET 2013

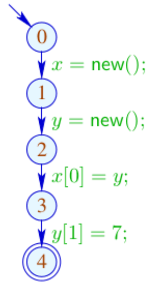
Duration: 88:24 min

Pages: 40

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

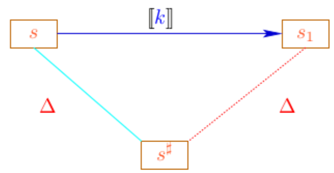
Each edge (u, lab, v) gives rise to constraints:

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:



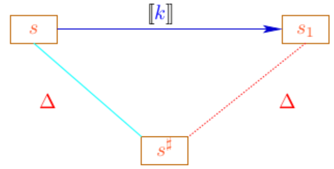
Each edge (u, lab, v) gives rise to constraints:

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:

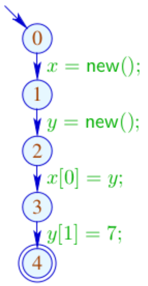
- The resulting constraint system has size $\mathcal{O}(k \cdot n)$ for k abstract addresses and n edges :-(
- The number of necessary iterations is $\mathcal{O}(k \cdot (k + \#Vars))$ ...
- The computed information is perhaps still too zu precise !!?
- In order to prove correctness of a solution $s^\sharp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



Alias Analysis 3. Idea:

Determine one equivalence relation $\equiv$ on variables x and memory accesses $y[\,]$ with $s_1 \equiv s_2$ whenever s_1, s_2 may contain the same address at some u_1, u_2

... in the Simple Example:

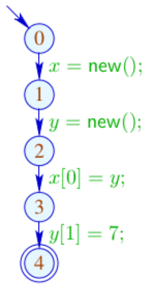


$$\equiv \; = \; \{\{x\}, \{y, x[\,]\}, \{y[\,]\}\}$$

Alias Analysis 3. Idea:

Determine one equivalence relation $\equiv$ on variables x and memory accesses $y[\,]$ with $s_1 \equiv s_2$ whenever s_1, s_2 may contain the same address at some u_1, u_2

... in the Simple Example:



$$\equiv \; = \; \{\{x\}, \{y, x[\,]\}, \{y[\,]\}\}$$

Discussion:

→ We compute a single information fo the whole program.

→ The computation of this information maintains partitions $\pi = \{P_1, \ldots, P_m\}$:-)

→ Individual sets P_i are identified by means of representatives $p_i \in P_i$.

→ The operations on a partition π are:

$$
\begin{array}{lcll}
\textsf{find}\,(\pi, p) & = & p_i & \text{if } p \in P_i \\
 & // & \text{returns the representative} & \\
\textsf{union}\,(\pi, p_{i_1}, p_{i_2}) & = & \{P_{i_1} \cup P_{i_2}\} \cup \{P_j \mid i_1 \neq j \neq i_2\} & \\
 & // & \text{unions the represented classes} &
\end{array}
$$

→ If $x_1, x_2 \in \mathit{Vars}$ are equivalent, then also $x_1[\,]$ and $x_2[\,]$ must be equivalent :-)

→ If $P_i \cap \mathit{Vars} \neq \emptyset$, then we choose $p_i \in \mathit{Vars}$. Then we can apply union recursively :

$$
\begin{array}{lcl}
\textsf{union}^*\,(\pi, q_1, q_2) & = & \textsf{let } p_{i_1} = \textsf{find}\,(\pi, q_1) \\
 & & \qquad p_{i_2} = \textsf{find}\,(\pi, q_2) \\
 & & \textsf{in if } p_{i_1} == p_{i_2} \textsf{ then } \pi \\
 & & \quad \textsf{else let } \pi = \textsf{union}\,(\pi, p_{i_1}, p_{i_2}) \\
 & & \qquad \textsf{in if } p_{i_1}, p_{i_2} \in \mathit{Vars} \textsf{ then} \\
 & & \qquad \quad \textsf{union}^*\,(\pi, p_{i_1}[\,], p_{i_2}[\,])
\end{array}
$$

else π

→ If $x_1, x_2 \in \mathit{Vars}$ are equivalent, then also $x_1[\,]$ and $x_2[\,]$ must be equivalent :-)

→ If $P_i \cap \mathit{Vars} \neq \emptyset$, then we choose $p_i \in \mathit{Vars}$. Then we can apply union recursively :

$$
\begin{array}{lcl}
\textsf{union}^*\,(\pi, q_1, q_2) & = & \textsf{let } p_{i_1} = \textsf{find}\,(\pi, q_1) \\
 & & \qquad p_{i_2} = \textsf{find}\,(\pi, q_2) \\
 & & \textsf{in if } p_{i_1} == p_{i_2} \textsf{ then } \pi \\
 & & \quad \textsf{else let } \pi = \textsf{union}\,(\pi, p_{i_1}, p_{i_2}) \\
 & & \qquad \textsf{in if } p_{i_1}, p_{i_2} \in \mathit{Vars} \textsf{ then} \\
 & & \qquad \quad \textsf{union}^*\,(\pi, p_{i_1}[\,], p_{i_2}[\,])
\end{array}
$$

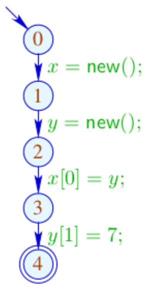
The analysis iterates over all edges once:

$$
\begin{array}{l}
\pi = \{\{x\}, \{x[\,]\} \mid x \in \mathit{Vars}\}; \\
\textsf{forall} \quad k = (\_, \mathit{lab}, \_) \quad \textsf{do} \quad \pi = [\![\mathit{lab}]\!]^\sharp\,\pi;
\end{array}
$$

where:

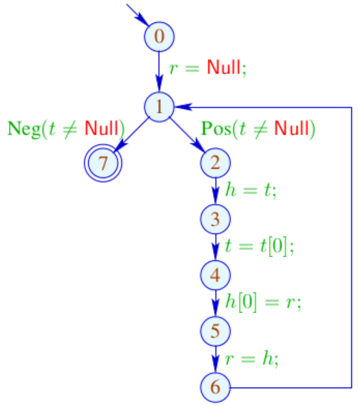
$$
\begin{array}{lcll}
[\![x = y;]\!]^\sharp\,\pi & = & \textsf{union}^*\,(\pi, x, y) & \\
[\![x = y[e];]\!]^\sharp\,\pi & = & \textsf{union}^*\,(\pi, x, y[\,]) & \\
[\![y[e] = x;]\!]^\sharp\,\pi & = & \textsf{union}^*\,(\pi, x, y[\,]) & \\
[\![\mathit{lab}]\!]^\sharp\,\pi & = & \pi & \text{otherwise}
\end{array}
$$

... in the Simple Example:



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

... in the More Complex Example:



	$\{\{h\},\{r\},\{t\},\{h[\,]\},\{t[\,]\}\}$
$(2,3)$	$\{\{h,t\},\{r\},\{h[\,],t[\,]\}\}$
$(3,4)$	$\{\{h,t,h[\,],t[\,]\},\{r\}\}$
$(4,5)$	$\{\{h,t,r,h[\,],t[\,]\}\}$
$(5,6)$	$\{\{h,t,r,h[\,],t[\,]\}\}$

Caveat:

In order to find something, we must assume that variables / addresses always receive a value before they are accessed.

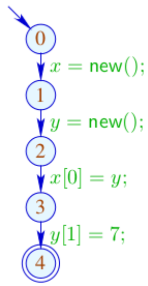
Complexity:

we have:

$\mathcal{O}(\#\,\textit{edges} + \#\,\textit{Vars})$	calls of	union*
$\mathcal{O}(\#\,\textit{edges} + \#\,\textit{Vars})$	calls of	find
$\mathcal{O}(\#\,\textit{Vars})$	calls of	union

$\Longrightarrow$ We require efficient Union-Find data-structure :-)

... in the Simple Example:



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

Caveat:

In order to find something, we must assume that variables / addresses always receive a value before they are accessed.

Complexity:

we have:

$\mathcal{O}(\# \, edges + \# \, Vars)$	calls of	union*
$\mathcal{O}(\# \, edges + \# \, Vars)$	calls of	find
$\mathcal{O}(\# \, Vars)$	calls of	union

$\Longrightarrow$ We require efficient Union-Find data-structure :-)

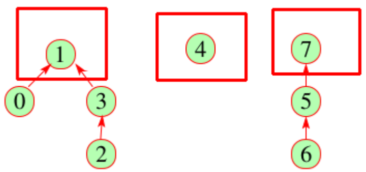
Idea:

Represent partition of U as directed forest:

- For $u \in U$ a reference $F[u]$ to the father is maintained;
- Roots are elements u with $F[u] = u$.

Single trees represent equivalence classes.

Their roots are their representatives ...



0	1	2	3	4	5	6	7
1	1	3	1	4	7	5	7

$\rightarrow$ find (π, u) follows the father references :-)

$\rightarrow$ union (π, u_1, u_2) re-directs the father reference of one u_i ...

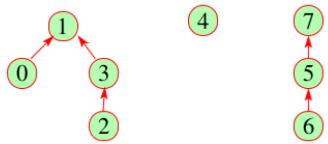
Idea:

Represent partition of U as directed forest:

- For $u \in U$ a reference $F[u]$ to the father is maintained;
- Roots are elements u with $F[u] = u$.

Single trees represent equivalence classes.

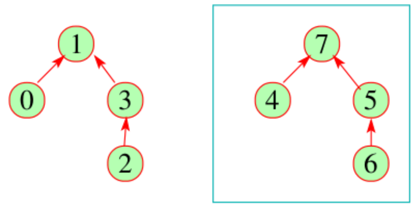
Their roots are their representatives ...



0	1	2	3	4	5	6	7
1	1	3	1	4	7	5	7

→ find (π, u) follows the father references :-)

→ union (π, u_1, u_2) re-directs the father reference of one u_i ...



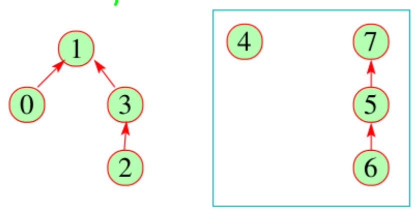
0	1	2	3	4	5	6	7
1	1	3	1	7	7	5	7

The Costs:

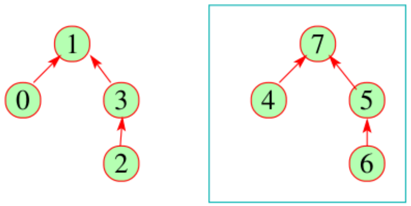
union	:	$\mathcal{O}(1)$	:-)
find	:	$\mathcal{O}(depth(\pi))$	:-(

Strategy to Avoid Deep Trees:

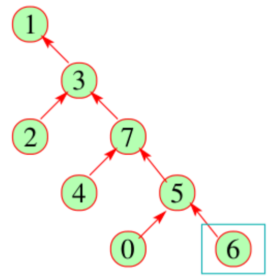
- Put the smaller tree below the bigger !
- Use find to compress paths ...



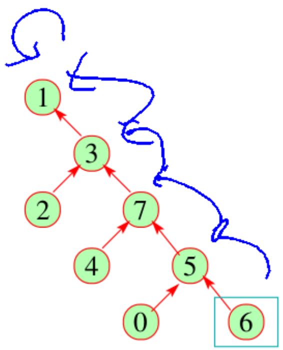
0	1	2	3	4	5	6	7
1	1	3	1	4	7	5	7



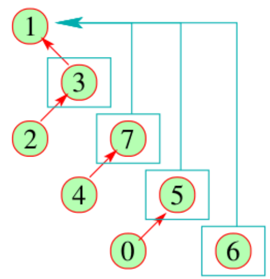
0	1	2	3	4	5	6	7
1	1	3	1	7	7	5	7



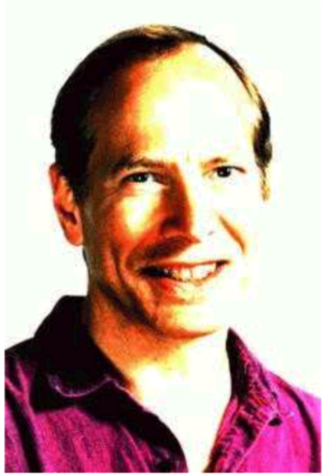
0	1	2	3	4	5	6	7
5	1	3	1	7	7	5	3



0	1	2	3	4	5	6	7
5	1	3	1	7	7	5	3



0	1	2	3	4	5	6	7
5	1	3	1	1	7	1	1



Robert Endre Tarjan, Princeton

Note:

- By this data-structure, n union- und m find operations require time $\mathcal{O}(n + m \cdot \alpha(n,n))$

  // α the inverse Ackermann-function :-)
- For our application, we only must modify union such that roots are from *Vars* whenever possible.
- This modification does not increase the asymptotic run-time. :-)

Summary:

The analysis is extremely fast — but may not find very much.

Note:

- By this data-structure, n union- und m find operations require time $\mathcal{O}(n + m \cdot \alpha(n,n))$

  // α the inverse Ackermann-function :-)
- For our application, we only must modify union such that roots are from *Vars* whenever possible.
- This modification does not increase the asymptotic run-time. :-)

Summary:

The analysis is extremely fast — but may not find very much.

Background 3: Fixpoint Algorithms

Consider: $x_i \sqsupseteq f_i(x_1, \ldots, x_n), \quad i = 1, \ldots, n$

Observation:

RR-Iteration is inefficient:

→ We require a complete round in order to detect termination :-(

→ If in some round, the value of just one unknown is changed, then we still re-compute all :-(

→ The practical run-time depends on the ordering on the variables :-(

Idea: Worklist Iteration

If an unknown x_i changes its value, we re-compute all unknowns which depend on x_i . Technically, we require:

→ the lists $Dep\ f_i$ of unknowns which are accessed during evaluation of f_i. From that, we compute the lists:

$$I[x_i] = \{x_j \mid x_i \in Dep\ f_j\}$$

i.e., a list of all x_j which depend on the value of x_i ;

→ the values $D[x_i]$ of the x_i where initially $D[x_i] = \bot$;

→ a list W of all unknowns whose value must be recomputed ...

The Algorithm:

$f_i : (Vars \to \mathbb{D}) \to \mathbb{D}$

```
W = [x_1, ..., x_n];
while (W ≠ []) {
    x_i = extract W;
    t = f_i eval;
    t = D[x_i] ⊔ t;
    if (t ≠ D[x_i]) {
        D[x_i] = t;
        W = append I[x_i] W;
    }
}
```

where : $eval\ x_j = D[x_j]$

Example:

$$x_1 \supseteq \{a\} \cup x_3$$
$$x_2 \supseteq x_3 \cap \{a, b\}$$
$$x_3 \supseteq x_1 \cup \{c\}$$

	I
x_1	$\{x_3\}$
x_2	$\emptyset$
x_3	$\{x_1, x_2\}$

Example:

$$x_1 \supseteq \{a\} \cup x_3$$
$$x_2 \supseteq x_3 \cap \{a, b\}$$
$$x_3 \supseteq x_1 \cup \{c\}$$

	I
x_1	$\{x_3\}$
x_2	$\emptyset$
x_3	$\{x_1, x_2\}$

$D[x_1]$	$D[x_2]$	$D[x_3]$	W	
$\emptyset$	$\emptyset$	$\emptyset$	x_1, x_2, x_3	1
$\{a\}$	$\emptyset$	$\emptyset$	x_2, x_3	2
$\{a\}$	$\emptyset$	$\emptyset$	x_3	3
$\{a\}$	$\emptyset$	$\{a, c\}$	x_1, x_2	4
$\{a, c\}$	$\emptyset$	$\{a, c\}$	x_3, x_2	5
$\{a, c\}$	$\emptyset$	$\{a, c\}$	x_2	6
$\{a, c\}$	$\{a\}$	$\{a, c\}$	[]	

Example:

$$x_1 \supseteq \{a\} \cup x_3$$
$$x_2 \supseteq x_3 \cap \{a, b\}$$
$$x_3 \supseteq x_1 \cup \{c\}$$

	I
x_1	$\{x_3\}$
x_2	$\emptyset$
x_3	$\{x_1, x_2\}$

$D[x_1]$	$D[x_2]$	$D[x_3]$	W
$\emptyset$	$\emptyset$	$\emptyset$	x_1, x_2, x_3
$\{a\}$	$\emptyset$	$\emptyset$	x_2, x_3
$\{a\}$	$\emptyset$	$\emptyset$	x_3
$\{a\}$	$\emptyset$	$\{a, c\}$	x_1, x_2
$\{a, c\}$	$\emptyset$	$\{a, c\}$	x_3, x_2
$\{a, c\}$	$\emptyset$	$\{a, c\}$	x_2
$\{a, c\}$	$\{a\}$	$\{a, c\}$	[ ]

Theorem

Let $x_i \sqsupseteq f_i(x_1, \ldots, x_n)$, $i = 1, \ldots, n$ denote a constraint system over the complete lattice $\mathbb{D}$ of hight $h > 0$.

(1) The algorithm terminates after at most $h \cdot N$ evaluations of right-hand sides where

$$N = \sum_{i=1}^{n} (1 + \#(Dep\ f_i)) \qquad // \text{ size of the system } :-)$$

(2) The algorithm returns a solution.
If all f_i are monotonic, it returns the least one.

Proof:

Ad (1):

Every unknown x_i may change its value at most h times :-)

Each time, the list $I[x_i]$ is added to W.

Thus, the total number of evaluations is:

$$\begin{aligned} &\leq n + \textstyle\sum_{i=1}^{n} (h \cdot \#(I[x_i])) \\ &= n + h \cdot \textstyle\sum_{i=1}^{n} \#(I[x_i]) \\ &= n + h \cdot \textstyle\sum_{i=1}^{n} \#(Dep\ f_i) \\ &\leq h \cdot \textstyle\sum_{i=1}^{n} (1 + \#(Dep\ f_i)) \\ &= h \cdot N \end{aligned}$$

Theorem

Let $x_i \sqsupseteq f_i(x_1, \ldots, x_n)$, $i = 1, \ldots, n$ denote a constraint system over the complete lattice $\mathbb{D}$ of hight $h > 0$.

(1) The algorithm terminates after at most $h \cdot N$ evaluations of right-hand sides where

$$N = \sum_{i=1}^{n} (1 + \#(Dep\ f_i)) \qquad // \text{ size of the system } :-)$$

(2) The algorithm returns a solution.
If all f_i are monotonic, it returns the least one.

Proof:

Ad (1):

Every unknown x_i may change its value at most h times :-)

Each time, the list $I[x_i]$ is added to W .

Thus, the total number of evaluations is:

$$
\begin{aligned}
&\leq && n + \sum_{i=1}^{n}(h \cdot \#(I[x_i])) \\
&= && n + h \cdot \sum_{i=1}^{n} \#(I[x_i]) \\
&= && n + h \cdot \sum_{i=1}^{n} \#(Dep\ f_i) \\
&\leq && h \cdot \sum_{i=1}^{n}(1 + \#(Dep\ f_i)) \\
&= && h \cdot N
\end{aligned}
$$