Script generated by TTT

Title: Seidl: Programmoptimierung (26.11.2015)

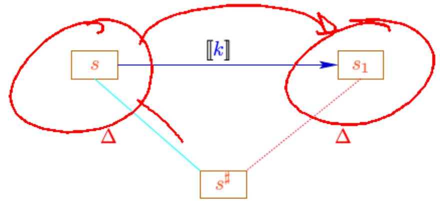
Date: Thu Nov 26 08:35:22 CET 2015

Duration: 90:23 min

Pages: 30

Discussion

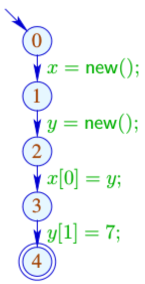
- The resulting constraint system has size $\mathcal{O}(k \cdot n)$ for k abstract addresses and n edges.
- The number of necessary iterations is $\mathcal{O}(k(k + \#Vars))$ ...
- The computed information is perhaps still too zu precise !!?
- In order to prove correctness of a solution $s^\sharp \in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



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:

$$\boxed{\text{find}}\,(\pi, p) \;=\; p_i \qquad \text{if } p \in P_i$$

// returns the representative

$$\boxed{\text{union}}\,(\pi, p_{i_1}, p_{i_2}) \;=\; \{P_{i_1} \cup P_{i_2}\} \cup \{P_j \mid i_1 \neq j \neq i_2\}$$

// unions the represented classes

→ 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 :

$$
\begin{array}{lll}
\mathsf{union}^*\,(\pi, q_1, q_2) & = & \mathsf{let}\;\; p_{i_1} = \mathsf{find}\,(\pi, q_1) \\
 & & \phantom{\mathsf{let}\;\;} p_{i_2} = \mathsf{find}\,(\pi, q_2) \\
 & & \mathsf{in}\;\; \mathsf{if}\; p_{i_1} == p_{i_2}\; \mathsf{then}\; \pi \\
 & & \phantom{\mathsf{in}\;\;} \mathsf{else}\;\; \mathsf{let}\; \pi = \mathsf{union}\,(\pi, p_{i_1}, p_{i_2}) \\
 & & \phantom{\mathsf{in}\;\; \mathsf{else}\;\;} \mathsf{in}\;\; \mathsf{if}\; p_{i_1}, p_{i_2} \in \mathit{Vars}\; \mathsf{then} \\
 & & \phantom{\mathsf{in}\;\; \mathsf{else}\;\; \mathsf{in}\;\;} \mathsf{union}^*\,(\pi, p_{i_1}[\,], p_{i_2}[\,]) \\
 & & \phantom{\mathsf{in}\;\; \mathsf{else}\;\; \mathsf{in}\;\;} \mathsf{else}\; \pi
\end{array}
$$

x[] z y[] z'

x y

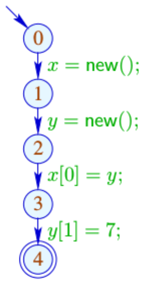
The analysis iterates over all edges once:

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

where:

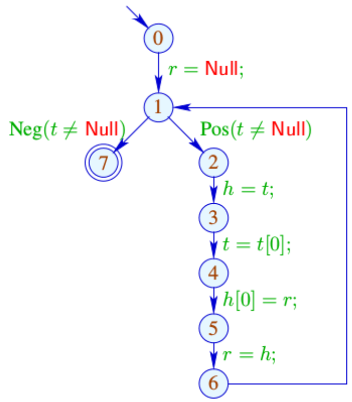
$$
\begin{array}{lll}
[\![x = y;]\!]^\sharp\, \pi & = & \mathsf{union}^*\,(\pi, x, y) \\
[\![x = y[e];]\!]^\sharp\, \pi & = & \mathsf{union}^*\,(\pi, x, y[\,]) \\
[\![y[e] = x;]\!]^\sharp\, \pi & = & \mathsf{union}^*\,(\pi, x, y[\,]) \\
[\![\mathit{lab}]\!]^\sharp\, \pi & = & \pi \qquad \text{otherwise}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

... in the Simple Example



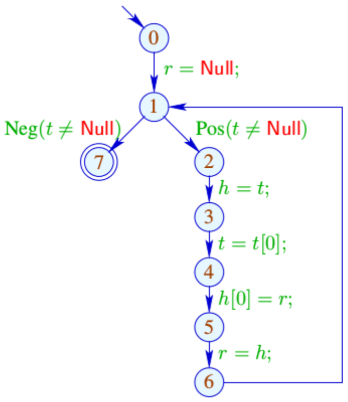
	$\{\{x\}, \{y\}, \{x[\,]\}, \{y[\,]\}\}$
$(0,1)$	$\{\{x\}, \{y\}, \{x[\,]\}, \{y[\,]\}\}$
$(1,2)$	$\{\{x\}, \{y\}, \{x[\,]\}, \{y[\,]\}\}$
$(2,3)$	$\{\{x\}, \boxed{\{y, x[\,]\}}, \{y[\,]\}\}$
$(3,4)$	$\{\{x\}, \{y, x[\,]\}, \{y[\,]\}\}$

... in the More Complex Example



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

... in the More Complex Example



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

Caveat

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

Complexity

we have:

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

$\Longrightarrow$ We require efficient Union-Find data-structure ...

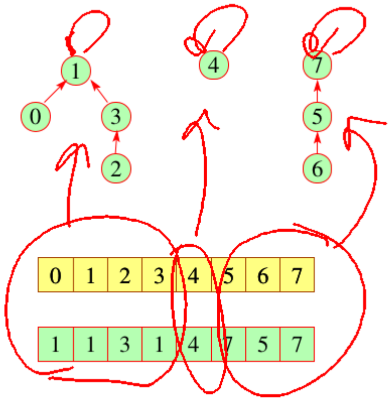
Idea

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 $F[u] = u$.

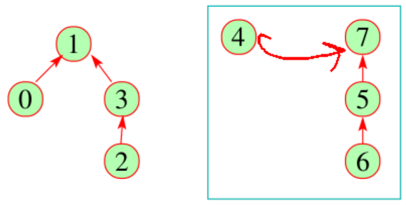
Single trees represent equivalence classes.

Their roots are their representatives ...



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

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



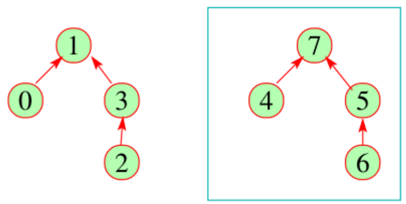
0	1	2	3	4	5	6	7
1	1	3	1	4	7	5	7

The Costs

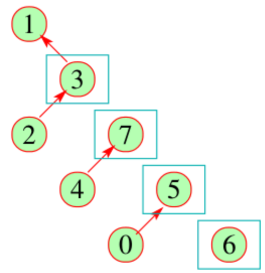
union	:	$\mathcal{O}(1)$
find	:	$\mathcal{O}(depth(\pi))$

Strategy to Avoid Deep Trees

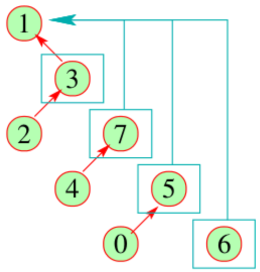
- Put the smaller tree below the bigger !
- Use find to compress paths ...



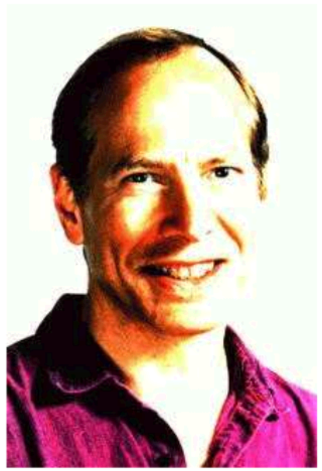
0	1	2	3	4	5	6	7
1	1	3	1	7	7	5	7



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	1	7	1	1



Robert Endre Tarjan, Princeton

Remark

- 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.

$n^2 \cdot \ell$

Remark

- 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.

400

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.

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.

401

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

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

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

$W = [x_1, \ldots, x_n];$

while $(W \neq [\,])$ {

  $x_i \;=\;$ extract $W;$

  $t \;=\;$ f_i eval;

  if $(t \not\sqsubseteq D[x_i])$ {

    $D[x_i] \;=\; D[x_i] \sqcup t;$

    $W \;=\;$ append $I[x_i]\ W;$

  }

}

$$\text{where}: \quad \mathit{eval}\ x_j \;=\; D[x_j]$$

Example

$$\begin{array}{lcl} x_1 & \supseteq & \{a\} \cup x_3 \\ x_2 & \supseteq & x_3 \cap \{a,b\} \\ x_3 & \supseteq & x_1 \cup \{c\} \end{array}$$

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

Example

$$\begin{array}{lcl} x_1 & \supseteq & \{a\} \cup x_3 \\ x_2 & \supseteq & x_3 \cap \{a,b\} \\ x_3 & \supseteq & x_1 \cup \{c\} \end{array}$$

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

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

Example

$$\begin{array}{lcl} x_1 & \supseteq & \{a\} \cup x_3 \\ x_2 & \supseteq & x_3 \cap \{a,b\} \\ x_3 & \supseteq & x_1 \cup \{c\} \end{array}$$

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

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

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 height $h > 0$.

(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:

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

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:

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

407

Ad (2):

We only consider the assertion for monotonic f_i .

Let D_0 denote the least solution. We show:

- $D_0[x_i] \sqsupseteq D[x_i]$ (all the time)
- $D[x_i] \not\sqsupseteq f_i$ eval $\Longrightarrow x_i \in W$ (at exit of the loop body)
- On termination, the algo returns a solution