Script generated by TTT

Title: Seidl: Programmoptimierung (21.10.2013)

Date: Mon Oct 21 14:02:56 CEST 2013

Duration: 88:59 min

Pages: 33

1 Removing superfluous computations

1.1 Repeated computations

Idea:

If the same value is computed repeatedly, then

→ store it after the first computation;

→ replace every further computation through a look-up!

$\Longrightarrow$ Availability of expressions

$\Longrightarrow$ Memoization

Problem: Identify repeated computations!

Example:

$$\begin{array}{lrcl} & z & = & 1; \\ & y & = & M[17]; \\ A: & x_1 & = & \boxed{y+z}; \\ & & \ldots & \\ B: & x_2 & = & \boxed{y+z}; \end{array}$$

Note:

B is a repeated computation of the value of $\boxed{y+z}$, if:

(1) A is always executed before B; and

(2) y and z at B have the same values as at A :-)

$\Longrightarrow$ We need:

→ an operational semantics :-)

→ a method which identifies at least some repeated computations ...

Note:

B is a repeated computation of the value of $\boxed{y + z}$, if:

(1) A is always executed before B; and

(2) y and z at B have the same values as at A :-)

$\Longrightarrow$ We need:

$\rightarrow$ an operational semantics :-)

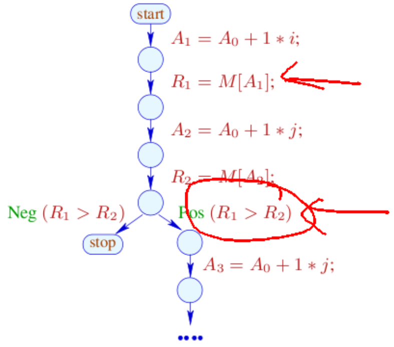
$\rightarrow$ a method which identifies at least some repeated computations ...

Background 1: An Operational Semantics

we choose a small-step operational approach.

Programs are represented as control-flow graphs.

In the example:



Thereby, represent:

vertex	program point
start	programm start
stop	program exit
edge	step of computation

Thereby, represent:

vertex	program point
start	programm start
stop	program exit
edge	step of computation

Edge Labelings:

Test :	Pos (e) or Neg (e)
Assignment :	$R = e;$
Load :	$R = M[e];$
Store :	$M[e_1] = e_2;$
Nop :	;

$x = M[-1]$

Computations follow paths.

Computations transform the current state

$$s = (\rho, \mu)$$

where:

$\rho : \mathit{Vars} \to \mathbf{int}$	contents of registers
$\mu : \mathbb{N} \to \mathbf{int}$	contents of storage

Every edge $k = (u, \mathit{lab}, v)$ defines a partial transformation

$$[\![k]\!] = [\![\mathit{lab}]\!]$$

of the state:

$$
\begin{array}{lcll}
[\![;]\!]\,(\rho,\mu) & = & (\rho,\mu) & \\
 & & & \\
[\![\mathrm{Pos}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho \neq 0 \\
[\![\mathrm{Neg}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho = 0
\end{array}
$$

$$
\begin{array}{lcll}
[\![;]\!]\,(\rho,\mu) & = & (\rho,\mu) & \\
 & & & \\
[\![\mathrm{Pos}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho \neq 0 \\
[\![\mathrm{Neg}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho = 0
\end{array}
$$

// $[\![e]\!]$: evaluation of the expression e, e.g.

// $[\![x+y]\!]\,\{x \mapsto 7, y \mapsto -1\} = 6$

// $[\![!(x == 4)]\!]\,\{x \mapsto 5\} = 1$

$$
\begin{array}{lcll}
[\![;]\!]\,(\rho,\mu) & = & (\rho,\mu) & \\
 & & & \\
[\![\mathrm{Pos}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho \neq 0 \\
[\![\mathrm{Neg}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho = 0
\end{array}
$$

// $[\![e]\!]$: evaluation of the expression e, e.g.

// $[\![x+y]\!]\,\{x \mapsto 7, y \mapsto -1\} = 6$

// $[\![!(x == 4)]\!]\,\{x \mapsto 5\} = 1$

$$[\![R = e;]\!]\,(\rho,\mu) \quad = \quad (\boxed{\rho \oplus \{R \mapsto [\![e]\!]\,\rho\}}, \mu)$$

// where "$\oplus$" modifies a mapping at a given argument

$\{x \mapsto 7, y \mapsto -1\} \oplus \{y$

$$
\begin{aligned}
[\![;]\!]\,(\rho,\mu) &= (\rho,\mu) \\
[\![\mathrm{Pos}\,(e)]\!]\,(\rho,\mu) &= (\rho,\mu) && \text{if } [\![e]\!]\,\rho \neq 0 \\
[\![\mathrm{Neg}\,(e)]\!]\,(\rho,\mu) &= (\rho,\mu) && \text{if } [\![e]\!]\,\rho = 0
\end{aligned}
$$

// $[\![e]\!]$: evaluation of the expression e, e.g.

// $[\![x+y]\!]\,\{x \mapsto 7, y \mapsto -1\} = 6$

// $[\![!(x == 4)]\!]\,\{x \mapsto 5\} = 1$

$\{x \mapsto 7, y \mapsto -1\} \oplus \{y \mapsto 6\}$

$$
\begin{aligned}
[\![;]\!]\,(\rho,\mu) &= (\rho,\mu) \\
[\![\mathrm{Pos}\,(e)]\!]\,(\rho,\mu) &= (\rho,\mu) && \text{if } [\![e]\!]\,\rho \neq 0 \\
[\![\mathrm{Neg}\,(e)]\!]\,(\rho,\mu) &= (\rho,\mu) && \text{if } [\![e]\!]\,\rho = 0
\end{aligned}
$$

// $[\![e]\!]$: evaluation of the expression e, e.g.

// $[\![x+y]\!]\,\{x \mapsto 7, y \mapsto -1\} = 6$

// $[\![!(x == 4)]\!]\,\{x \mapsto 5\} = 1$

$$[\![R = e;]\!]\,(\rho,\mu) = (\boxed{\rho \oplus \{R \mapsto [\![e]\!]\,\rho\}}, \mu)$$

// where "$\oplus$" modifies a mapping at a given argument

$$
\begin{aligned}
[\![R = M[e];]\!]\,(\rho,\mu) &= (\boxed{\rho \oplus \{R \mapsto \mu([\![e]\!]\,\rho))\}}, \mu) \\
[\![M[e_1] = e_2;]\!]\,(\rho,\mu) &= (\rho, \boxed{\mu \oplus \{[\![e_1]\!]\,\rho \mapsto [\![e_2]\!]\,\rho\}})
\end{aligned}
$$

Example:

$[\![x = x+1;]\!]\,(\{x \mapsto 5\}, \mu) = (\rho, \mu)$ where:

$$
\begin{aligned}
\rho &= \{x \mapsto 5\} \oplus \{x \mapsto [\![x+1]\!]\,\{x \mapsto 5\}\} \\
&= \{x \mapsto 5\} \oplus \{x \mapsto 6\} \\
&= \{x \mapsto 6\}
\end{aligned}
$$

$$
\begin{aligned}
[\![R = M[e];]\!]\,(\rho,\mu) &= (\boxed{\rho \oplus \{R \mapsto \mu([\![e]\!]\,\rho))\}}, \mu) \\
[\![M[e_1] = e_2;]\!]\,(\rho,\mu) &= (\rho, \boxed{\mu \oplus \{[\![e_1]\!]\,\rho \mapsto [\![e_2]\!]\,\rho\}})
\end{aligned}
$$

Example:

$[\![x = x+1;]\!]\,(\{x \mapsto 5\}, \mu) = (\rho, \mu)$ where:

$$
\begin{aligned}
\rho &= \{x \mapsto 5\} \oplus \{x \mapsto [\![x+1]\!]\,\{x \mapsto 5\}\} \\
&= \{x \mapsto 5\} \oplus \{x \mapsto 6\} \\
&= \{x \mapsto 6\}
\end{aligned}
$$

A path $\pi = k_1 k_2 \ldots k_m$ is a computation for the state s if:

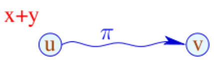
$$s \in def\ ([\![k_m]\!] \circ \ldots \circ [\![k_1]\!])$$

The result of the computation is:

$$[\![\pi]\!]\ s = ([\![k_m]\!] \circ \ldots \circ [\![k_1]\!])\ s$$

Application:

Assume that we have computed the value of $x + y$ at program point u:



We perform a computation along path π and reach v where we evaluate again $x + y$ ...

Idea:

If x and y have not been modified in π, then evaluation of $x + y$ at v must return the same value as evaluation at u :-)

We can check this property at every edge in π :-}

Idea:

If x and y have not been modified in π, then evaluation of $x + y$ at v must return the same value as evaluation at u :-)

We can check this property at every edge in π :-}

More generally:

Assume that the values of the expressions $A = \{e_1, \ldots, e_r\}$ are available at u.

A path $\pi = k_1 k_2 \ldots k_m$ is a computation for the state s if:

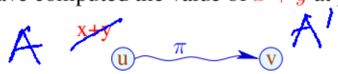
$$s \in def\ ([\![k_m]\!] \circ \ldots \circ [\![k_1]\!])$$

The result of the computation is:

$$[\![\pi]\!]\ s = ([\![k_m]\!] \circ \ldots \circ [\![k_1]\!])\ s$$

Application:

Assume that we have computed the value of $x + y$ at program point u:



We perform a computation along path π and reach v where we evaluate again $x + y$ ...

Idea:

If x and y have not been modified in π, then evaluation of $x + y$ at v must return the same value as evaluation at u :-)

We can check this property at every edge in π :-}

More generally:

Assume that the values of the expressions $A = \{e_1, \ldots, e_r\}$ are available at u.

Every edge k transforms this set into a set $[\![k]\!]^\sharp\, A$ of expressions whose values are available after execution of k ...

... which transformations can be composed to the effect of a path $\pi = k_1 \ldots k_r$:

$$[\![\pi]\!]^\sharp = [\![k_r]\!]^\sharp \circ \ldots \circ [\![k_1]\!]^\sharp$$

... which transformations can be composed to the effect of a path $\pi = k_1 \ldots k_r$:

$$[\![\pi]\!]^\sharp = [\![k_r]\!]^\sharp \circ \ldots \circ [\![k_1]\!]^\sharp$$

The effect $[\![k]\!]^\sharp$ of an edge $k = (u, lab, v)$ only depends on the label lab, i.e., $[\![k]\!]^\sharp = [\![lab]\!]^\sharp$

... which transformations can be composed to the effect of a path $\pi = k_1 \ldots k_r$:

$$[\![\pi]\!]^\sharp = [\![k_r]\!]^\sharp \circ \ldots \circ [\![k_1]\!]^\sharp$$

The effect $[\![k]\!]^\sharp$ of an edge $k = (u, lab, v)$ only depends on the label lab, i.e., $[\![k]\!]^\sharp = [\![lab]\!]^\sharp$ where:

$$\begin{array}{lll}
[\![;]\!]^\sharp\, A & = & A \\
[\![Pos(e)]\!]^\sharp\, A & = & [\![Neg(e)]\!]^\sharp\, A \quad = \quad A \cup \{e\} \\
[\![x = e;]\!]^\sharp\, A & = & (A \cup \{e\}) \backslash Expr_x \quad \text{where} \\
 & & Expr_x \text{ all expressions which contain } x
\end{array}$$

x = x + 1

$M[e]$

$$\begin{aligned}
[\![x = M[e];]\!]^\sharp\, A &= (A \cup \{e\}) \backslash \mathit{Expr}_x \\
[\![M[e_1] = e_2;]\!]^\sharp\, A &= \big(A \cup \{e_1, e_2\}\big) \backslash \mathit{Loads}
\end{aligned}$$

$x = \underline{M[y]}$

$y = y + 1$

$M[y-1] = 5$

$$\begin{aligned}
[\![x = M[e];]\!]^\sharp\, A &= (A \cup \{e\}) \backslash \mathit{Expr}_x \\
[\![M[e_1] = e_2;]\!]^\sharp\, A &= A \cup \{e_1, e_2\}
\end{aligned}$$

By that, every path can be analyzed :-)

A given program may admit several paths :-(

For any given input, another path may be chosen :-((

$$\begin{aligned}
[\![x = M[e];]\!]^\sharp\, A &= (A \cup \{e\}) \backslash \mathit{Expr}_x \\
[\![M[e_1] = e_2;]\!]^\sharp\, A &= A \cup \{e_1, e_2\}
\end{aligned}$$

By that, every path can be analyzed :-)

A given program may admit several paths :-(

For any given input, another path may be chosen :-((

$\Longrightarrow$ We require the set:

$$\mathcal{A}[v] = \bigcap \{ [\![\pi]\!]^\sharp \emptyset \mid \pi : \mathit{start} \to^* v \}$$

Concretely:

- $\rightarrow$ We consider all paths π which reach v.
- $\rightarrow$ For every path π, we determine the set of expressions which are available along π.
- $\rightarrow$ Initially at program start, nothing is available :-)
- $\rightarrow$ We compute the intersection $\Longrightarrow$ safe information

$$[\![x = M[e];]\!]^\sharp\, A \quad = \quad (A \cup \{e\}) \backslash Expr_x$$
$$[\![M[e_1] = e_2;]\!]^\sharp\, A \quad = \quad A \cup \{e_1, e_2\}$$

By that, every path can be analyzed :-)

A given program may admit several paths :-(

For any given input, another path may be chosen :-((

$\Longrightarrow$ We require the set:

$$\mathcal{A}[v] \quad = \quad \bigcap \{[\![\pi]\!]^\sharp \emptyset \mid \pi : start \rightarrow^* v\}$$

Concretely:

$\rightarrow$ We consider all paths π which reach v.

$\rightarrow$ For every path π, we determine the set of expressions which are available along π.

$\rightarrow$ Initially at program start, nothing is available :-)

$\rightarrow$ We compute the intersection $\Longrightarrow$ safe information

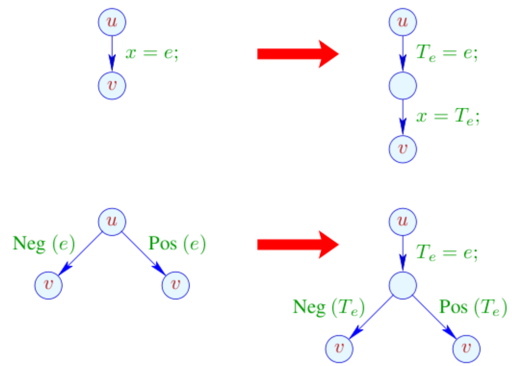
How do we exploit this information ???

Transformation 1.1:

We provide novel registers T_e as storage for the e:

Transformation 1.1:

We provide novel registers T_e as storage for the e:



... analogously for $R = M[e];$ and $M[e_1] = e_2;$.

Transformation 1.2:

If e is available at program point u, then e need not be re-evaluated:

We replace the assignment with Nop :-)

... analogously for $R = M[e];$ and $M[e_1] = e_2;$.

Transformation 1.2:

If e is available at program point u, then e need not be re-evaluated:

We replace the assignment with Nop :-)

47