Script **generated by TTT**

Title: Seidl: Programmoptimierung (15.10.2015)

Date: Thu Oct 15 08:33:21 CEST 2015

Duration: 90:10 min

Pages: 44

Remark

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;

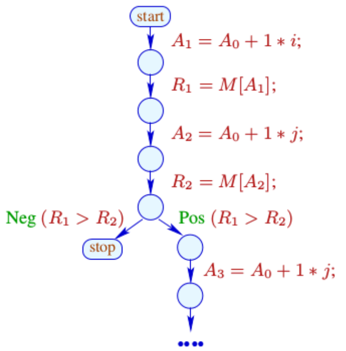
$\rightarrow$ a method which identifies at least some repeated computations ...

Background 1: An Operational Semantics

we choose a small-step operational approach.

Programs are represented as control-flow graphs.

In the example:



Thereby, represent:

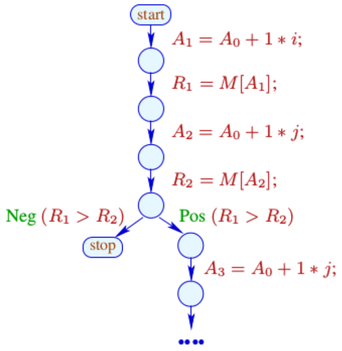
vertex	program point
start	programm start
stop	program exit
edge	step of computation

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

Edge Labelings:

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

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 :	;

Computations follow paths.

Computations transform the current state

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

where:

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

Every edge $k = (u, lab, v)$ defines a partial transformation

$$[\![k]\!] = [\![lab]\!]$$

of the state:

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 :	$;$

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 $k = (u, lab, v)$ defines a partial transformation

$$[\![k]\!] = [\![lab]\!]$$

of the state:

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 $k = (u, lab, v)$ defines a partial transformation

$$[\![k]\!] = [\![lab]\!]$$

of the state:

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

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 $k = (u, lab, v)$ defines a partial transformation

$$[\![k]\!] = [\![lab]\!]$$

of the state:

$$[\![;]\!]\,(\rho,\mu) = (\rho,\mu)$$

$$[\![\mathrm{Pos}\,(e)]\!]\,(\rho,\mu) = (\rho,\mu) \qquad \text{if } [\![e]\!]\,\rho \neq 0$$

$$[\![\mathrm{Neg}\,(e)]\!]\,(\rho,\mu) = (\rho,\mu) \qquad \text{if } [\![e]\!]\,\rho = 0$$

$$[\![;]\!]\,(\rho,\mu) = (\rho,\mu)$$

$$[\![\mathrm{Pos}\,(e)]\!]\,(\rho,\mu) = (\rho,\mu) \qquad \text{if } [\![e]\!]\,\rho \neq 0$$

$$[\![\mathrm{Neg}\,(e)]\!]\,(\rho,\mu) = (\rho,\mu) \qquad \text{if } [\![e]\!]\,\rho = 0$$

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

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

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

$$[\![;]\!]\,(\rho,\mu) = (\rho,\mu)$$

$$[\![\mathrm{Pos}\,(e)]\!]\,(\rho,\mu) = (\rho,\mu) \qquad \text{if } [\![e]\!]\,\rho \neq 0$$

$$[\![\mathrm{Neg}\,(e)]\!]\,(\rho,\mu) = (\rho,\mu) \qquad \text{if } [\![e]\!]\,\rho = 0$$

// $[\![e]\!]$: 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

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

$$\llbracket M[e_1] = e_2; \rrbracket\,(\rho, \mu) \;=\; (\rho, \boxed{\mu \oplus \{\llbracket e_1 \rrbracket\,\rho \mapsto \llbracket e_2 \rrbracket\,\rho\}})$$

Example

$\llbracket x = x + 1; \rrbracket\,(\{x \mapsto 5\}, \mu) = (\rho, \mu)$ where:

$$\begin{aligned} \rho &= \{x \mapsto 5\} \oplus \{x \mapsto \llbracket x+1 \rrbracket\,\{x \mapsto 5\}\} \\ &= \{x \mapsto 5\} \oplus \{x \mapsto 6\} \\ &= \{x \mapsto 6\} \end{aligned}$$

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

$$\llbracket M[e_1] = e_2; \rrbracket\,(\rho, \mu) \;=\; (\rho, \boxed{\mu \oplus \{\llbracket e_1 \rrbracket\,\rho \mapsto \llbracket e_2 \rrbracket\,\rho\}})$$

Example

$\llbracket x = x + 1; \rrbracket\,(\{x \mapsto 5\}, \mu) = (\rho, \mu)$ where:

$$\begin{aligned} \rho &= \{x \mapsto 5\} \oplus \{x \mapsto \llbracket x+1 \rrbracket\,\{x \mapsto 5\}\} \\ &= \{x \mapsto 5\} \oplus \{x \mapsto 6\} \\ &= \{x \mapsto 6\} \end{aligned}$$

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

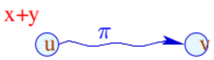
$$s \in \mathit{def}\,(\llbracket k_m \rrbracket \circ \ldots \circ \llbracket k_1 \rrbracket)$$

The result of the computation is:

$$\llbracket \pi \rrbracket\, s = (\llbracket k_m \rrbracket \circ \ldots \circ \llbracket k_1 \rrbracket)\, s$$

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.

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:

$$\begin{array}{lllll} [\![;]\!]^\sharp A & = & A & & \\ [\![Pos(e)]\!]^\sharp A & = & [\![Neg(e)]\!]^\sharp A & = & A \cup \{e\} \\ [\![x = e;]\!]^\sharp A & = & (A \cup \{e\}) \backslash Expr_x & & \text{where} \end{array}$$

$Expr_x$ all expressions which contain x

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

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

By that, every path can be analyzed.

A given program may admit several paths.

For any given input, another path may be chosen ...

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

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:

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

Concretely:

→ We consider all paths π which reach v.

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

→ Initially at program start, nothing is available

→ We compute the intersection $\Longrightarrow$ safe information

Concretely:

→ We consider all paths π which reach v.

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

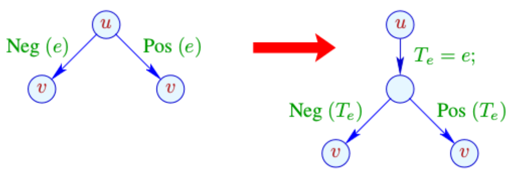
→ Initially at program start, nothing is available

→ 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:



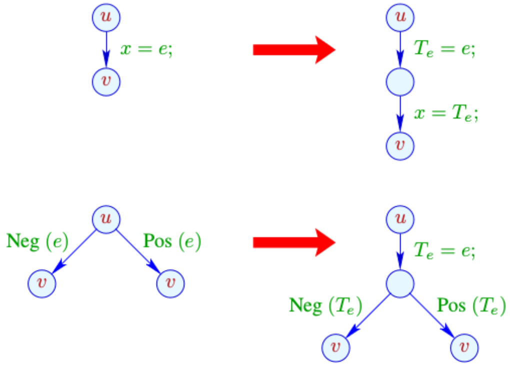
Transformation 1.1:

e

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 $\quad R = M[e]; \quad$ and $\quad M[e_1] = e_2;$.

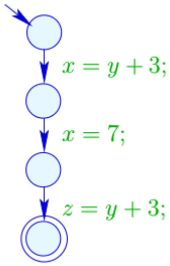
Transformation 1.2:

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

We replace the assignment with Nop :-)

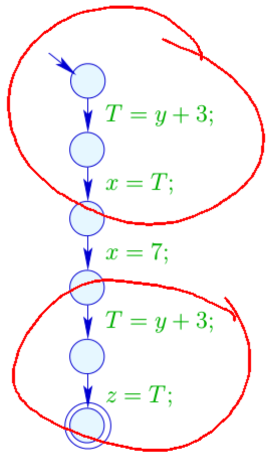
Example:

$$\begin{array}{lcl} x & = & y+3; \\ x & = & 7; \\ z & = & y+3; \end{array}$$



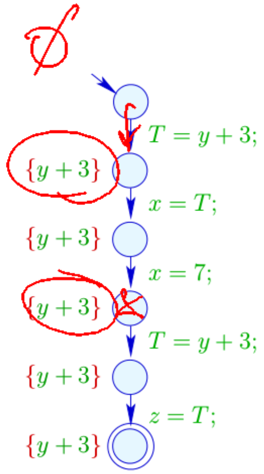
Example:

$$\begin{array}{lcl} x & = & y+3; \\ x & = & 7; \\ z & = & y+3; \end{array}$$



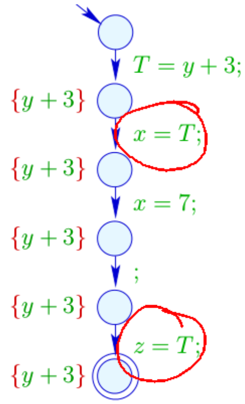
Example:

$$
\begin{aligned}
x &= y + 3;\\
x &= 7;\\
z &= y + 3;
\end{aligned}
$$



Example:

$$
\begin{aligned}
x &= y + 3;\\
x &= 7;\\
z &= y + 3;
\end{aligned}
$$



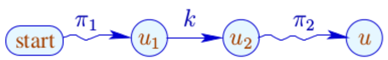
Correctness: (Idea)

Transformation 1.1 preserves the semantics and $\mathcal{A}[u]$ for all program points u :-)

Assume $\pi : start \to^* u$ is the path taken by a computation.

If $e \in \mathcal{A}[u]$, then also $e \in [\![\pi]\!]^\sharp \, \emptyset$.

Therefore, π can be decomposed into:



with the following properties:

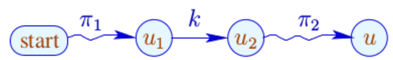
Correctness: (Idea)

Transformation 1.1 preserves the semantics and $\mathcal{A}[u]$ for all program points u :-)

Assume $\pi : start \to^* u$ is the path taken by a computation.

If $e \in \mathcal{A}[u]$, then also $e \in [\![\pi]\!]^\sharp \, \emptyset$.

Therefore, π can be decomposed into:



with the following properties:

- The expression e is evaluated at the edge k;
- The expression e is not removed from the set of available expressions at any edge in π_2, i.e., no variable of e receives a new value :-)

- The expression e is evaluated at the edge k;
- The expression e is not removed from the set of available expressions at any edge in π_2, i.e., no variable of e receives a new value :-)

Warning:

Transformation 1.1 is only meaningful for assignments $x = e$; where:

$\rightarrow$ $e \notin Vars$;

$\rightarrow$ the evaluation of e is non-trivial :-}

- The expression e is evaluated at the edge k;
- The expression e is not removed from the set of available expressions at any edge in π_2, i.e., no variable of e receives a new value :-)

$$\Longrightarrow$$

The register T_e contains the value of e whenever u is reached :-))

Warning:

Transformation 1.1 is only meaningful for assignments $x = e;$ where:

- → $e \notin Vars;$
- → the evaluation of e is non-trivial :-}