Script **generated by TTT**

Title: Seidl: Programmoptimierung (24.10.2012)

Date: Wed Oct 24 08:33:36 CEST 2012

Duration: 84:52 min

Pages: 58

Helmut Seidl

Program Optimization

TU München

Winter 2012/13

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

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

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:

$$\overset{x+y}{u} \xrightarrow{\pi} v$$

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

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

Example:

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

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

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

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

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

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

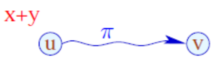
$$
s \in \mathit{def}\,([\![k_m]\!] \circ \ldots \circ [\![k_1]\!])
$$

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 $x + y$ ...

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

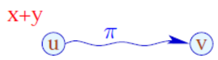
$$
s \in \mathit{def}\,([\![k_m]\!] \circ \ldots \circ [\![k_1]\!])
$$

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 $x + y$ ...

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

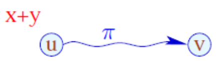
$$s \in def\ ([\![k_m]\!] \circ \dots \circ [\![k_1]\!])$$

The result of the computation is:

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

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 $A = \{e_1, \dots, e_r\}$ 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 $A = \{e_1, \dots, e_r\}$ 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 $\pi = k_1 \dots k_r$:

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

$\sharp$ $\#$

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

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

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 $\pi = k_1 \dots k_r$:

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

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

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

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

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

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

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

$$[\![x = x+1;]\!]^\# A = A \setminus Expr_x$$

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

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

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:

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

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

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 ???

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

By that, every path can be analyzed :-)

A given program may admit several paths :-(

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

x = y + z;

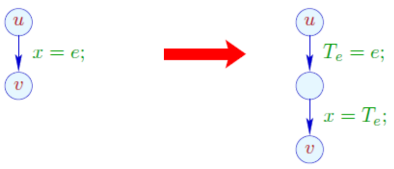
x = 1;

$\Longrightarrow$ We require the set:

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

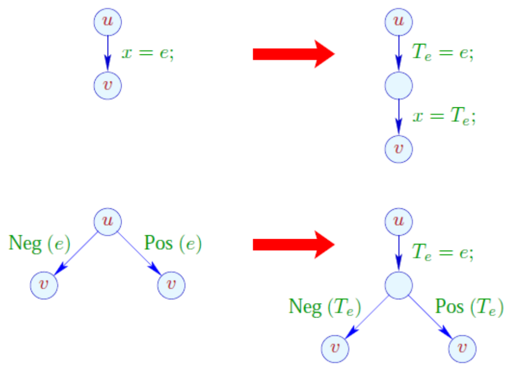
Transformation 1.1:

We provide novel registers T_e as storage for the e:



Transformation 1.1:

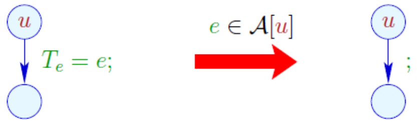
We provide novel registers T_e as storage for the e:



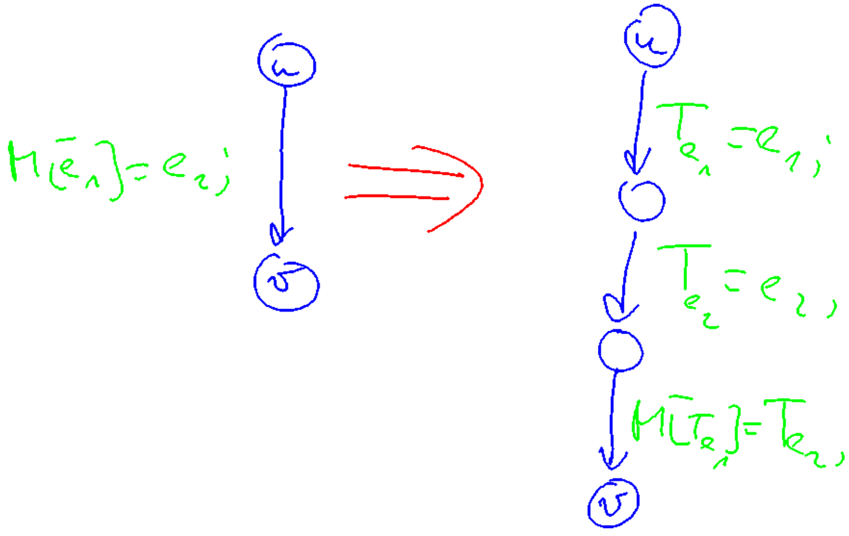
... analogously for $R = M[e];$ and $M[e_1] = e_2;$.

Transformation 1.2:

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



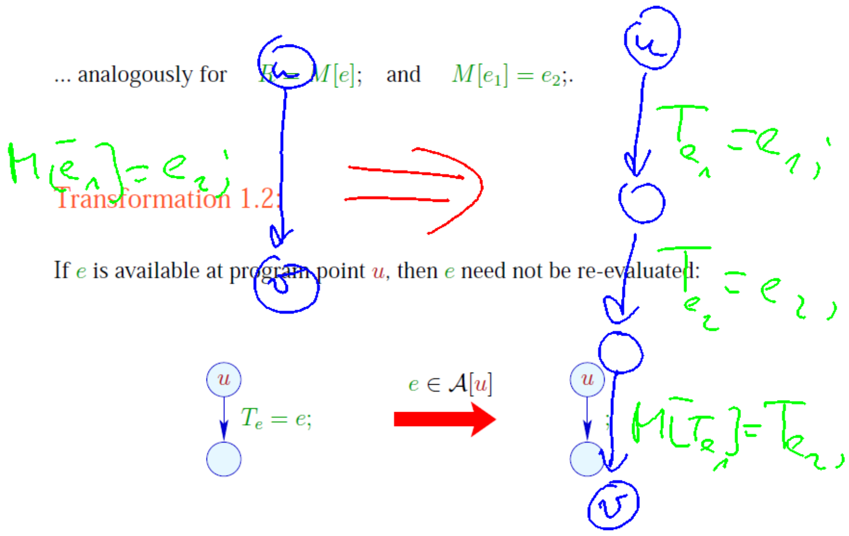
We replace the assignment with Nop :-)



... 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:

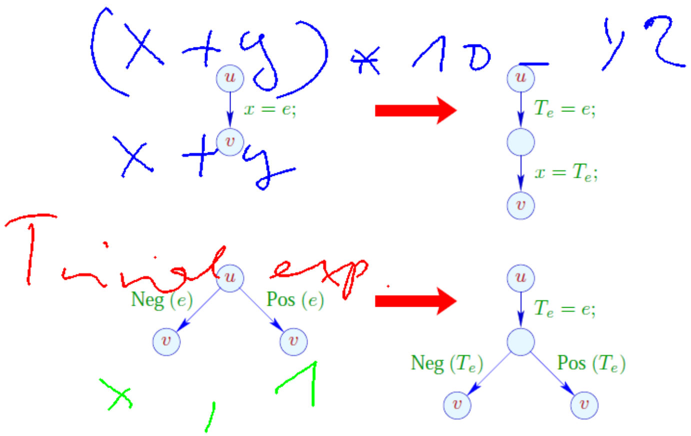


We replace the assignment with Nop :-)

47

Transformation 1.1:

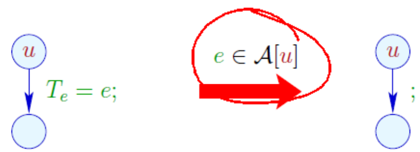
We provide novel registers T_e as storage for the e:



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

Transformation 1.2:

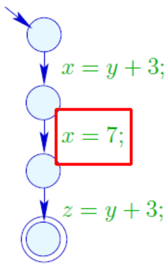
If e is available at program point u, then e need not be re-evaluated:



We replace the assignment with Nop :-)

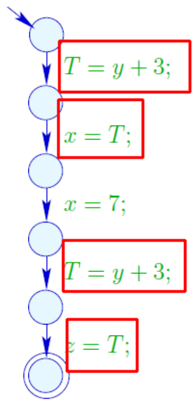
Example:

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



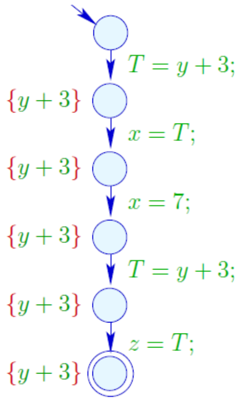
Example:

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



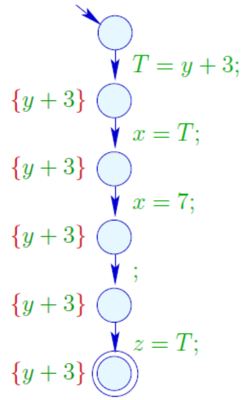
Example:

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



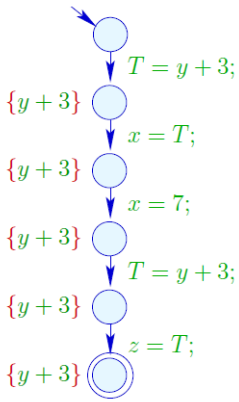
Example:

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



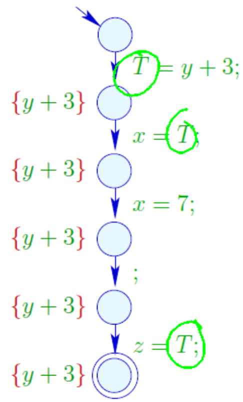
Example:

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



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

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

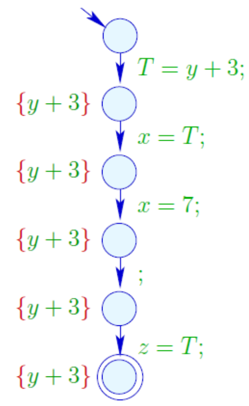
Therefore, π can be decomposed into:

$$\text{start} \xrightarrow{\pi_1} u_1 \xrightarrow{k} u_2 \xrightarrow{\pi_2} u$$

with the following properties:

Example:

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



Correctness: (Idea)

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

Assume $\pi : start \rightarrow^* u$ 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:

$$\text{start} \xrightarrow{\pi_1} u_1 \xrightarrow{k} u_2 \xrightarrow{\pi_2} u$$

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 :-)

Correctness: (Idea)

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

Assume $\pi : \mathit{start} \to^* u$ 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:

$$\text{start} \xrightarrow{\pi_1} u_1 \xrightarrow{k} u_2 \xrightarrow{\pi_2} u$$

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 :-)

Warning:

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

$\rightarrow$ $e \notin \mathit{Vars}$;

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

Warning:

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

$\rightarrow$ $x \notin \mathit{Vars}(e)$;

$\rightarrow$ $e \notin \mathit{Vars}$;

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

Which leaves us with the following question ...

Question:

How can we compute $\mathcal{A}[u]$ for every program point u ??

We collect all restrictions to the values of $\mathcal{A}[u]$ into a system of constraints:

$$\begin{array}{lll} \mathcal{A}[start] & \subseteq & \emptyset \\ \mathcal{A}[v] & \subseteq & [\![k]\!]^\sharp\,(\mathcal{A}[u]) \qquad k = (u, \_, v) \quad \text{edge} \end{array}$$

Question:

How can we compute $\mathcal{A}[u]$ for every program point u ??

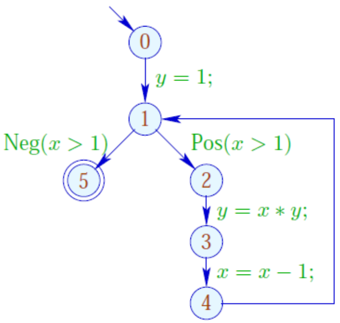
We collect all restrictions to the values of $\mathcal{A}[u]$ into a system of constraints:

$$\begin{array}{lll} \mathcal{A}[start] & \subseteq & \emptyset \\ \mathcal{A}[v] & \subseteq & [\![k]\!]^\sharp\,(\mathcal{A}[u]) \qquad k = (u, \_, v) \quad \text{edge} \end{array}$$

Wanted:

- a maximally large solution (??)
- an algorithm which computes this :-)

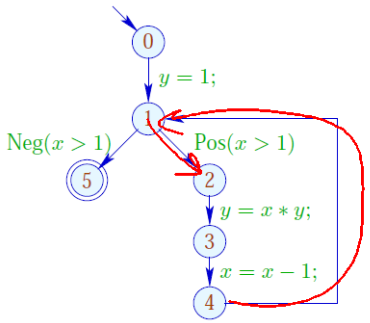
Example:



Wanted:

- a maximally large solution (??)
- an algorithm which computes this :-)

Example:

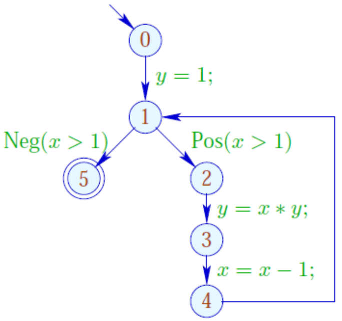


$$\mathcal{A}[0] \subseteq \emptyset$$

Wanted:

- a maximally large solution (??)
- an algorithm which computes this :-)

Example:

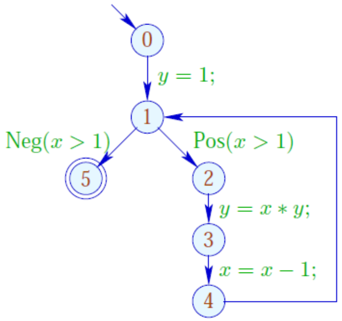


$$\begin{aligned}
\mathcal{A}[0] &\subseteq \emptyset \\
\mathcal{A}[1] &\subseteq (\mathcal{A}[0] \cup \{1\}) \backslash Expr_y \\
\mathcal{A}[1] &\subseteq \mathcal{A}[4] \\
\mathcal{A}[2] &\subseteq \mathcal{A}[1] \cup \{x > 1\} \\
\mathcal{A}[3] &\subseteq (\mathcal{A}[2] \cup \{x * y\}) \backslash Expr_y
\end{aligned}$$

Wanted:

- a maximally large solution (??)
- an algorithm which computes this :-)

Example:

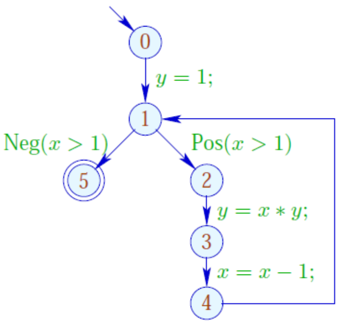


$$\begin{aligned}
\mathcal{A}[0] &\subseteq \emptyset \\
\mathcal{A}[1] &\subseteq (\mathcal{A}[0] \cup \{1\}) \backslash Expr_y \\
\mathcal{A}[1] &\subseteq \mathcal{A}[4] \\
\mathcal{A}[2] &\subseteq \mathcal{A}[1] \cup \{x > 1\}
\end{aligned}$$

Wanted:

- a maximally large solution (??)
- an algorithm which computes this :-)

Example:



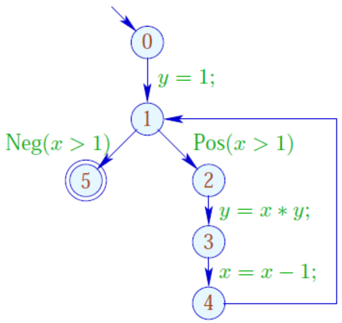
$$\begin{aligned}
\mathcal{A}[0] &\subseteq \emptyset \\
\mathcal{A}[1] &\subseteq (\mathcal{A}[0] \cup \{1\}) \backslash Expr_y \\
\mathcal{A}[1] &\subseteq \mathcal{A}[4] \\
\mathcal{A}[2] &\subseteq \mathcal{A}[1] \cup \{x > 1\} \\
\mathcal{A}[3] &\subseteq (\mathcal{A}[2] \cup \{x * y\}) \backslash Expr_y
\end{aligned}$$

63

Wanted:

- a maximally large solution (??)
- an algorithm which computes this :-)

Example:

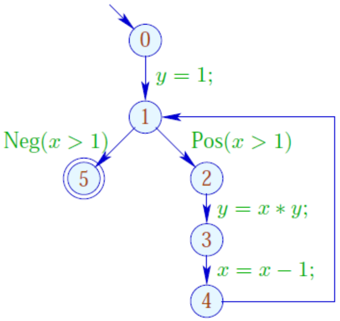


$$\begin{aligned}
\mathcal{A}[0] &\subseteq \emptyset \\
\mathcal{A}[1] &\subseteq (\mathcal{A}[0] \cup \{1\}) \backslash Expr_y \\
\mathcal{A}[1] &\subseteq \mathcal{A}[4] \\
\mathcal{A}[2] &\subseteq \mathcal{A}[1] \cup \{x > 1\} \\
\mathcal{A}[3] &\subseteq (\mathcal{A}[2] \cup \{x * y\}) \backslash Expr_y \\
\mathcal{A}[4] &\subseteq (\mathcal{A}[3] \cup \{x - 1\}) \backslash Expr_x \\
\mathcal{A}[5] &\subseteq \mathcal{A}[1] \cup \{x > 1\}
\end{aligned}$$

Wanted:

- a maximally large solution (??)
- an algorithm which computes this :-)

Example:



Solution:

$$
\begin{array}{lcl}
\mathcal{A}[0] & = & \emptyset \\
\mathcal{A}[1] & = & \{1\} \\
\mathcal{A}[2] & = & \{1, x > 1\} \\
\mathcal{A}[3] & = & \{1, x > 1\} \\
\mathcal{A}[4] & = & \{1\} \\
\mathcal{A}[5] & = & \{1, x > 1\}
\end{array}
$$

Observation:

- The possible values for $\mathcal{A}[u]$ form a complete lattice:

$$\mathbb{D} = 2^{Expr} \quad \text{with} \quad B_1 \sqsubseteq B_2 \quad \text{iff} \quad B_1 \supseteq B_2$$

Observation:

- The possible values for $\mathcal{A}[u]$ form a complete lattice:

$$\mathbb{D} = 2^{Expr} \quad \text{with} \quad B_1 \sqsubseteq B_2 \quad \text{iff} \quad B_1 \supseteq B_2$$

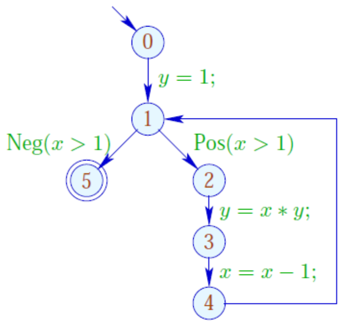
- The functions $[\![k]\!]^\sharp : \mathbb{D} \to \mathbb{D}$ are monotonic, i.e.,

$$[\![k]\!]^\sharp(B_1) \sqsubseteq [\![k]\!]^\sharp(B_2) \quad \text{whenever} \quad B_1 \sqsubseteq B_2$$

Wanted:

- a maximally large solution (??)
- an algorithm which computes this :-)

Example:



$$
\begin{array}{lcl}
\mathcal{A}[0] & \subseteq & \emptyset \\
\mathcal{A}[1] & \subseteq & (\mathcal{A}[0] \cup \{1\}) \backslash Expr_y \\
\mathcal{A}[1] & \subseteq & \mathcal{A}[4] \\
\mathcal{A}[2] & \subseteq & \mathcal{A}[1] \cup \{x > 1\} \\
\mathcal{A}[3] & \subseteq & (\mathcal{A}[2] \cup \{x * y\}) \backslash Expr_y \\
\mathcal{A}[4] & \subseteq & (\mathcal{A}[3] \cup \{x - 1\}) \backslash Expr_x \\
\mathcal{A}[5] & \subseteq & \mathcal{A}[1] \cup \{x > 1\}
\end{array}
$$

Observation:

- The possible values for $\mathcal{A}[u]$ form a complete lattice:

$$\mathbb{D} = 2^{Expr} \quad \text{with} \quad B_1 \sqsubseteq B_2 \quad \text{iff} \quad B_1 \supseteq B_2$$

- The functions $[\![k]\!]^\sharp : \mathbb{D} \to \mathbb{D}$ are monotonic, i.e.,

$$[\![k]\!]^\sharp(B_1) \sqsubseteq [\![k]\!]^\sharp(B_2) \quad \text{whenever} \quad B_1 \sqsubseteq B_2$$

Observation:

- The possible values for $\mathcal{A}[u]$ form a complete lattice:

$$\mathbb{D} = 2^{Expr} \quad \text{with} \quad B_1 \sqsubseteq B_2 \quad \text{iff} \quad B_1 \supseteq B_2$$

- The functions $[\![k]\!]^\sharp : \mathbb{D} \to \mathbb{D}$ are monotonic, i.e.,

$$[\![k]\!]^\sharp(B_1) \sqsubseteq [\![k]\!]^\sharp(B_2) \quad \text{whenever} \quad B_1 \sqsubseteq B_2$$

68

Background 2: Complete Lattices

A set $\mathbb{D}$ together with a relation $\sqsubseteq \subseteq \mathbb{D} \times \mathbb{D}$ is a partial order if for all $a, b, c \in \mathbb{D}$,

$$\begin{array}{lll} a \sqsubseteq a & & \textit{reflexivity} \\ a \sqsubseteq b \wedge b \sqsubseteq a & \Longrightarrow \quad a = b & \textit{anti-symmetry} \\ a \sqsubseteq b \wedge b \sqsubseteq c & \Longrightarrow \quad a \sqsubseteq c & \textit{transitivity} \end{array}$$

Examples:

1. $\mathbb{D} = 2^{\{a,b,c\}}$ with the relation "$\subseteq$" :