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

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

**Example:**

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

$$\rho = \{ x \mapsto 5 \} \oplus \{ x \mapsto \{ x \mapsto 5 \} \}$$

$$= \{ x \mapsto 5 \} \oplus \{ x \mapsto 3 \}$$

$$= \{ x \mapsto 6 \}$$

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

$$s \in \text{def} (\{ k_m \} \circ \ldots \circ \{ k_1 \})$$

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 $$u$$:

$$x + y$$

We perform a computation along path $$\pi$$ and reach $$v$$ where we evaluate again $$x + y$$. ...
\[ [R = M[e]; \rho, \nu] = (\rho \cup \{ R \mapsto \mu([e]\mu) \}, \nu) \]

\[ [M[e_1] = e_2; \rho, \nu] = (\rho, \nu \cup \{ [e_1] \rho \mapsto [e_2] \rho \}) \]

**Example:**

\[ [x = x + 1; \{ x \mapsto 5 \}, \rho] = (\rho, \nu) \quad \text{where:} \]

\[ \rho = \{ x \mapsto 5 \} \cup \{ x \mapsto [x + 1] \{ x \mapsto 5 \} \} \]
\[ = \{ x \mapsto 5 \} \cup \{ x \mapsto 6 \} \]
\[ = \{ x \mapsto 6 \} \]

\[ [x = x + 1; \{ x \mapsto 5 \}, \rho] = (\rho, \nu) \quad \text{where:} \]

\[ \rho = \{ x \mapsto 5 \} \cup \{ x \mapsto [e] \rho \} \]
\[ = \{ x \mapsto 5 \} \cup \{ x \mapsto 6 \} \]
\[ = \{ x \mapsto 6 \} \]

**Application:**

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

\[ x + y \]

\[ u \xrightarrow{\pi} v \]

We perform a computation along path \( \pi \) and reach \( v \) where we evaluate again \( x + y \)...

**Example:**

\[ [\{ e \} \{ p, \mu \}] = (p, \mu) \]

\[ [P(e)] (p, \mu) = (p, \mu) \quad \text{if} \ [e] \rho \neq 0 \]

\[ [E(e)] (p, \mu) = (p, \mu) \quad \text{if} \ [e] \rho = 0 \]

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

// \{ x + y \} \{ x \mapsto 7, y \mapsto -1 \} = 6

// \{ (x = 4) \} \{ x \mapsto 5 \} = 1

\[ [R = e; \rho, \nu] = (\rho \cup \{ R \mapsto \mu([e]\mu) \}, \nu) \]

// where "∪" modifies a mapping at a given argument

**Application:**

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

\[ x + y \]

\[ u \xrightarrow{\pi} v \]

We perform a computation along path \( \pi \) and reach \( v \) where we evaluate again \( x + y \)...

**Application:**

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

\[ x + y \]

\[ u \xrightarrow{\pi} v \]

We perform a computation along path \( \pi \) and reach \( v \) where we evaluate again \( x + y \)...

**Application:**

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

\[ x + y \]

\[ u \xrightarrow{\pi} v \]

We perform a computation along path \( \pi \) and reach \( v \) where we evaluate again \( x + y \)...

**Application:**

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

\[ x + y \]

\[ u \xrightarrow{\pi} v \]

We perform a computation along path \( \pi \) and reach \( v \) where we evaluate again \( x + y \)...

**Application:**

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

\[ x + y \]

\[ u \xrightarrow{\pi} v \]

We perform a computation along path \( \pi \) and reach \( v \) where we evaluate again \( x + y \)...
A path $\pi = k_1k_2 \ldots k_m$ is a computation for the state $s$ if:

$s \in \text{def} \left( [k_m] \circ \ldots \circ [k_1] \right)$

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 $u$:

$\begin{array}{c}
\text{x+y} \\
\text{u} \xrightarrow{\pi} \text{v}
\end{array}$

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

---

**Idea:**

If $x$ and $y$ have not been modified in $\pi$, 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 $\pi$ :-)

**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 $[k]^A$ of expressions whose values are available after execution of $k$ ...

---

**Idea:**

If $x$ and $y$ have not been modified in $\pi$, 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 $\pi$ :-)

**More generally:**

Assume that the values of the expressions $A = \{e_1, \ldots, e_r\}$ are available at $u$. 
... which transformations can be composed to the effect of a path 
\(\pi = k_1 \ldots k_\omega: \)
\([\pi]^I = [k_\omega]^I \circ \ldots \circ [k_1]^I\)**

**The effect \([k]^I\) of an edge \(k = (u, \text{lab}, v)\) only depends on the label \(\text{lab}\), i.e., \([k]^I = [\text{lab}]^I\)**

\[\begin{align*}
[\text{\text{[]}}]^I A &= A \\
[\text{\text{Pos}(e)}]^I A &= [\text{\text{Neg}(e)}]^I A = A \cup \{e\} \\
[x = e]^I A &= (A \cup \{e\}) \setminus \text{Expr}_x \quad \text{where} \\
&\quad \text{Expr}_x \text{ all expressions which contain } x
\end{align*}\]
\[ [x = M[e];]^{A} A \quad = \quad (A \cup \{ e \}) \setminus \text{Expr}_x \]
\[ [M[e_1] = e_2]^{A} A \quad = \quad A \cup \{ e_1, e_2 \} \]

By that, every path can be analyzed  
A given program may admit several paths  
For any given input, another path may be chosen  

\[ [x = M[e];]^{A} A \quad = \quad (A \cup \{ e \}) \setminus \text{Expr}_x \]
\[ [M[e_1] = e_2]^{A} A \quad = \quad A \cup \{ e_1, e_2 \} \]

Concretely:

\[ \rightarrow \quad \text{We consider all paths \( \pi \) which reach \( v \).} \]
\[ \rightarrow \quad \text{For every path \( \pi \), we determine the set of expressions which are available along \( \pi \).} \]
\[ \rightarrow \quad \text{Initially at program start, nothing is available  
}
\[ \rightarrow \quad \text{We compute the intersection safe information} \]

\[ A[v] \quad = \quad \bigcap \{ \pi(v) : \pi : \text{start} \rightarrow^* v \} \]
Concretely:

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

How do we exploit this information ????

\[
[x = M[e]; A] = (A \cup \{v\}) \setminus \text{Expr}_x
\]

\[
[M[e_1] = e_2; A] = A \cup \{e_1, e_2\}
\]

By that, every path can be analyzed.

A given program may admit several paths.
For any given input, another path may be chosen.

\[\mathcal{A}[v] = \bigcap \{[\pi; A] \mid \pi : \text{start} \rightarrow^* v\}\]

Transformation 1.1:
We provide novel registers \( T_e \) as storage for the \( e \):

![Diagram](image1)

![Diagram](image2)
... 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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

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:

\[
\begin{align*}
    T_e &= e; \\
    e &\in A[u]
\end{align*}
\]

We replace the assignment with \( Nop \). :-)
Transformation 1.1:  
We provide novel registers $T_e$ as storage for the $e$:

$$ (x + y) \times 10 - y \times x $$

$$ x = e; $$

$$ T_e = e; $$

$$ x = T_e; $$

$$ T_e = e; $$

$$ T_e = e; $$

Example:

\[
x = y + 3; \\
x = y + 3; \\
\]

\[
x = y + 3; \\
x = 7; \\
\]

\[
z = y + 3; \\
z = y + 3; \\
\]

\[
x = y + 3; \\
x = 7; \\
\]

\[
z = y + 3; \\
z = y + 3; \\
\]

Example:

\[
T = y + 3; \\
x = 7; \\
\]

\[
x = T; \\
\]

\[
z = y + 3; \\
z = T; \\
\]

\[
x = T; \\
\]

\[
z = T; \\
\]

\[
x = T; \\
\]

\[
z = T; \\
\]

\[
x = T; \\
\]

\[
z = T; \\
\]

\[
x = T; \\
\]

\[
z = T; \\
\]

\[
x = T; \\
\]

\[
z = T; \\
\]

\[
x = T; \\
\]

\[
z = T; \\
\]

\[
x = T; \\
\]

\[
z = T; \\
\]

\[
x = T; \\
\]

\[
z = T; \\
\]

\[
x = T; \\
\]

\[
z = T; \\
\]

\[
x = T; \\
\]

\[
z = T; \\
\]
Example:

\[ x = y + 3; \]
\[ x = 7; \]
\[ z = y + 3; \]

\[ T = y + 3; \]
\[ x = T; \]
\[ x = 7; \]
\[ z = y + 3; \]

\[ y + 3 \]
\[ y + 3 \]
\[ y + 3 \]
\[ y + 3 \]
Correctness: (Idea)

Transformation 1.1 preserves the semantics and $A[u]$ for all program points $u$. :-)

Assume $\pi : start \rightarrow^* u$ is the path taken by a computation. If $e \in A[u]$, then also $e \in [\pi] \emptyset$.

Therefore, $\pi$ can be decomposed into:

$\begin{array}{c}
\text{start} \xrightarrow{\pi_1} u_1 \xrightarrow{k} u_2 \xrightarrow{\pi_2} u \end{array}$

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 $\pi_2$, i.e., no variable of $e$ receives a new value. :-)

Example:

$\begin{array}{c}
T = y + 3; \\
\{y + 3\} \xrightarrow{x = T;} \\
x = y + 3; \\
\{y + 3\} \xrightarrow{x = 7;} \\
z = y + 3; \\
\{y + 3\} \xrightarrow{;}
\end{array}$

$\begin{array}{c}
\{y + 3\} \xrightarrow{z = T;} \\
\{y + 3\}
\end{array}$
Correctness: (Idea)

Transformation 1.1 preserves the semantics and $A[u]$ for all program points $u$.

Assume $\pi : start \rightarrow^* u$ is the path taken by a computation.
If $e \in A[u]$, then also $e \in [\pi]^F \emptyset$.

Therefore, $\pi$ can be decomposed into:

with the following properties:

Warning:

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

$\rightarrow e \not\in Vars$;
$\rightarrow$ 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 $\pi_2$, i.e., no variable of $e$ receives a new value ::-)

Warning:

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

$\rightarrow x \not\in Vars(e)$;
$\rightarrow e \not\in Vars$;
$\rightarrow$ the evaluation of $e$ is non-trivial ::-)

Which leaves us with the following question ...
Question:
How can we compute $A[u]$ for every program point $u$ ??

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

$$A[\text{start}] \subseteq \emptyset$$
$$A[v] \subseteq [k]^2 (A[u]) \quad k = (u, \ldots, v) \text{ edge}$$

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

Example:

$$\begin{array}{c}
\text{Neg}(x > 1) \\
\text{Pos}(x > 1) \\
\text{y = x + y;} \\
\text{z = z - 1;} \\
\end{array}$$

$$\begin{array}{c}
\text{1} \\
\text{2} \\
\text{3} \\
\text{4} \\
\end{array}$$

Question:
How can we compute $A[u]$ for every program point $u$ ??

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

$$A[\text{start}] \subseteq \emptyset$$
$$A[v] \subseteq [k]^2 (A[u]) \quad k = (u, \ldots, v) \text{ edge}$$

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

Example:

$$\begin{array}{c}
\text{Neg}(x > 1) \\
\text{Pos}(x > 1) \\
\text{y = x + y;} \\
\text{z = z - 1;} \\
\end{array}$$

$$\begin{array}{c}
\text{1} \\
\text{2} \\
\text{3} \\
\text{4} \\
\end{array}$$
Wanted:
- a maximally large solution (??)
- an algorithm which computes this :-)

Example:

\[ y = 1; \]
\[ y = x \times y; \]
\[ z = x - 1; \]

\[ \mathcal{A}[0] \subseteq 0 \]
\[ \mathcal{A}[1] \subseteq (\mathcal{A}[0] \cup \{1\}) \setminus \text{Expr}_y \]
\[ \mathcal{A}[2] \subseteq \mathcal{A}[1] \cup \{x > 1\} \]
\[ \mathcal{A}[3] \subseteq (\mathcal{A}[2] \cup \{x \times y\}) \setminus \text{Expr}_y \]
Wanted:
- a maximally large solution
- an algorithm which computes this

Example:

Solution:
- $A[0] = \emptyset$
- $A[1] = \{1\}$
- $A[2] = \{1, x > 1\}$
- $A[3] = \{1, x > 1\}$
- $A[5] = \{1, x > 1\}$

Observation:
- The possible values for $A[u]$ form a complete lattice:
  $D = 2^\text{Expr}$ with $B_1 \subseteq B_2$ if $B_1 \supseteq B_2$

- The functions $[A]^F : D \to D$ are monotonic, i.e.
  $[A]^F(B_1) \subseteq [A]^F(B_2)$ whenever $B_1 \subseteq B_2$

Wanted:
- a maximally large solution
- an algorithm which computes this

Example:
Observation:

- The possible values for \(A[a]\) form a complete lattice:
  \[ D = \mathcal{B}_{\text{expr}} \] with \( B_1 \sqsubseteq B_2 \) iff \( B_1 \supseteq B_2 \)

- The functions \([k]^D : D \to D\) are monotonic, i.e.,
  \[ [k]^D(B_1) \sqsubseteq [k]^D(B_2) \] whenever \( B_1 \sqsubseteq B_2 \)

- The functions \([k]^D : D \to D\) are monotonic, i.e.,
  \[ [k]^D(B_1) \sqsubseteq [k]^D(B_2) \] whenever \( B_1 \sqsubseteq B_2 \)

---

Background 2: Complete Lattices

A set \(D\) together with a relation \(\sqsubseteq \subseteq D \times D\) is a partial order if for all \(a, b, c \in D\),

- \(a \sqsubseteq a\) \hspace{1cm} \text{reflexivity}
- \(a \sqsubseteq b \land b \sqsubseteq a \Rightarrow a = b\) \hspace{1cm} \text{anti-symmetry}
- \(a \sqsubseteq b \land b \sqsubseteq c \Rightarrow a \sqsubseteq c\) \hspace{1cm} \text{transitivity}

Examples:

1. \(D = 2^{\{a, b, c\}}\) with the relation \(\subseteq\)