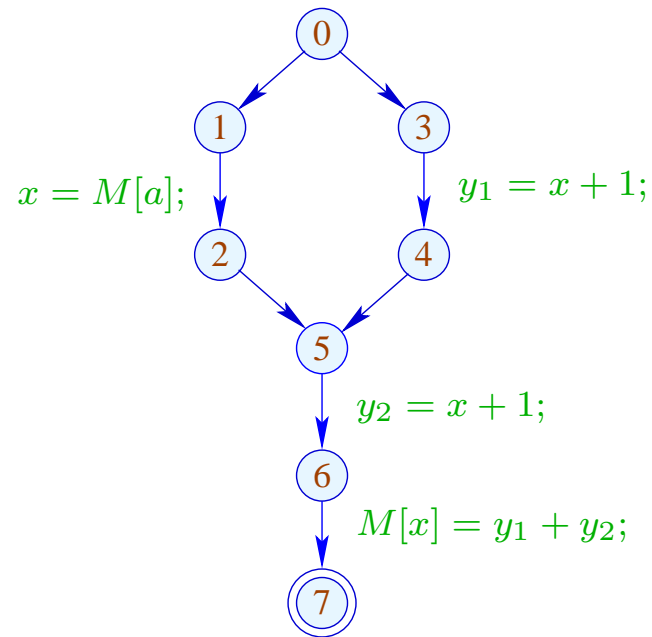


## 1.13 Eliminating Partial Redundancies

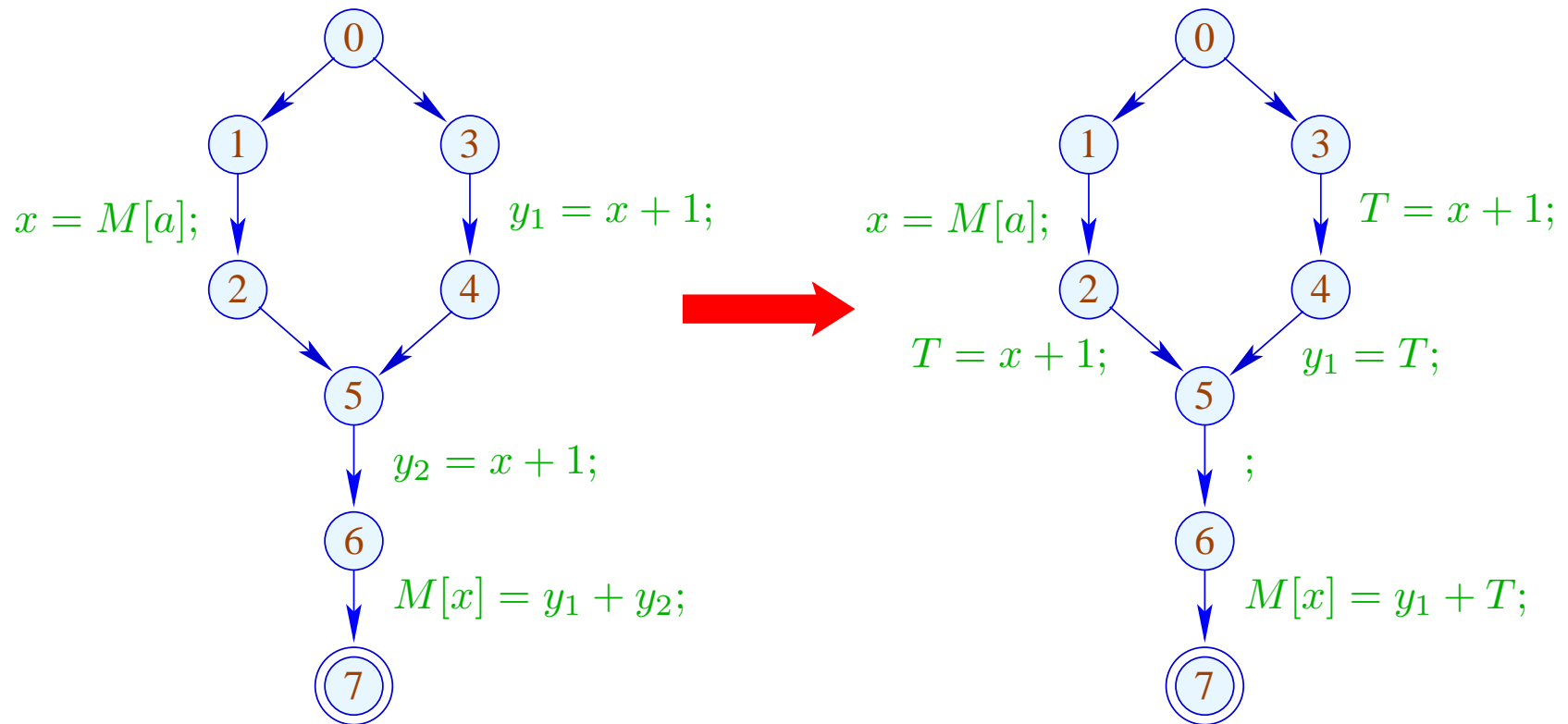
Example:



//  $x + 1$  is evaluated on every path

// on one path, however, even twice

## Removal of a partially available assignment:



## Terminological Confusion

In Section 1.4 we used, in describing the analysis:

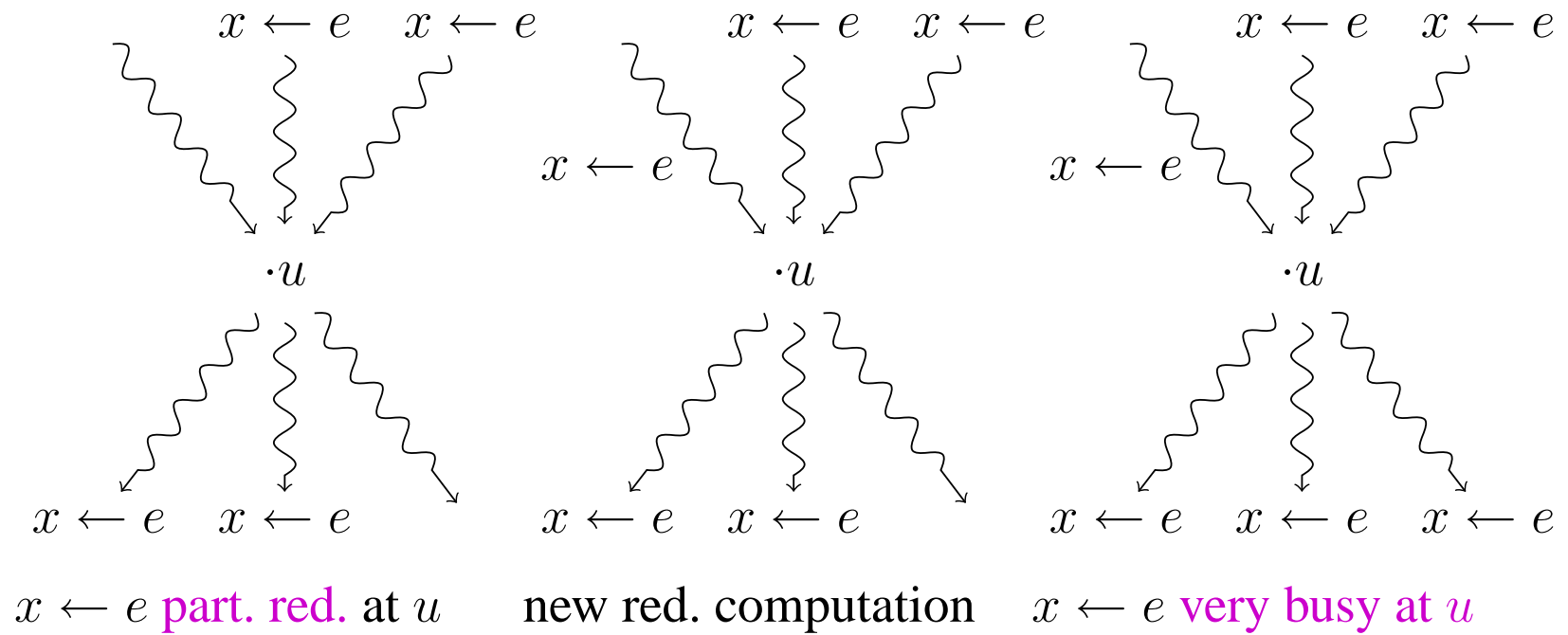
- **availability along a path**,
- **definite availability**, i.e., availability along all paths.

secretly switched to **availability** meaning **definite availability**.

The compiler literature uses the terms

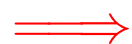
- **redundancy elimination** for the ensuing transformation,
- **partial redundancy elimination** for the actual transformation, and
- **partially redundant** for available along some paths.

## Making assignments totally available without introducing redundant computations



## Idea:

- (1) Insert assignments  $T_e = e$ ; such that  $e$  is available at all points where the value of  $e$  is required.
- (2) Thereby spare program points where  $e$  either is already **available** or will **definitely be computed** in future.  
Expressions with the latter property are called **very busy**.
- (3) Replace the original evaluations of  $e$  by accesses to the variable  $T_e$ .



we require a novel analysis

An expression  $e$  is called **busy** along a path  $\pi$ , if the expression  $e$  is evaluated before any of the variables  $x \in Vars(e)$  is overwritten.

// backward analysis!

$e$  is called **very busy** at  $u$ , if  $e$  is busy along every path  $\pi : u \rightarrow^* stop$ .

An expression  $e$  is called **busy** along a path  $\pi$ , if the expression  $e$  is evaluated before any of the variables  $x \in Vars(e)$  is overwritten.

// backward analysis!

$e$  is called **very busy** at  $u$ , if  $e$  is busy along every path  $\pi : u \rightarrow^* stop$ .

Accordingly, we require:

$$\mathcal{B}[u] = \bigcap \{ \llbracket \pi \rrbracket^\# \emptyset \mid \pi : u \rightarrow^* stop \}$$

where for  $\pi = k_1 \dots k_m$ :

$$\llbracket \pi \rrbracket^\# = \llbracket k_1 \rrbracket^\# \circ \dots \circ \llbracket k_m \rrbracket^\#$$

Our complete lattice is given by:

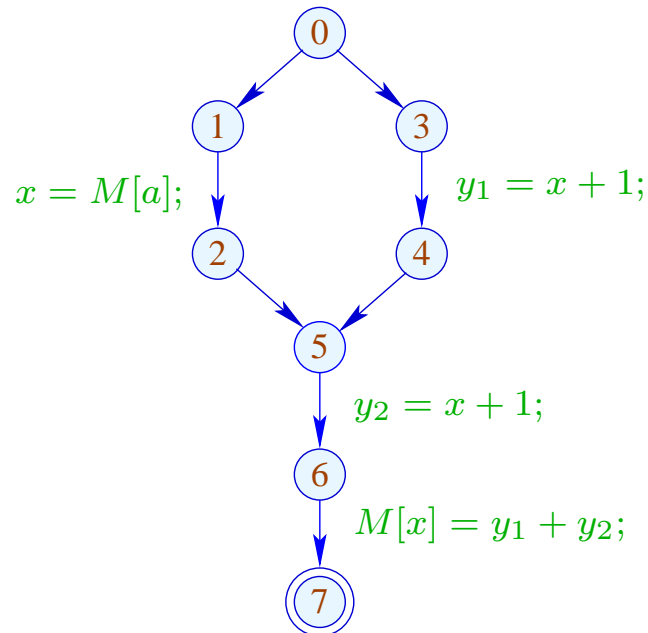
$$\mathbb{B} = 2^{\text{Expr} \setminus \text{Vars}} \quad \text{with} \quad \sqsubseteq = \supseteq$$

The effect  $\llbracket k \rrbracket^\#$  of an edge  $k = (u, \text{lab}, v)$  only depends on  $\text{lab}$ ,  
i.e.,  $\llbracket k \rrbracket^\# = \llbracket \text{lab} \rrbracket^\#$  where:

$$\begin{aligned} \llbracket ; \rrbracket^\# B &= B \\ \llbracket \text{Pos}(e) \rrbracket^\# B &= \llbracket \text{Neg}(e) \rrbracket^\# B = B \cup \{e\} \\ \llbracket x = e; \rrbracket^\# B &= (B \setminus \text{Expr}_x) \cup \{e\} \\ \llbracket x = M[e]; \rrbracket^\# B &= (B \setminus \text{Expr}_x) \cup \{e\} \\ \llbracket M[e_1] = e_2; \rrbracket^\# B &= B \cup \{e_1, e_2\} \end{aligned}$$

These effects are all **distributive**. Thus, the least solution of the constraint system yields precisely the MOP — given that *stop* is reachable from every program point

### Example:



7	$\emptyset$
6	$\{y_1 + y_2\}$
5	$\{x + 1\}$
4	$\{x + 1\}$
3	$\{x + 1\}$
2	$\{x + 1\}$
1	$\emptyset$
0	$\emptyset$

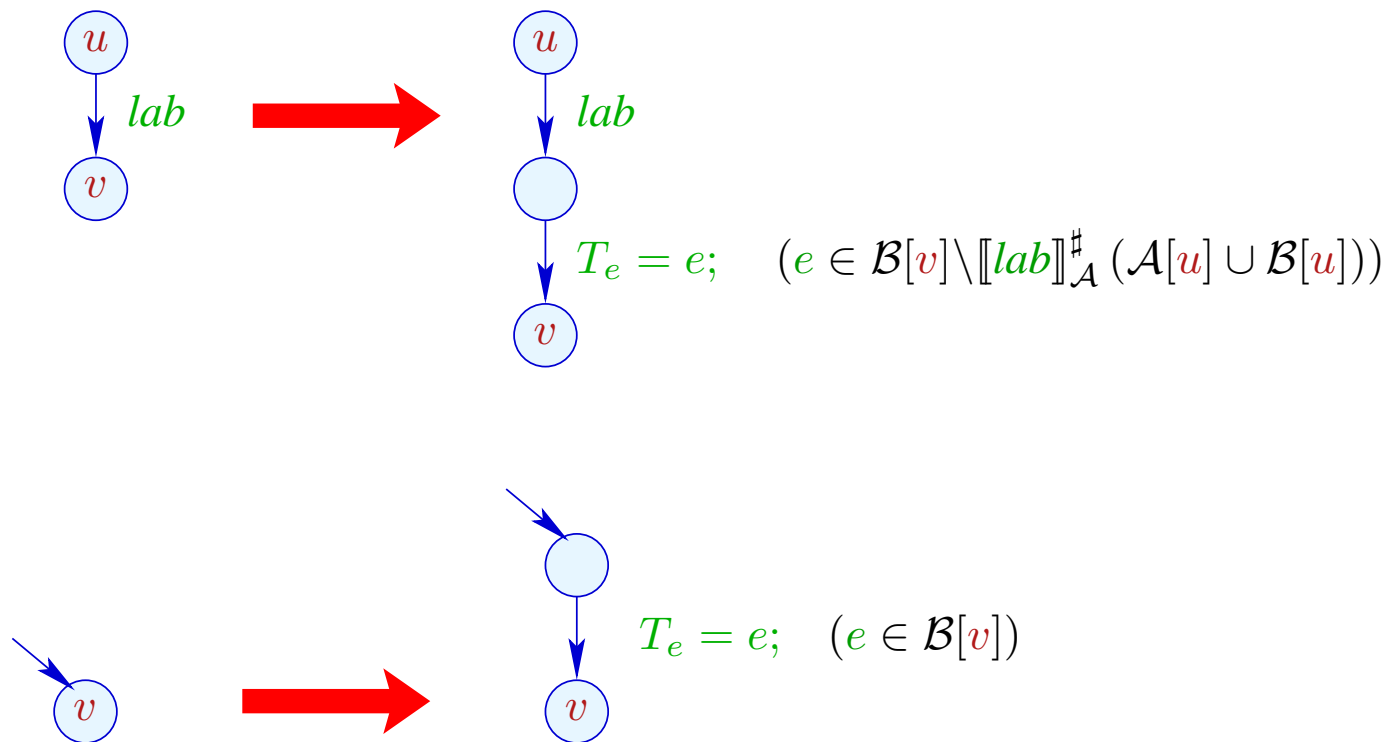
A program point  $u$  is called **safe** for  $e$ , if  $e \in \mathcal{A}[u] \cup \mathcal{B}[u]$ , i.e.,  $e$  is either available or very busy.

### Idea:

- We insert computations of  $e$  such that  $e$  becomes available at all safe program points
- We insert  $T_e = e$ ; after every edge  $(u, lab, v)$  with

$$e \in \mathcal{B}[v] \setminus \llbracket lab \rrbracket_{\mathcal{A}}^{\#}(\mathcal{A}[u] \cup \mathcal{B}[u])$$

# Transformation PRE-1:

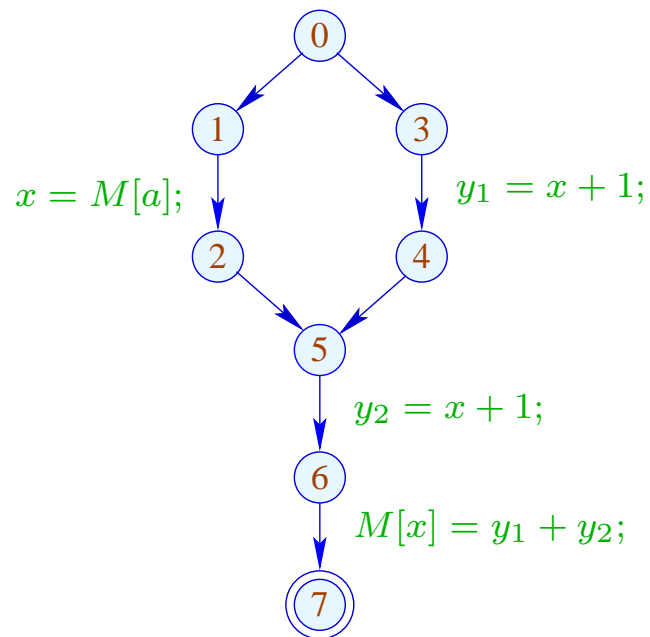


## Transformation PRE-2:



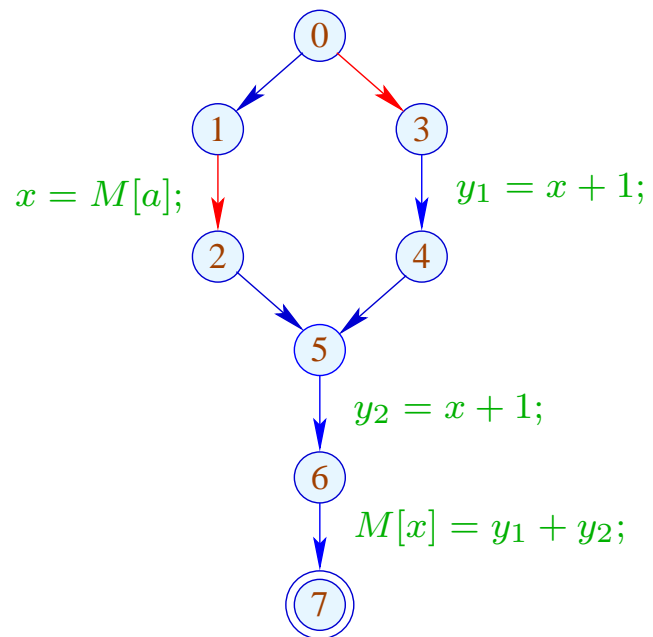
// analogously for the other uses of  $e$   
// at old edges of the program.

## In the Example:



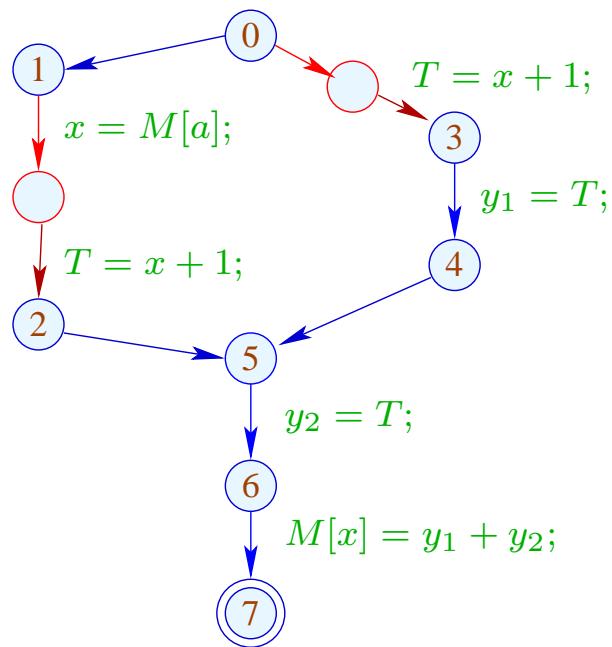
	$\mathcal{A}$	$\mathcal{B}$
0	$\emptyset$	$\emptyset$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{x + 1\}$
3	$\emptyset$	$\{x + 1\}$
4	$\{x + 1\}$	$\{x + 1\}$
5	$\emptyset$	$\{x + 1\}$
6	$\{x + 1\}$	$\{y_1 + y_2\}$
7	$\{x + 1\}$	$\emptyset$

## In the Example:



	$\mathcal{A}$	$\mathcal{B}$
0	$\emptyset$	$\emptyset$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{x + 1\}$
3	$\emptyset$	$\{x + 1\}$
4	$\{x + 1\}$	$\{x + 1\}$
5	$\emptyset$	$\{x + 1\}$
6	$\{x + 1\}$	$\{y_1 + y_2\}$
7	$\{x + 1\}$	$\emptyset$

## Im Example:



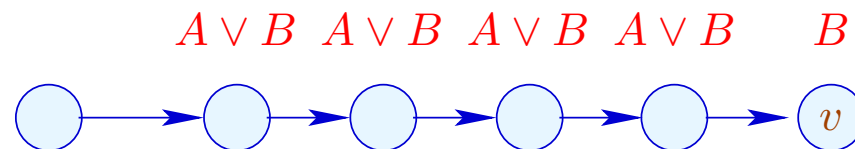
	$\mathcal{A}$	$\mathcal{B}$
0	$\emptyset$	$\emptyset$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{x + 1\}$
3	$\emptyset$	$\{x + 1\}$
4	$\{x + 1\}$	$\{x + 1\}$
5	$\emptyset$	$\{x + 1\}$
6	$\{x + 1\}$	$\{y_1 + y_2\}$
7	$\{x + 1\}$	$\emptyset$

## Correctness:

Let  $\pi$  denote a path reaching  $v$  after which a computation of an edge with  $e$  follows.

Then there is a maximal suffix of  $\pi$  such that for every edge  $k = (u, lab, u')$  in the suffix:

$$e \in \llbracket lab \rrbracket_{\mathcal{A}}^{\#}(\mathcal{A}[u] \cup \mathcal{B}[u])$$



## Correctness:

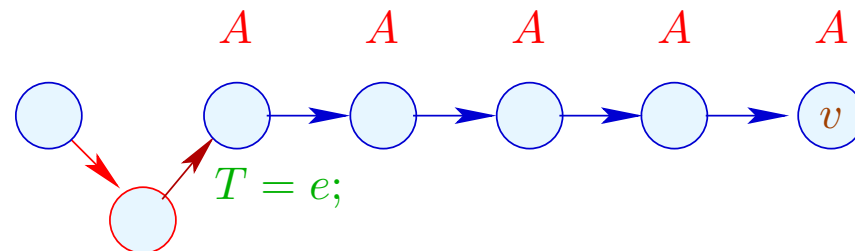
Let  $\pi$  denote a path reaching  $v$  after which a computation of an edge with  $e$  follows.

Then there is a maximal suffix of  $\pi$  such that for every edge  $k = (u, lab, u')$  in the suffix:

$$e \in \llbracket lab \rrbracket_{\mathcal{A}}^{\#}(\mathcal{A}[u] \cup \mathcal{B}[u])$$

In particular, no variable in  $e$  receives a new value

Then  $T_e = e;$  is inserted before the suffix



We conclude:

- Whenever the value of  $e$  is required,  $e$  is available  
 $\implies$  correctness of the transformation
- Every  $T = e$ ; which is inserted into a path corresponds to an  $e$   
which is replaced with  $T$   
 $\implies$  non-degradation of the efficiency