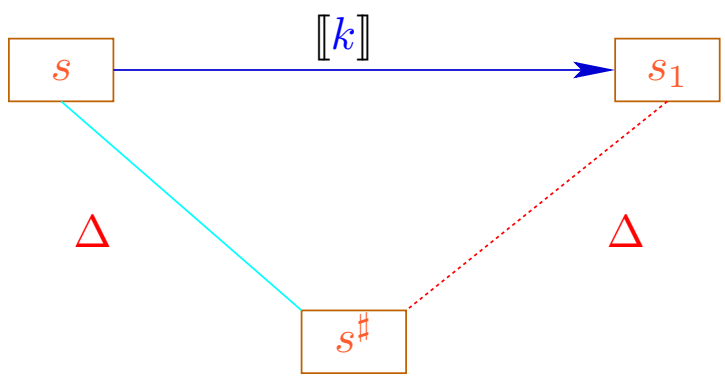
Discussion:

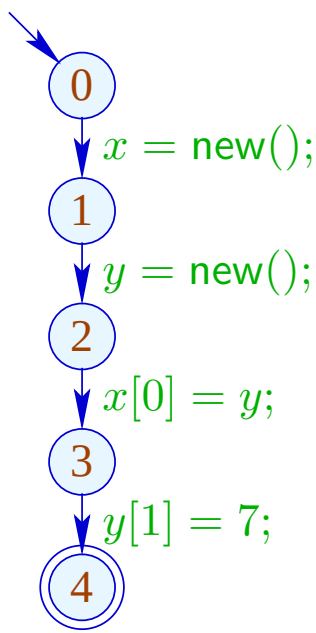
- The resulting constraint system has size $\mathcal{O}(k \cdot n)$ for k abstract addresses and n edges :-(
- The number of necessary iterations is $\mathcal{O}(k + \# Vars)$ ...
- The computed information is perhaps still too zu precise !!?
- In order to prove correctness of a solution $s^\sharp \in States^\sharp$ we show:



Alias Analysis 3. Idea:

Determine one equivalence relation $\equiv$ on variables x and memory accesses $y[\,]$ with $s_1 \equiv s_2$ whenever s_1, s_2 may contain the same address at some u_1, u_2

... in the Simple Example:



$$\equiv \;=\; \{\{x\},\ \{y, x[\,]\},\ \{y[\,]\}\}$$

Discussion:

- $\rightarrow$ We compute a single information fo the whole program.
- $\rightarrow$ The computation of this information maintains partitions $\pi = \{P_1, \ldots, P_m\}$:-)
- $\rightarrow$ Individual sets P_i are identified by means of representatives $p_i \in P_i$.
- $\rightarrow$ The operations on a partition π are:

$$
\begin{array}{lcll}
\mathsf{find}\,(\pi, p) & = & p_i & \text{if } p \in P_i \\
 & // & \text{returns the representative} & \\
\mathsf{union}\,(\pi, p_{i_1}, p_{i_2}) & = & \{P_{i_1} \cup P_{i_2}\} \cup \{P_j \mid i_1 \neq j \neq i_2\} & \\
 & // & \text{unions the represented classes} &
\end{array}
$$

→ If $x_1, x_2 \in Vars$ are equivalent, then also $x_1[\,]$ and $x_2[\,]$ must be equivalent :-)

→ If $P_i \cap Vars \neq \emptyset$, then we choose $p_i \in Vars$. Then we can apply union recursively :

$$
\begin{array}{lllll}
\mathsf{union}^*\,(\pi, q_1, q_2) & = & \mathsf{let} & p_{i_1} & = & \mathsf{find}\,(\pi, q_1) \\
 & & & p_{i_2} & = & \mathsf{find}\,(\pi, q_2) \\
 & & \mathsf{in} & \mathsf{if}\ p_{i_1} == p_{i_2}\ \mathsf{then}\ \pi & & \\
 & & & \mathsf{else}\ \ \mathsf{let}\ \pi \ = \ \mathsf{union}\,(\pi, p_{i_1}, p_{i_2}) & & \\
 & & & \qquad \mathsf{in}\ \ \mathsf{if}\ p_{i_1}, p_{i_2} \in Vars\ \mathsf{then} & & \\
 & & & \qquad\quad \mathsf{union}^*\,(\pi, p_{i_1}[\,], p_{i_2}[\,]) & &
\end{array}
$$

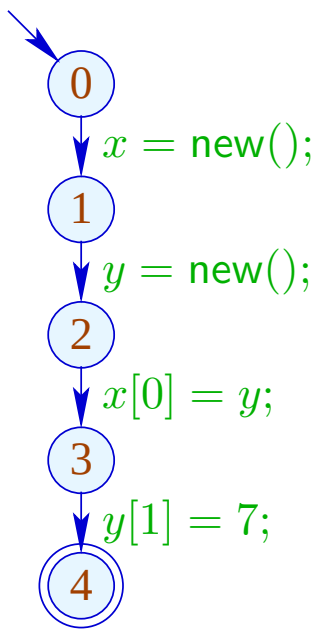
The analysis iterates over all edges once:

$$\pi = \{\{x\}, \{x[\,]\} \mid x \in \mathit{Vars}\};$$
$$\textsf{forall} \quad k = (\_, \mathit{lab}, \_) \quad \textsf{do} \quad \pi = [\![\mathit{lab}]\!]^\sharp\, \pi;$$

where:

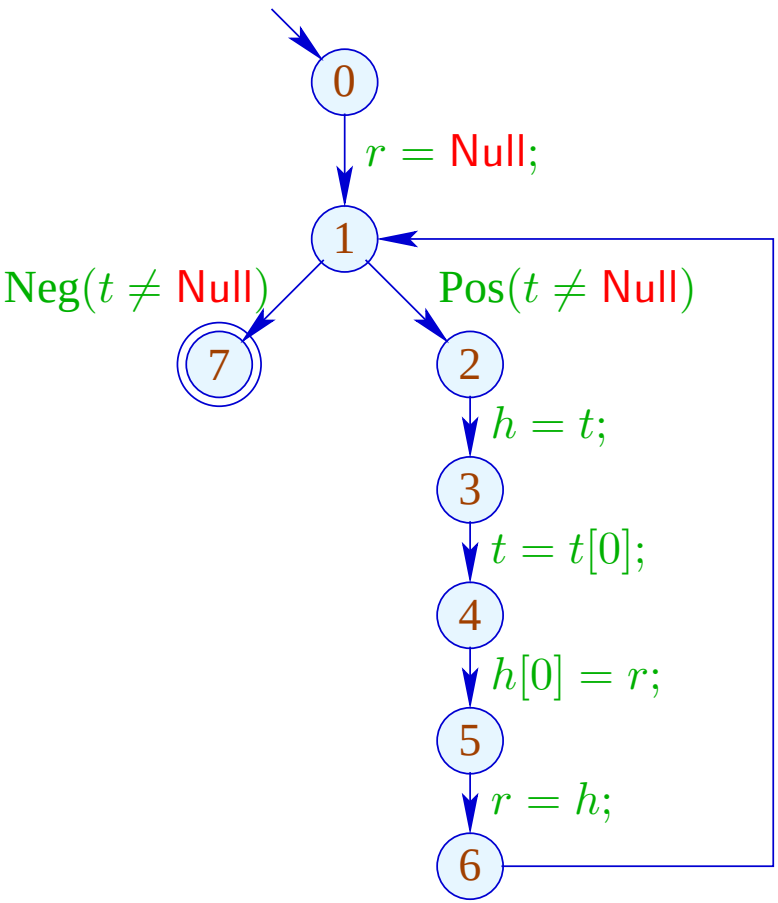
$$\begin{array}{lcll}
[\![x = y;]\!]^\sharp\, \pi & = & \textsf{union}^*\,(\pi, x, y) & \\
[\![x = y[e];]\!]^\sharp\, \pi & = & \textsf{union}^*\,(\pi, x, y[\,]) & \\
[\![y[e] = x;]\!]^\sharp\, \pi & = & \textsf{union}^*\,(\pi, x, y[\,]) & \\
[\![\mathit{lab}]\!]^\sharp\, \pi & = & \pi & \text{otherwise}
\end{array}$$

... in the Simple Example:



	$\{\{x\}, \{y\}, \{x[\,]\}, \{y[\,]\}\}$
$(0, 1)$	$\{\{x\}, \{y\}, \{x[\,]\}, \{y[\,]\}\}$
$(1, 2)$	$\{\{x\}, \{y\}, \{x[\,]\}, \{y[\,]\}\}$
$(2, 3)$	$\{\{x\}, \boxed{\{y, x[\,]\}}, \{y[\,]\}\}$
$(3, 4)$	$\{\{x\}, \{y, x[\,]\}, \{y[\,]\}\}$

... in the More Complex Example:



	$\{\{h\}, \{r\}, \{t\}, \{h[\,]\}, \{t[\,]\}\}$
$(2, 3)$	$\{\boxed{\{h, t\}}, \{r\}, \boxed{\{h[\,], t[\,]\}}\}$
$(3, 4)$	$\{\boxed{\{h, t, h[\,], t[\,]\}}, \{r\}\}$
$(4, 5)$	$\{\boxed{\{h, t, r, h[\,], t[\,]\}}\}$
$(5, 6)$	$\{\{h, t, r, h[\,], t[\,]\}\}$

Caveat:

In order to find something, we must assume that variables / addresses always receive a value before they are accessed.

Complexity:

we havve:

$\mathcal{O}(\# \, edges + \# \, Vars)$	calls of	union*
$\mathcal{O}(\# \, edges + \# \, Vars)$	calls of	find
$\mathcal{O}(\# \, Vars)$	calls of	union

$\Longrightarrow$ We require efficient Union-Find data-structure :-)

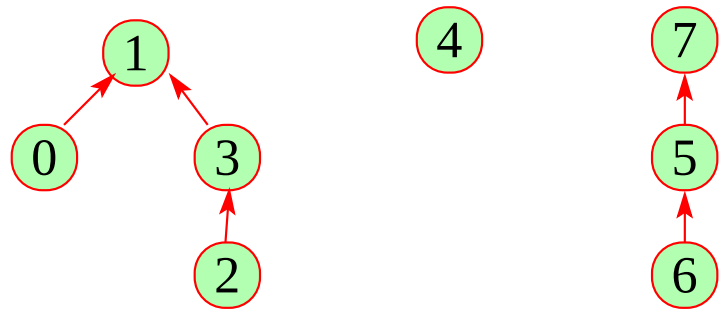
Idea:

Represent partition of U as directed forest:

- For $u \in U$ a reference $F[u]$ to the father is maintained;
- Roots are elements u with $F[u] = u$.

Single trees represent equivalence classes.

Their roots are their representatives ...

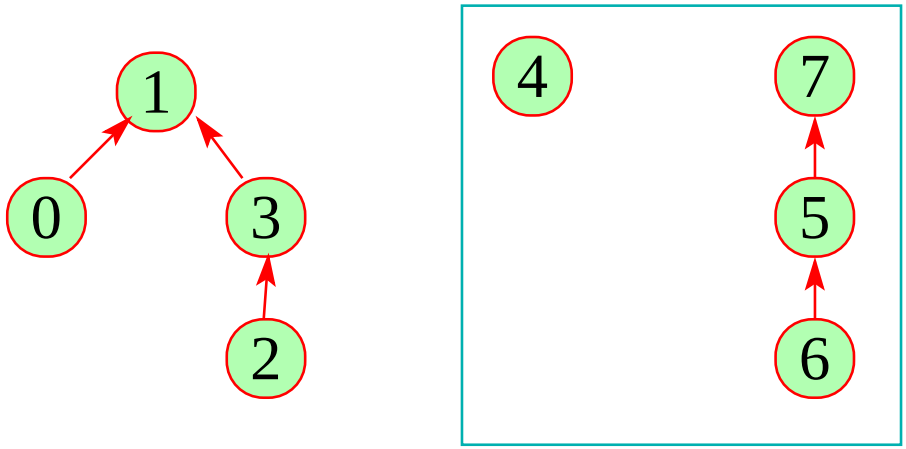


0	1	2	3	4	5	6	7

1	1	3	1	4	7	5	7

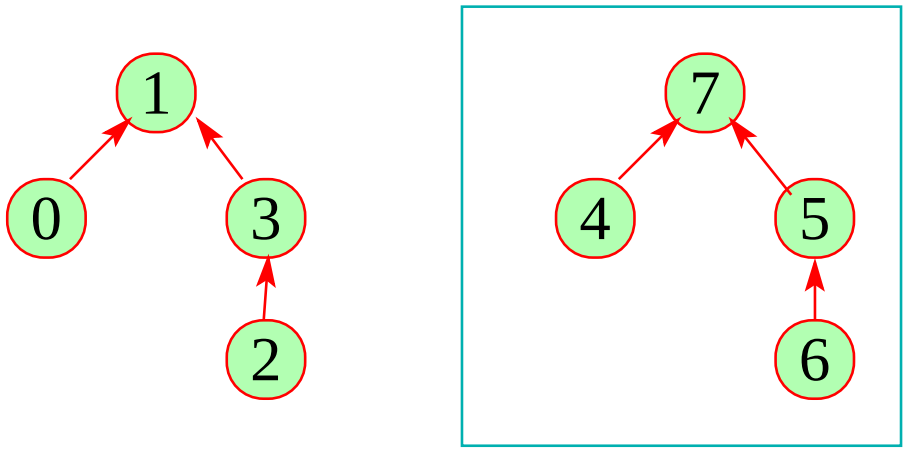
$\rightarrow$ find (π, u) follows the father references :-)

$\rightarrow$ union (π, u_1, u_2) re-directs the father reference of one u_i ...



0	1	2	3	4	5	6	7

1	1	3	1	4	7	5	7



0	1	2	3	4	5	6	7

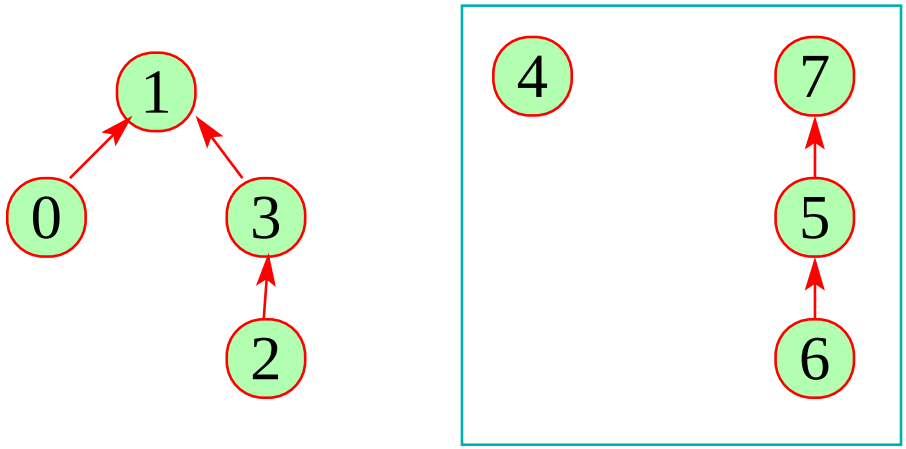
1	1	3	1	7	7	5	7

The Costs:

union	:	$\mathcal{O}(1)$	:-)
find	:	$\mathcal{O}(depth(\pi))$	:-(

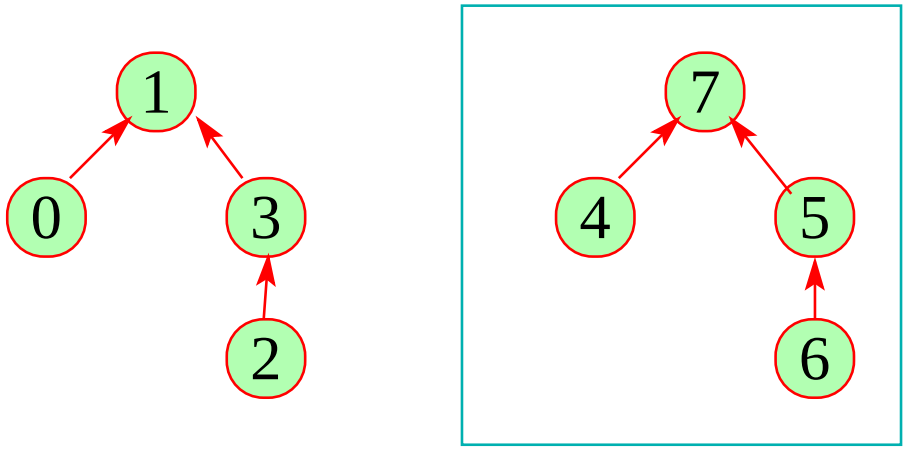
Strategy to Avoid Deep Trees:

- Put the smaller tree below the bigger !
- Use find to compress paths ...



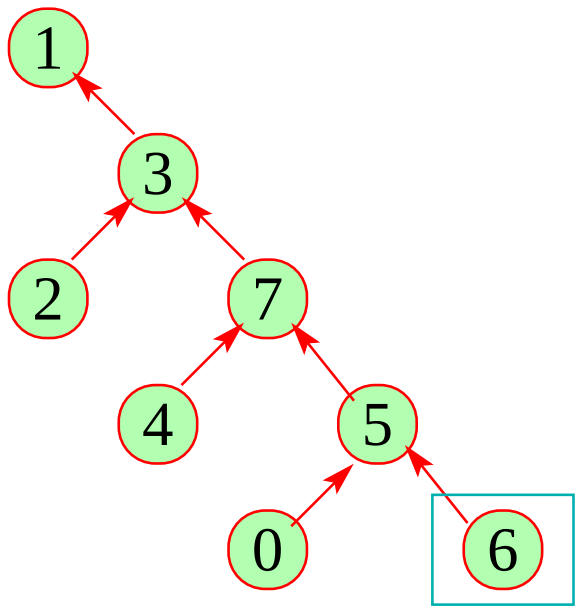
0	1	2	3	4	5	6	7

1	1	3	1	4	7	5	7

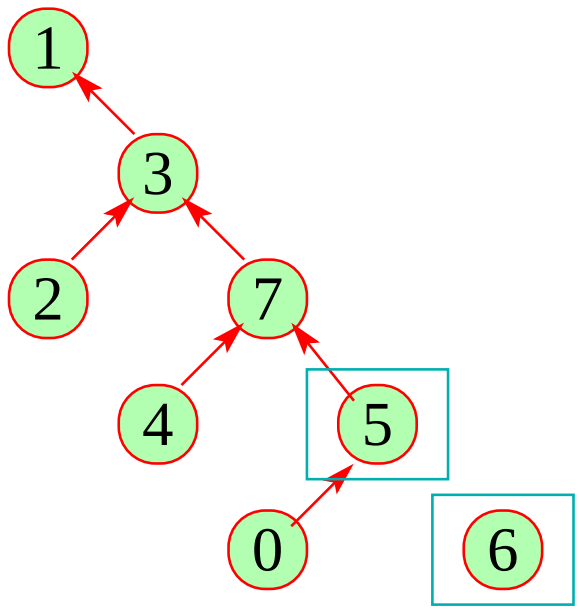


0	1	2	3	4	5	6	7

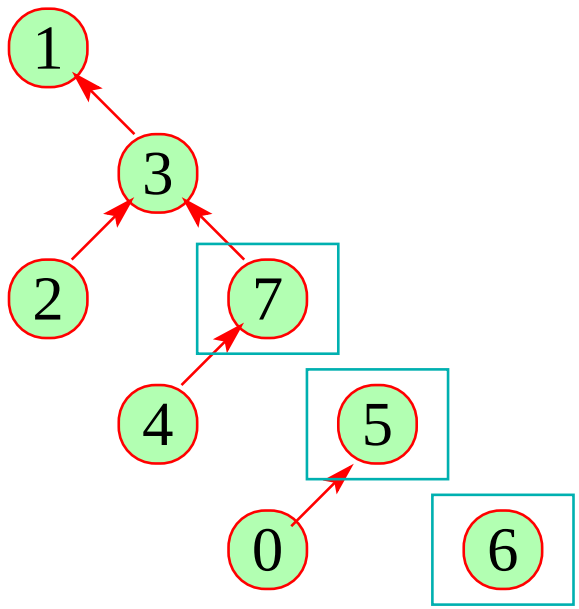
1	1	3	1	7	7	5	7



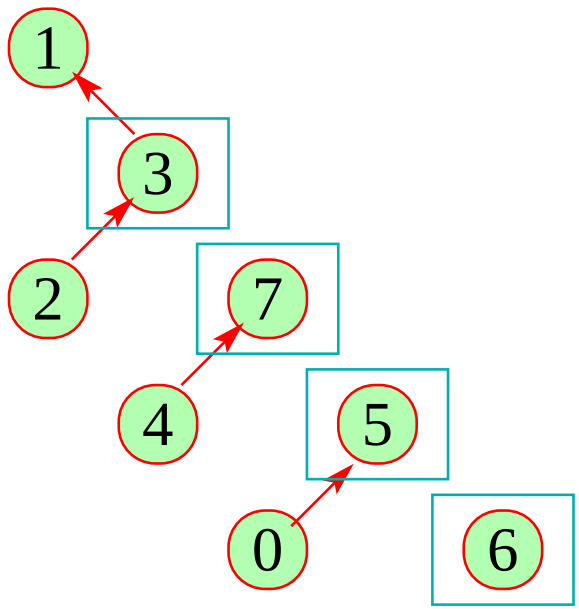
0	1	2	3	4	5	6	7
5	1	3	1	7	7	5	3



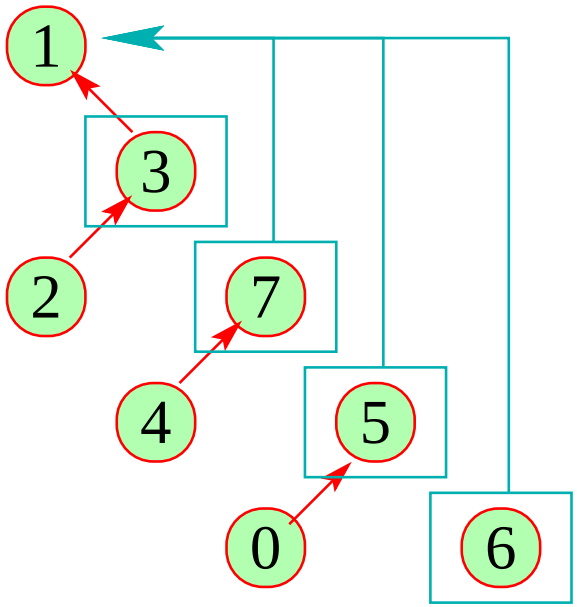
0	1	2	3	4	5	6	7
5	1	3	1	7	7	5	3



0	1	2	3	4	5	6	7
5	1	3	1	7	7	5	3



0	1	2	3	4	5	6	7
5	1	3	1	7	7	5	3



0	1	2	3	4	5	6	7
5	1	3	1	1	7	1	1

Robert Endre Tarjan, Princeton

Note:

- By this data-structure, n union- und m find operations require time $\mathcal{O}(n + m \cdot \alpha(n, n))$

  // α the inverse Ackermann-function :-)

- For our application, we only must modify union such that roots are from *Vars* whenever possible.
- This modification does not increase the asymptotic run-time. :-)

Summary:

The analysis is extremely fast — but may not find very much.

Background 3: Fixpoint Algorithms

Consider: $x_i \sqsupseteq f_i(x_1, \ldots, x_n), \quad i = 1, \ldots, n$

Observation:

RR-Iteration is inefficient:

- $\rightarrow$ We require a complete round in order to detect termination :-(
- $\rightarrow$ If in some round, the value of just one unknown is changed, then we still re-compute all :-(
- $\rightarrow$ The practical run-time depends on the ordering on the variables :-(