

GVNPRE in LLVM

Project Presentation

Prashanth Radhakrishnan
CS7968 - Program Analysis
April 19, 2007

PRE

```
if (OPAQUE)
    x = a + b;
else
    x = 0;
y = a + b;
```

PRE

```
if (OPAQUE)
    x = a + b;
    t = x;
else
    x = 0;
    t = a + b;
y = t;
```

- Powerful -> subsumes CSE & LICM
- Traditionally, lexical equivalence

GVN

```
input t1, t2, t3  
t4 = t2 + t3  
t5 = t3  
t6 = t2 + t5
```

GVN

```
v1: t1  
v2: t2  
v3: t3, t5  
v4: t4, v2 + v3, t6
```

- Global => across basic blocks
- Works on SSA form

GVNPRE

```
c = a;  
if (OPAQUE)  
    x = a + b;  
else  
    x = 0;  
y = c + b;
```

GVN-PRE

```
c = a;  
if (OPAQUE)  
    x = a + b;  
    t = x;  
else  
    x = 0;  
    t = c + b;  
y = t;
```

- Traditional PRE can't do this (assume no copy-propagation)
- Due to VanDrunen et al. [CC 2004]
- Assumes SSA, Dominator & PostDominator Trees, lack of critical edges

What I planned to do

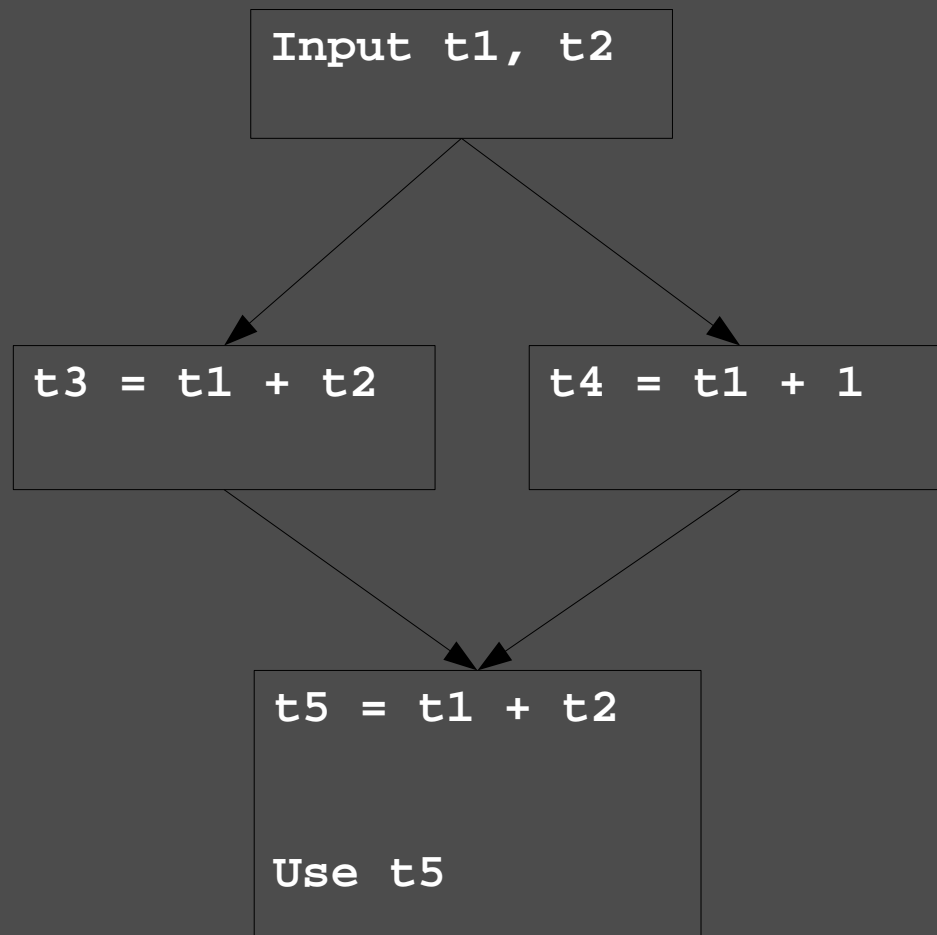
- Implement GVNPRE in LLVM
- Show it works right
- See if it subsumes LLVM's GCSE & LICM

What I planned to do

- Implement GVN-PRE in LLVM
- Show it works right
- Show it subsumes LLVM's GCSE & LICM
- And, Why?
 - No PRE pass in LLVM
 - Substitute for GCSE + LICM
 - Listed in open proj
 - GVNPRE "seemed" straightforward

How GVNPRE works

Example CFG



How GVNPRE works

Global Value Numbering (DomTree)

Value Table:

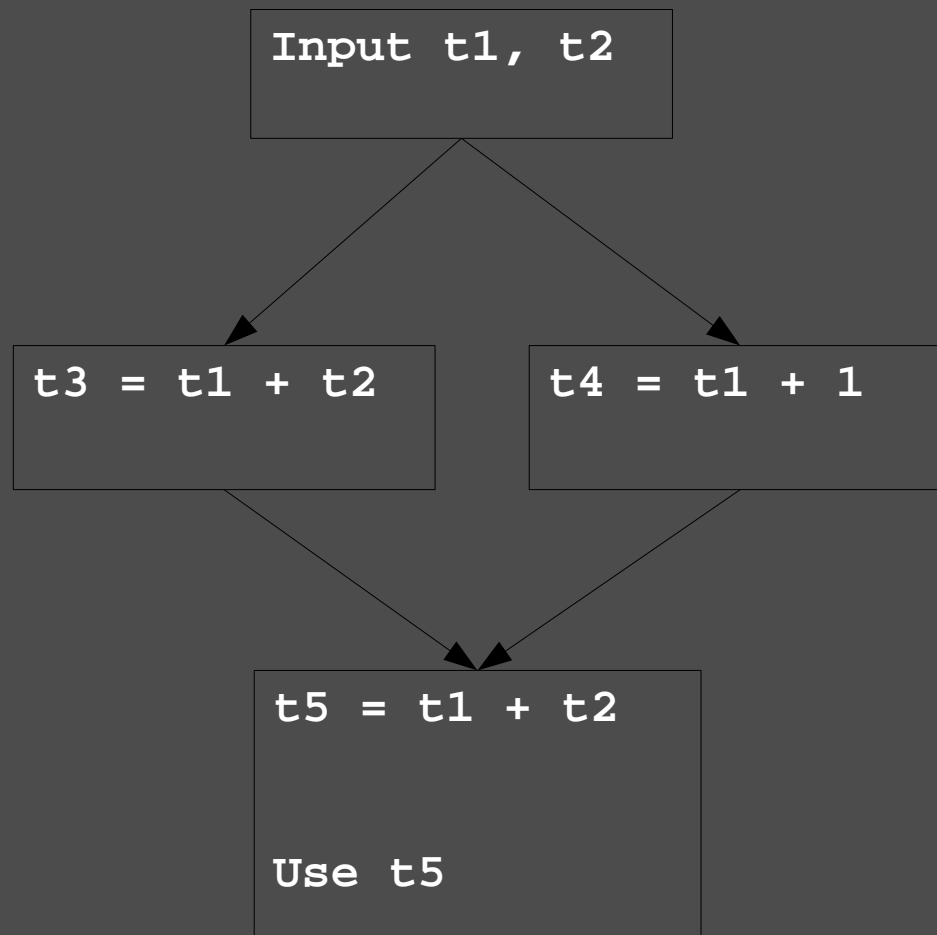
v0: 1

v1: t1

v2: t2

v3: t3, v1 + v2, t5

v4: t4, v1 + v0



How GVNPRE works

Available Sets (Dom) - Leader sets

Value Table:

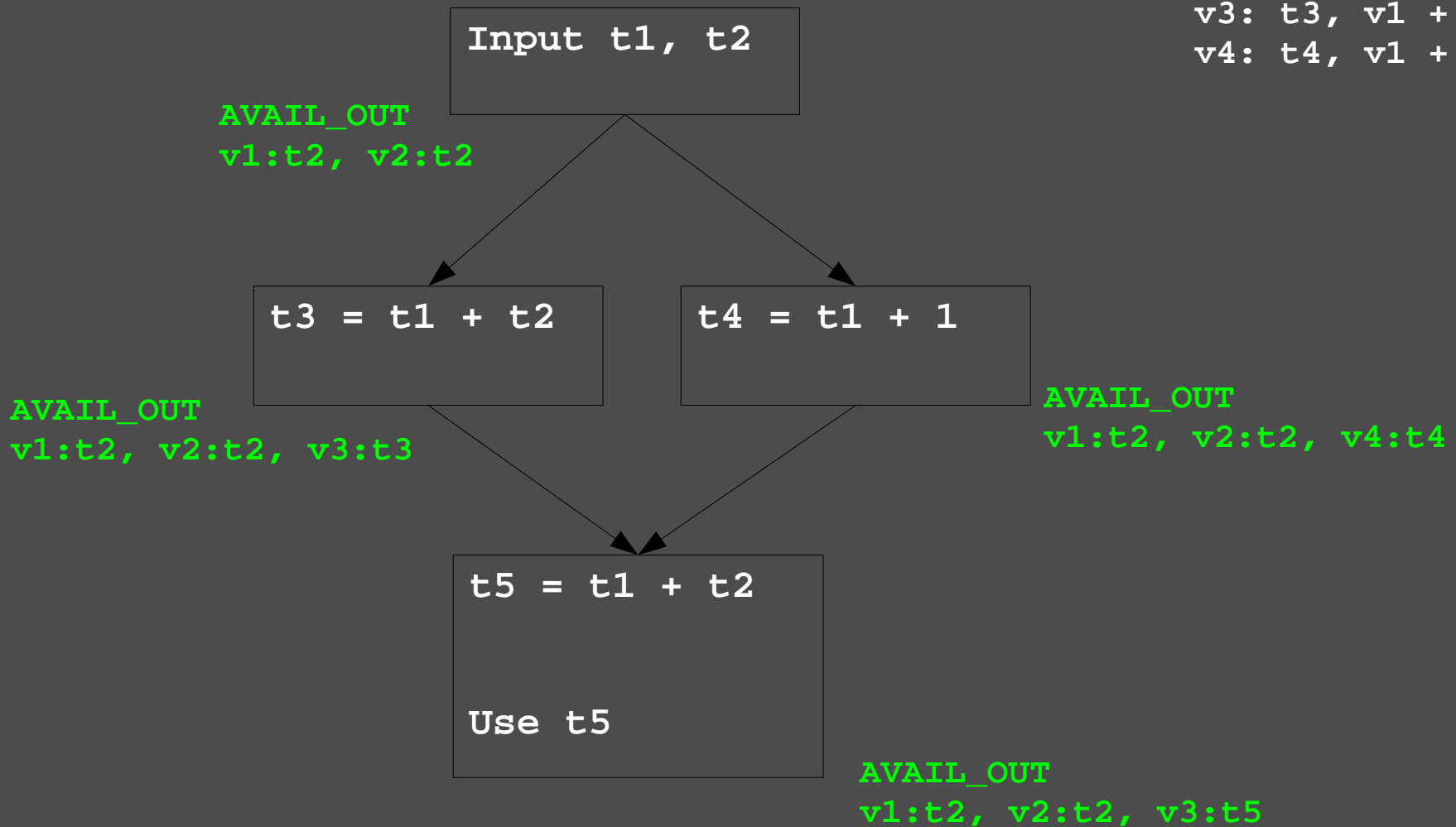
v0: 1

v1: t1

v2: t2

v3: t3, v1 + v2, t5

v4: t4, v1 + v0



How GVNPRE works

Anticipated Sets (PostDom) - AntiLeader Sets

Value Table:

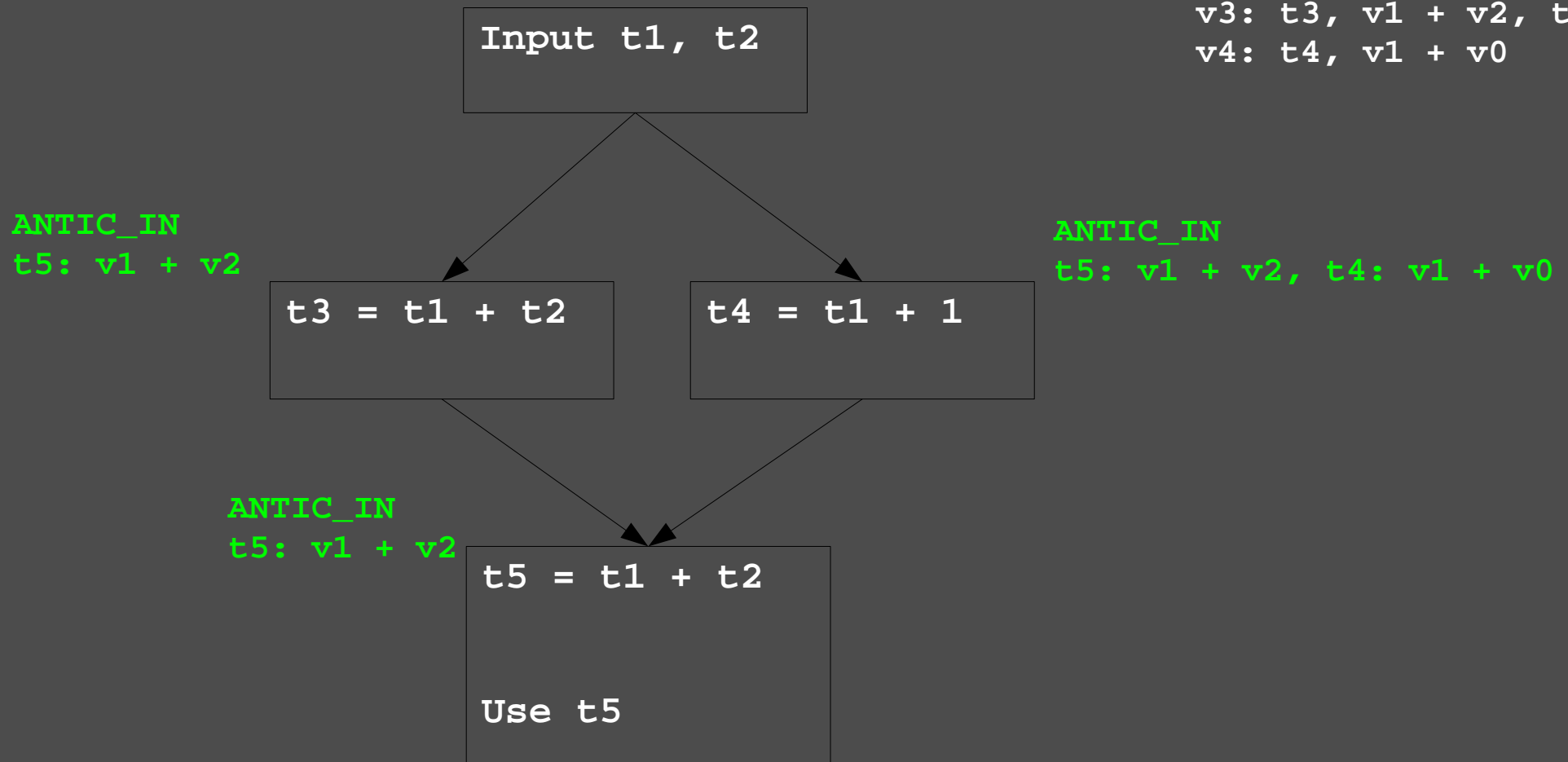
v0: 1

v1: t1

v2: t2

v3: t3, v1 + v2, t5

v4: t4, v1 + v0



How GVNPRE works

Insert - identify partial redundancy

Value Table:

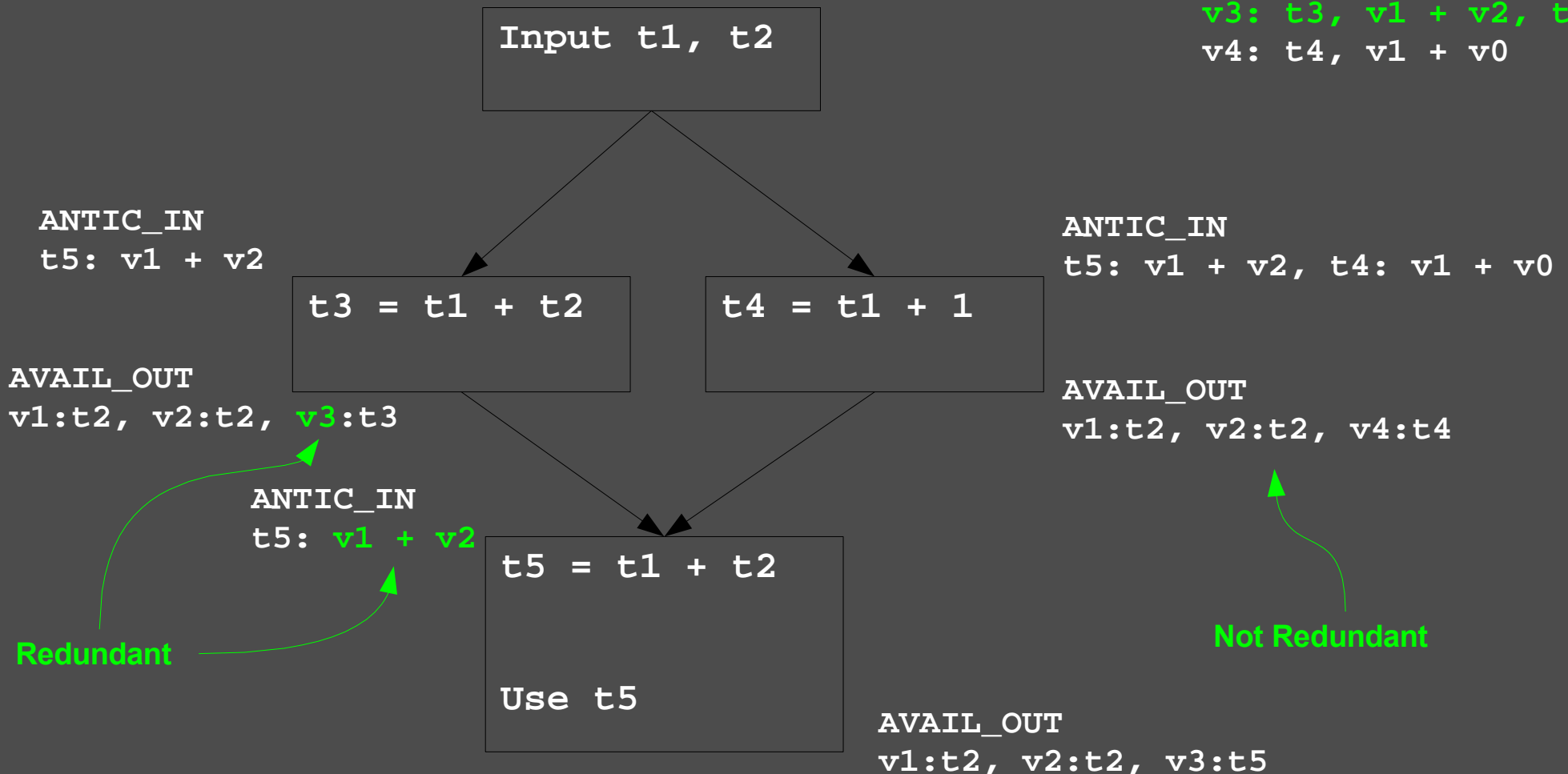
v0: 1

v1: t1

v2: t2

v3: t3, v1 + v2, t5

v4: t4, v1 + v0

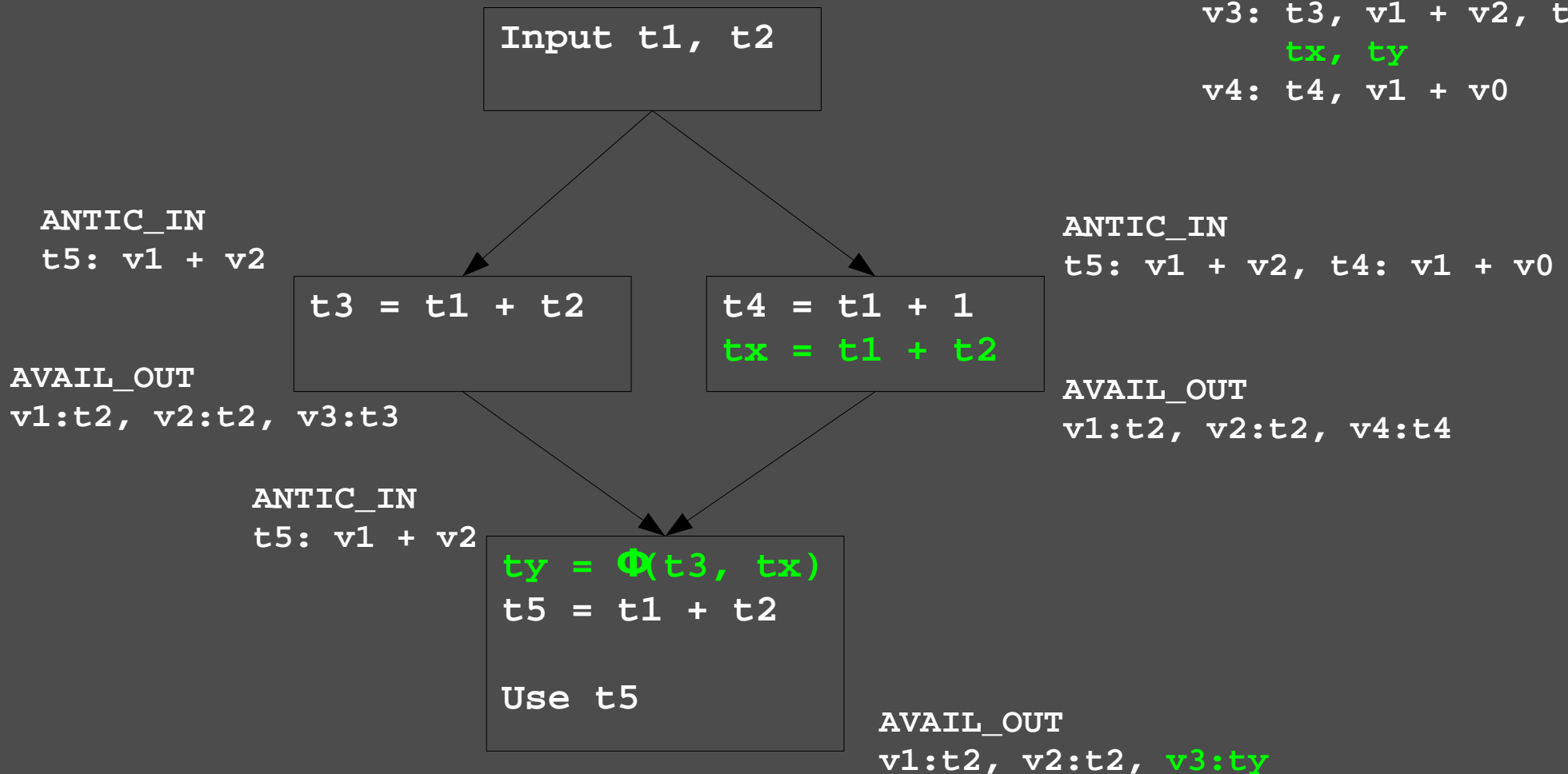


How GVNPRE works

Insert - convert to full redundancy

Value Table:

v0: 1
v1: t1
v2: t2
v3: t3, v1 + v2, t5
tx, ty
v4: t4, v1 + v0



How GVNPRE works

Eliminate - Identify instructions to be removed

Value Table:

v0: 1

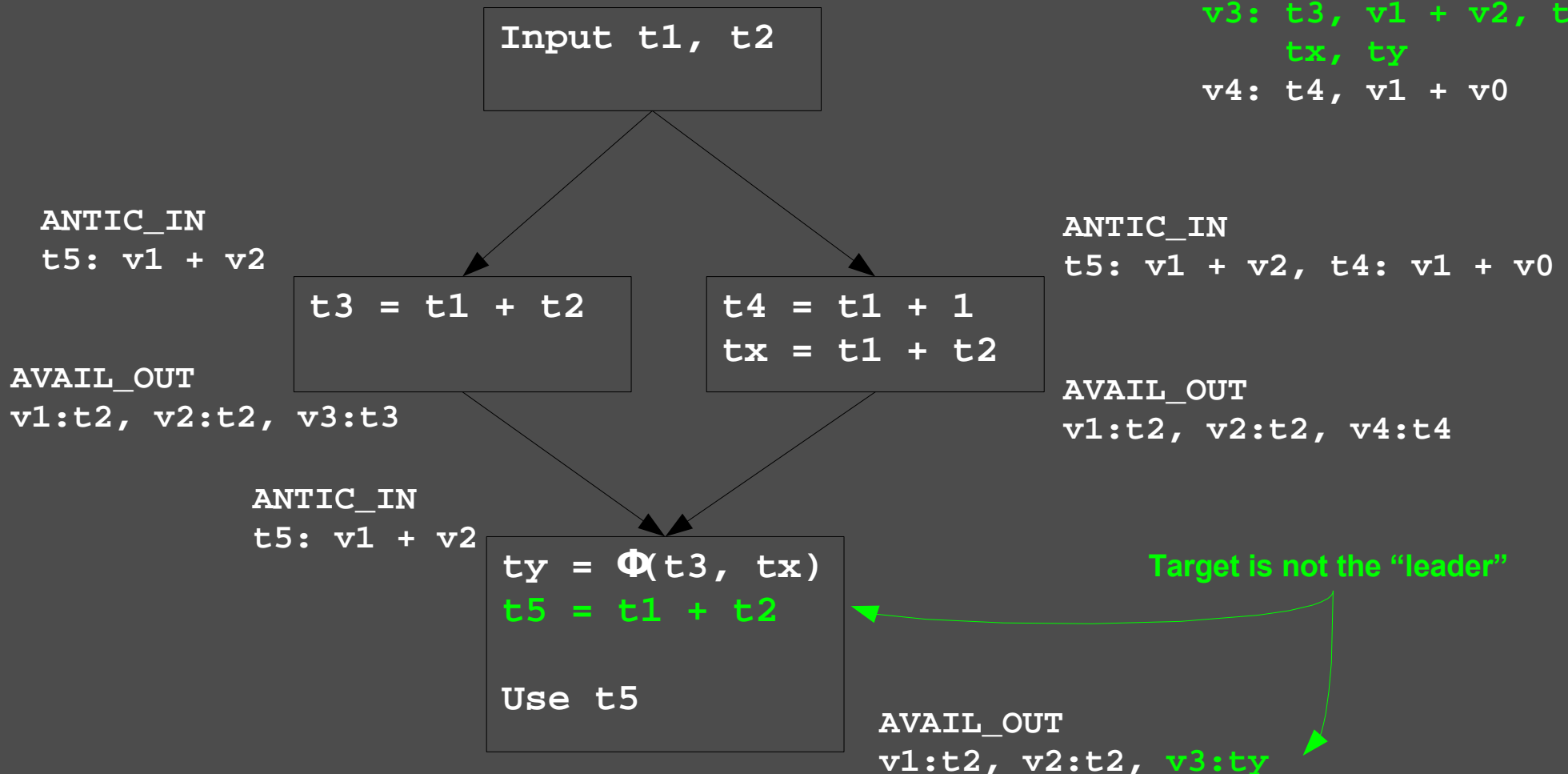
v1: t1

v2: t2

v3: t3, v1 + v2, t5

tx, ty

v4: t4, v1 + v0

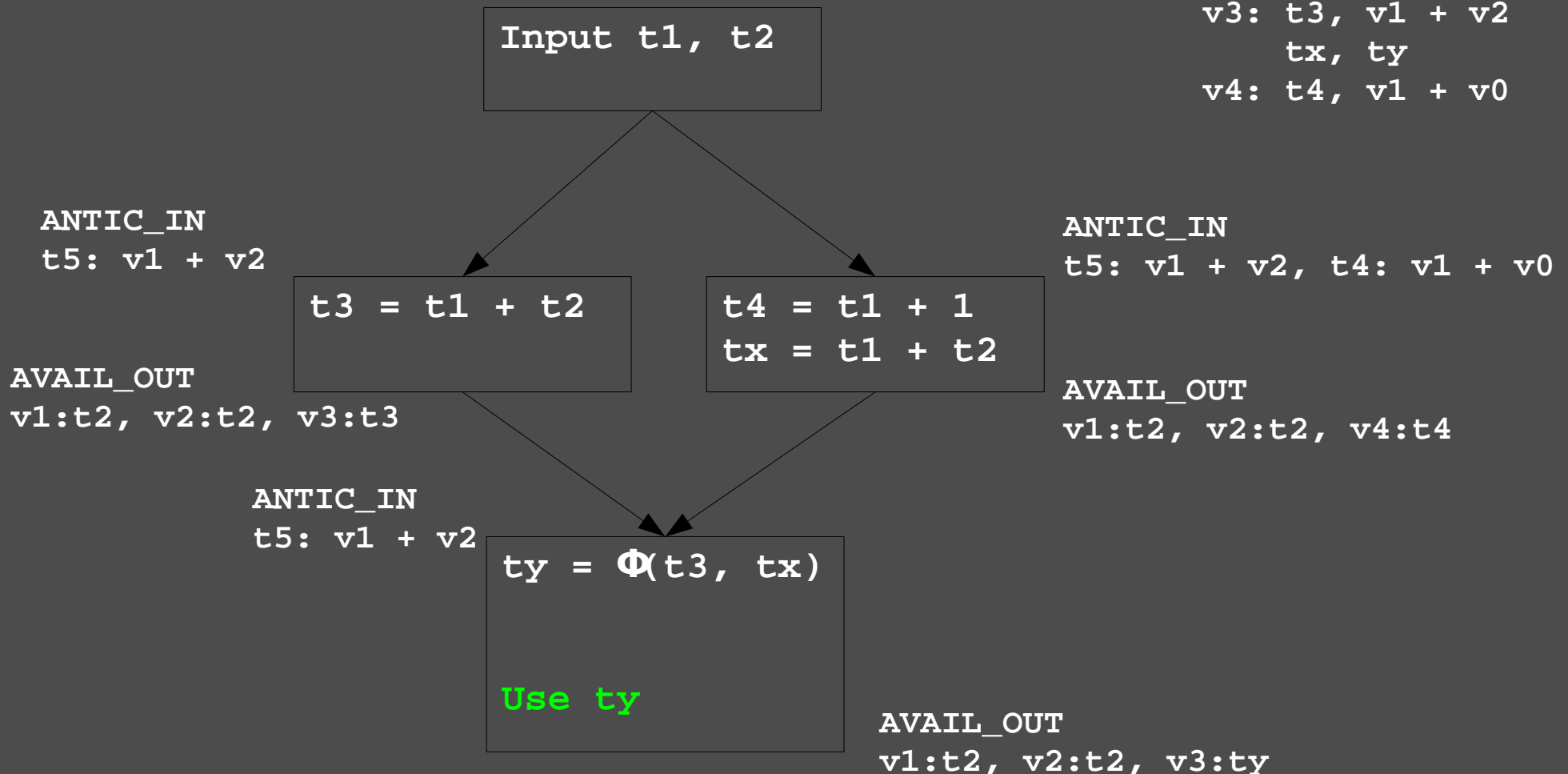


How GVNPRE works

Eliminate - Remove them and update uses

Value Table:

v0: 1
v1: t1
v2: t2
v3: t3, v1 + v2
tx, ty
v4: t4, v1 + v0



GVNPRE - the messy details

- Phi Translate values into predecessors
- Clean Antic sets => using DAGs
- Canonicalize Avail and Antic sets

$$\begin{aligned} \text{AVAIL_IN}[b] &= \text{AVAIL_OUT}[\text{dom}(b)] \\ \text{AVAIL_OUT}[b] &= \text{canon}(\text{AVAIL_IN}[b] \cup \text{PHI_GEN}(b) \\ &\quad \cup \text{TMP_GEN}(b)) \end{aligned}$$

$$\text{ANTIC_OUT}[b] = \begin{cases} \{e \mid e \in \text{ANTIC_IN}[\text{succ}_0(b)] \wedge \\ \forall b' \in \text{succ}(b), \exists e' \in \text{ANTIC_IN}[b'] \\ \text{lookup}(e) = \text{lookup}(e')\} & \text{if } |\text{succ}(b)| > 1 \\ \text{phi_translate}(\text{ANTIC_IN}[\text{succ}(b)], b, \\ \text{succ}(b)) & \text{if } |\text{succ}(b)| = 1 \end{cases}$$

$$\begin{aligned} \text{ANTIC_IN}[b] &= \text{clean}(\text{canon}_{\text{EXP}}(\text{ANTIC_OUT}[b] \cup \text{EXP_GEN}[b] \\ &\quad - \text{TMP_GEN}[b])) \end{aligned}$$

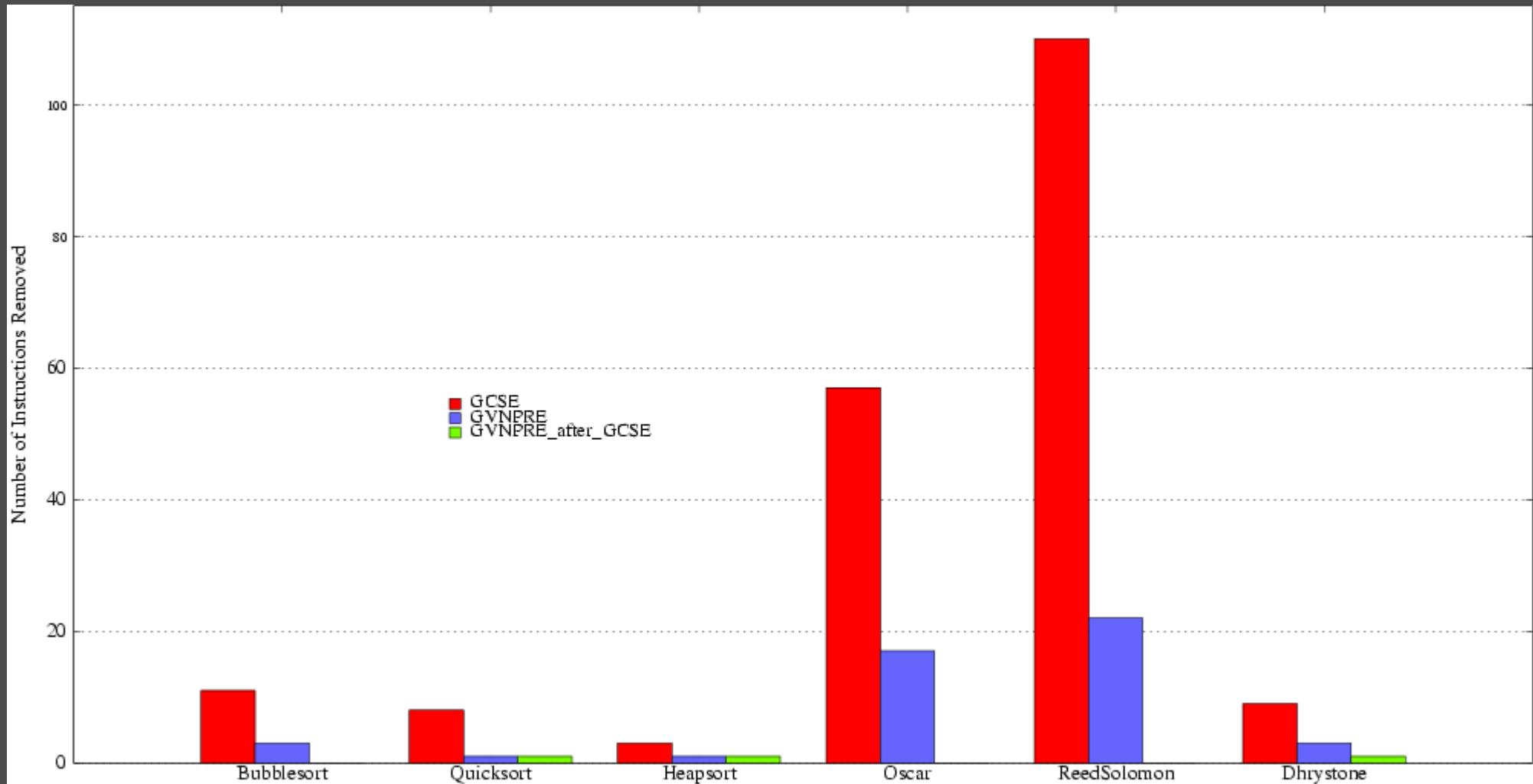
GVNPRE in LLVM - Highlights

- Implemented as a loadable module
- 5 passes through the code
 - GVN, Avail - Antic (FP) - Insert (FP) - Eliminate
- GVN not merged with Avail => reusable GVN
- ~2400 lines of STL-intensive C++ code

So, what was tough

- Things not detailed in the *GVNPRE* paper
 - Back-edges violating DAG property
 - $t3 = \Phi(t1, t2); t2 = t3 + 1;$
 - Handling "new" phi-translated expressions
- LLVM artifacts
 - Bug with loadable module having multiple passes
 - Inconsistency with constants
- Several unforeseen conditions...

How well (read badly) does it work



Several Limitations

- PRE is handled for binary operations only
 - Doesn't consider loads, casts, getelementptr etc. (that GCSE does)
- Constants are opaque to GVN
 - No constant folding (unlike GCSE)
- Handling DAGs when there are back-edges
 - Doesn't detect PR with cyclic expressions

In conclusion..

- GVNPRE has been implemented (somewhat inaccurately)
- Seems to work right
- Incapable of subsuming LLVM's GCSE or LICM in its current state

Thanks