• Deadline for HW4 extended to Monday
• HW5 out today
• Don’t wait until Tuesday to start HW5!
Data flow analysis
Recall: constant propagation

• Let $G = (N, E, s)$ be a control flow graph.

• $cp$ is the smallest$^1$ function such that
  
  - $cp(s) = \{x_1 \mapsto \top, \ldots, x_n \mapsto \top\}$
  
  - For each $p \rightarrow n \in E$, $post(p, cp(p)) \leq cp(n)$

\[ cp(s) = \{x_1 \mapsto \top, \ldots, x_n \mapsto \top\}; \]
\[ cp(n) = \{x_1 \mapsto \bot, \ldots, x_n \mapsto \bot\} \text{ for all other nodes}; \]

\[ work \leftarrow N \setminus \{s\}; \]

\[ \text{while work} \neq \emptyset \text{ do} \]
  
  \[ \text{Pick some } n \text{ from work; } \]
  \[ \text{work} \leftarrow \text{work} \setminus \{n\}; \]
  
  \[ C \leftarrow \bigsqcup_{p \in \text{pred}(n)} post(p, cp(p)); \]

  \[ \text{if } C \neq cp(n) \text{ then} \]
  \[ cp(n) \leftarrow C; \]
  \[ \text{work} \leftarrow \text{work} \cup \text{succ}(n) \]

$^1$Pointwise order: $f \preceq g$ if for all nodes $n$ and all variables $x$, $f(n)(x) \leq g(n)(x)$
```c
int sum2(int n) {
    int sum = 0;
    int step = 2;
    while (n > 0) {
        sum = sum + n;
        n = n - step;
    }
    return sum;
}
```
Common subexpression elimination

- Common subexpression elimination searches for expressions that
  - appear at multiple points in a program
  - evaluate to the same value at those points
  and (possibly) save the cost of re-evaluation by storing that value.

```c
void print (long *m, long n) {
    long i, j;
    for (i = 0; i < n*n; i += n) {
        for (j = 0; j < n; j += 1) {
            printf("%ld", *(m + i + j));
        }
        if (i + n < n*n) {
            printf("\n");
        }
    }
}
```

```c
void print (long *m, long n) {
    long i, j;
    long n_times_n = n*n;
    for (i = 0; i < n_times_n; i += n) {
        for (j = 0; j < n; j += 1) {
            printf("%ld", *(m + i + j));
        }
        long i_plus_n = i+n;
        if (i_plus_n < n_times_n) {
            printf("\n");
        }
        i = i_plus_n;
    }
}
```
Available expressions

• An expression in our simple simple imperative language has one of the following forms:
  • add <opn>, <opn>
  • mul <opn>, <opn>

• Fix control flow graph \( G = (N, E, s) \)

• An expression \( e \) is available at basic block \( n \in N \) if for every path from \( s \) to \( n \) in \( G \):
  1. the expression \( e \) is evaluated along the path
  2. after the last evaluation of \( e \) along the path, no variables in \( e \) are overwritten

• Idea: if expression \( e \) is available at node \( n \), then can eliminate redundant computations of \( e \) within \( n \)
```
i = 0
br loop

\[
t1 = n*n
\]
\[
t2 = -1*t1
\]
\[
t3 = i+t2
\]
\[
blz t3, body, exit
\]

\[
t4 = i+n
\]
\[
t5 = n*n
\]
\[
t6 = -1*t5
\]
\[
t7 = t4+t6
\]
\[
br t7, line, merge
\]

\[
i = i+n
\]
br loop

\[
\]
return

T

F

\[
line = line+1
\]
\[
br merge
\]
```
Given a set of expressions $E$ and an instruction $x = e$

Assuming the set of expressions $E$ is available before the instruction, what expressions are available after the instruction?
Propagating available expressions

- Given a set of expressions $E$ and an instruction $x = e$
  
  Assuming the set of expressions $E$ is available before the instruction, what expressions are available after the instruction?

  - $post_{AE}(x = e, E) = \{ e' \in E : x \text{ not in } e' \} \cup \{ e \}$
Propagating available expressions

- Given a set of expressions $E$ and an instruction $x = e$
  
  Assuming the set of expressions $E$ is available before the instruction, what expressions are available after the instruction?
  
  - $\text{post}_{AE}(x = e, E) = \{ e' \in E : \text{x not in } e' \} \cup \{ e \}$

- How do we propagate available expressions through a basic block?
Propagating available expressions

• Given a set of expressions $E$ and an instruction $x = e$
  Assuming the set of expressions $E$ is available before the instruction, what expressions are available after the instruction?
  
  \begin{itemize}
  \item $\text{post}_{AE}(x = e, E) = \{e' \in E : x \ \text{not in} \ e'\} \cup \{e\}$
  \end{itemize}

• How do we propagate available expressions through a basic block?
  
  \begin{itemize}
  \item Block takes the form $\text{instr}_1, \ldots, \text{instr}_n, \text{term}$.
  \item take $\text{post}_{AE}(\text{block}, E) = \text{post}_{AE}(\text{instr}_n, \ldots \text{post}_{AE}(\text{instr}_1, E))$
  \end{itemize}
Available expressions

- Let \( G = (N, E, s) \) be a control flow graph.
- \( ae \) is the smallest\(^2\) function such that
  - \( ae(s) = \emptyset \)
  - For each \( p \rightarrow n \in E, \text{post}_{AE}(p, ae(p)) \supseteq ae(n) \)

\[
\begin{align*}
  ae(s) &= \emptyset; \\
  ae(n) &= \{ \text{all expressions} \} \text{ for all other nodes;} \\
  \text{work} &\leftarrow N \setminus \{s\} ; \\
  \text{while work} \neq \emptyset \text{ do} \\
  &\quad \text{Pick some } n \text{ from work;} \\
  &\quad \text{work} \leftarrow \text{work} \setminus \{n\} ; \\
  &\quad E \leftarrow \bigcap_{p \in \text{pred}(n)} \text{post}_{AE}(p, ae(p)); \\
  &\quad \text{if } E \neq ae(n) \text{ then} \\
  &\quad \quad ae(n) \leftarrow E; \\
  &\quad \quad \text{work} \leftarrow \text{work} \cup \text{succ}(n) \\
\end{align*}
\]

\(^2\)Pointwise reverse-inclusion order: \( f \leq g \) if for all nodes \( n, f(n) \supseteq g(n) \)
\begin{align*}
i &= 0 \\
br \text{loop} \\
t1 &= n \times n \\
t2 &= -1 \times t1 \\
t3 &= i + t2 \\
\text{blz t3, body, exit} \\
\text{return}
\end{align*}

\begin{align*}
t4 &= i + n \\
t5 &= n \times n \\
t6 &= -1 \times t5 \\
t7 &= t4 + t6 \\
br \text{t7, line, merge} \\
i &= i + n \\
br \text{loop}
\end{align*}

\begin{align*}
\text{line} &= \text{line} + 1 \\
br \text{merge}
\end{align*}
Constant propagation

$cp$ is the *smallest* function such that

- $cp(s) = \{ x_1 \mapsto T, \ldots, x_n \mapsto T \}$
- For each $p \rightarrow n \in E$,
  
  $\text{post}(p, \text{cp}(p)) \leq \text{cp}(n)$

**Commonality:** $cp$ and $ae$ are *least solutions* to a system of local constraints

- “Local”: defined in terms of edges; contrast with “global”, which depends on the structure of the whole graph (e.g., paths)

Available expressions

$ae$ is the *smallest* function such that

- $ae(s) = \emptyset$
- For each $p \rightarrow n \in E$,
  
  $\text{post}_{AE}(p, \text{ae}(p)) \supseteq \text{ae}(n)$
Constant propagation

\[ cp(s) = \{ x_1 \mapsto \top, \ldots, x_n \mapsto \top \}; \]
\[ cp(n) = \{ x_1 \mapsto \bot, \ldots, x_n \mapsto \bot \} \text{ for all other nodes;} \]

work \leftarrow N \setminus \{ s \};

while work \neq \emptyset do

\begin{align*}
& \text{Pick some } n \text{ from work; } \\
& \text{work} \leftarrow \text{work} \setminus \{ n \}; \\
& C \leftarrow \bigcup_{p \in \text{pred}(n)} \text{post}(p, cp(p)); \\
& \text{if } C \neq cp(n) \text{ then } \\
& \quad \text{cp}(n) \leftarrow C; \\
& \quad \text{work} \leftarrow \text{work} \cup \text{succ}(n)
\end{align*}

Available expressions

\[ ae(s) = \emptyset; \]
\[ ae(n) = \{ \text{all expressions} \} \text{ for all other nodes;} \]

work \leftarrow N \setminus \{ s \};

while work \neq \emptyset do

\begin{align*}
& \text{Pick some } n \text{ from work; } \\
& \text{work} \leftarrow \text{work} \setminus \{ n \}; \\
& E \leftarrow \bigcap_{p \in \text{pred}(n)} \text{post}_{AE}(p, ae(p)); \\
& \text{if } E \neq ae(n) \text{ then } \\
& \quad \text{ae}(n) \leftarrow E; \\
& \quad \text{work} \leftarrow \text{work} \cup \text{succ}(n)
\end{align*}

- The algorithms for computing \( cp \) and \( ae \) are essentially the same
Dataflow analysis

- **Dataflow analysis** is an approach to program analysis that unifies the presentation and implementation of many different analyses
- **Formulate** problem as a system of constraints
- **Solve** the constraints iteratively (using some variation of the workset algorithm)
- What now:
  - General theory & algorithms
  - Conditions under which the approach works
  - Guarantees about the solution
- Not covered: *abstract interpretation* – a general theory for relating program analysis to program semantics
  - What does it mean for a constraint system to be correct?
  - How do we prove it?
A (forward) dataflow analysis consists of:

- **An abstract domain** $\mathcal{L}$
  - Defines the space of program “properties” that we are interested in

- **An abstract transformer** $\text{post}_L$
  - Determines how each basic block transforms properties
  - i.e., if property $p$ holds before $n$, then $\text{post}_L(n, p)$ is a property that holds after $n$
Abstract domains

An abstract domain is a set $\mathcal{L}$ equipped with:

- A partial order $\sqsubseteq$
  - $x \sqsubseteq y$ means that $x$ represents more precise information about the program than $y$\(^3\)
  - Technical requirement: ascending chain condition – any infinite ascending sequence

$$x_1 \sqsubseteq x_2 \sqsubseteq x_3 \sqsubseteq \ldots$$

must eventually stabilize: for some $i$, we have $x_j = x_i$ for all $j \geq i$.

---

\(^3\)The other direction also works, and is the one taken in classical compilers literature. In this class, we will stick to this direction, which is the convention established in abstract interpretation.
Abstract domains

An abstract domain is a set $\mathcal{L}$ equipped with:

- A partial order $\sqsubseteq$
  - $x \sqsubseteq y$ means that $x$ represents more precise information about the program than $y$\(^3\)
  - Technical requirement: ascending chain condition – any infinite ascending sequence
    $x_1 \sqsubseteq x_2 \sqsubseteq x_3 \sqsubseteq \ldots$
    must eventually stabilize: for some $i$, we have $x_j = x_i$ for all $j \geq i$.
- A least upper bound ("join") operator, $\sqcup$
  1. $x \sqsubseteq x \sqcup y$
  2. $y \sqsubseteq x \sqcup y$
  3. $x \sqcup y \sqsubseteq z$ for any $z$ satisfying 1 and 2
- A least element ("bottom"), $\bot$
- A greatest element ("top"), $\top$

---

\(^3\)The other direction also works, and is the one taken in classical compilers literature. In this class, we will stick to this direction, which is the convention established in abstract interpretation.
Abstract domains

An abstract domain is a set \( \mathcal{L} \) equipped with:

- A partial order \( \sqsubseteq \)
  - \( x \sqsubseteq y \) means that \( x \) represents more precise information about the program than \( y \)
  - Technical requirement: ascending chain condition – any infinite ascending sequence
    \[
    x_1 \sqsubseteq x_2 \sqsubseteq x_3 \sqsubseteq \ldots
    \]
    must eventually stabilize: for some \( i \), we have \( x_j = x_i \) for all \( j \geq i \).
- A least upper bound ("join") operator, \( \sqcup \)
  1. \( x \sqsubseteq x \sqcup y \)
  2. \( y \sqsubseteq x \sqcup y \)
  3. \( x \sqcup y \sqsubseteq z \) for any \( z \) satisfying 1 and 2
- A least element ("bottom"), \( \bot \)
- A greatest element ("top"), \( \top \)

What are the abstract domains of constant propagation & available expressions?

\(^3\)The other direction also works, and is the one taken in classical compilers literature. In this class, we will stick to this direction, which is the convention established in abstract interpretation.
Transfer functions

A transfer function $post_L : Basic\ Block \times \mathcal{L} \rightarrow \mathcal{L}$

- Technical requirement: $post_L$ is monotone

$$x \sqsubseteq y \Rightarrow post_L(n, x) \sqsubseteq post_L(n, y)$$

(“more information in $\Rightarrow$ more information out”)

- Desirable property: $post_L$ is distributive

$\text{post}_{AE}$ is distributive

$\text{post}_{CP}$ is not (why?)

- General family of distributive transfer functions: "gen/kill" analyses.

- Suppose we have a finite set of data flow "facts"

- Elements of the abstract domain are sets of facts

- For each basic block $n$, associate a set of generated facts $gen(n)$ and killed facts $kill(n)$

- Define $post_L(n, F_{\mathcal{L}}) = (F_{\mathcal{L}} \setminus kill(n)) \cup gen(n)$

$post_L$ is distributive!
Transfer functions

A transfer function $post_{\mathcal{L}} : Basic\ Block \times \mathcal{L} \rightarrow \mathcal{L}$

- Technical requirement: $post_{\mathcal{L}}$ is montone

$$x \sqsubseteq y \Rightarrow post_{\mathcal{L}}(n, x) \sqsubseteq post_{\mathcal{L}}(n, y)$$

(“more information in $\Rightarrow$ more information out”)

- Desirable property: $post_{\mathcal{L}}$ is distributive: for all $x, y \in L$,

$$post_{\mathcal{L}}(n, x \sqcup y) = post_{\mathcal{L}}(n, x) \sqcup post_{\mathcal{L}}(n, y)$$

- $post_{AE}$ is distributive
- $post_{CP}$ is not (why?)
Transfer functions

A transfer function $\text{post}_L : \text{Basic Block} \times L \rightarrow L$

- Technical requirement: $\text{post}_L$ is monotonous

$$x \sqsubseteq y \Rightarrow \text{post}_L(n, x) \sqsubseteq \text{post}_L(n, y)$$

(“more information in ⇒ more information out”)

- Desirable property: $\text{post}_L$ is distributive: for all $x, y \in L$,

$$\text{post}_L(n, x \sqcup y) = \text{post}_L(n, x) \sqcup \text{post}_L(n, y)$$

- $\text{post}_{AE}$ is distributive
- $\text{post}_{CP}$ is not (why?)
- General family of distributive transfer functions: “gen/kill” analyses.
  - Suppose we have a finite set of data flow “facts”
  - Elements of the abstract domain are sets of facts
  - For each basic block $n$, associate a set of generated facts $\text{gen}(n)$ and killed facts $\text{kill}(n)$
  - Define $\text{post}_L(n, F) = (F \setminus \text{kill}(n)) \cup \text{gen}(n)$. $\text{post}_L$ is distributive!
Generic (forward) dataflow analysis algorithm

- Given:
  - Abstract domain \((\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)\)
  - Transfer function 
    \(\text{post}_\mathcal{L} : \text{Basic Block} \times \mathcal{L} \rightarrow \mathcal{L}\)
  - Control flow graph \(G = (N, E, s)\)

- Compute: least function \(f\) such that
  1. \(f(s) = \top\)
  2. For all \(p \rightarrow n \in E, \text{post}_\mathcal{L}(p, f(p)) \sqsubseteq f(n)\)
Generic (forward) dataflow analysis algorithm

- Given:
  - Abstract domain \((\mathcal{L}, \subseteq, \sqcup, \bot, \top)\)
  - Transfer function \(\text{post}_\mathcal{L} : \text{Basic Block} \times \mathcal{L} \to \mathcal{L}\)
  - Control flow graph \(G = (N, E, s)\)
- Compute: least function \(f\) such that
  1. \(f(s) = \top\)
  2. For all \(p \rightarrow n \in E, \text{post}_\mathcal{L}(p, f(p)) \subseteq f(n)\)

\[
\begin{align*}
  f(s) &\leftarrow \top; \\
  f(n) &= \bot \text{ for all other nodes;} \\
  \text{work} &\leftarrow N \setminus \{s\}; \\
  \text{while } \text{work} \neq \emptyset \text{ do} \\
  \quad \text{Pick some } n \text{ from work;} \\
  \quad \text{work} \leftarrow \text{work} \setminus \{n\} ; \\
  \quad v &\leftarrow \bigsqcup_{p \in \text{pred}(n)} \text{post}_\mathcal{L}(p, f(p)); \\
  \quad \text{if } v \neq f(n) \text{ then} \\
  \quad \quad f(n) &\leftarrow v; \\
  \quad \quad \text{work} \leftarrow \text{work} \cup \text{succ}(n)
\end{align*}
\]
Generic (forward) dataflow analysis algorithm

- Given:
  - Abstract domain \((\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)\)
  - Transfer function \(\text{post}_\mathcal{L} : \text{Basic Block} \times \mathcal{L} \rightarrow \mathcal{L}\)
  - Control flow graph \(G = (N, E, s)\)

- Compute: least function \(f\) such that
  1. \(f(s) = \top\)
  2. For all \(p \rightarrow n \in E\), \(\text{post}_\mathcal{L}(p, f(p)) \sqsubseteq f(n)\)

Invariants:
- \(\text{work}\) contains all \(n \in N\) that may violate their constraints (\(\text{post}(p, f(p)) \not\sqsubseteq f(n)\) for some \(p \rightarrow n \in E\))
- Use \(f_i\) to denote \(f\) on the \(i\)th iteration and \(f^*\) to denote least solution to the constraint system. Then for all \(n\), \(f_i(n) \sqsubseteq f^*(n)\).

\[
\begin{align*}
f(s) &\leftarrow \top; \\
f(n) &= \bot \text{ for all other nodes}; \\
\text{work} &\leftarrow N \setminus \{s\}; \\
\text{while } \text{work} \neq \emptyset \text{ do} \\
&\quad \text{Pick some } n \text{ from work; } \\
&\quad \text{work} \leftarrow \text{work} \setminus \{n\}; \\
&\quad v \leftarrow \bigsqcup_{p \in \text{pred}(n)} \text{post}_\mathcal{L}(p, f(p)); \\
&\quad \text{if } v \neq f(n) \text{ then} \\
&\quad \quad f(n) \leftarrow v; \\
&\quad \quad \text{work} \leftarrow \text{work} \cup \text{succ}(n)
\end{align*}
\]
Generic (forward) dataflow analysis algorithm

- **Given:**
  - Abstract domain \((\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)\)
  - Transfer function \(\text{post}_\mathcal{L}: \text{Basic Block} \times \mathcal{L} \to \mathcal{L}\)
  - Control flow graph \(G = (N, E, s)\)

- **Compute:** least function \(f\) such that
  1. \(f(s) = \top\)
  2. For all \(p \to n \in E, \text{post}_\mathcal{L}(p, f(p)) \sqsubseteq f(n)\)

**Invariants:**
- \(\text{work}\) contains all \(n \in N\) that may violate their constraints \((\text{post}_\mathcal{L}(p, f(p)) \not\sqsubseteq f(n)\) for some \(p \to n \in E)\)
- Use \(f_i\) to denote \(f\) on the \(i\)th iteration and \(f^*\) to denote least solution to the constraint system. Then for all \(n, f_i(n) \sqsubseteq f^*(n)\).

**Termination:**
- Why does this algorithm terminate?

### Code

\[
\begin{align*}
f(s) &\leftarrow \top; \\
f(n) &\leftarrow \bot \text{ for all other nodes}; \\
work &\leftarrow N \setminus \{s\}; \\
\textbf{while } work \neq \emptyset \textbf{ do} \\
& \quad \text{Pick some } n \text{ from work}; \\
& \quad work \leftarrow work \setminus \{n\}; \\
& \quad v \leftarrow \bigsqcup_{p \in \text{pred}(n)} \text{post}_\mathcal{L}(p, f(p)); \\
& \quad \textbf{if } v \neq f(n) \textbf{ then} \\
& \quad \quad f(n) \leftarrow v; \\
& \quad \quad work \leftarrow work \cup \text{succ}(n)
\end{align*}
\]
Generic (forward) dataflow analysis algorithm

• Given:
  • Abstract domain \((\mathcal{L}, \sqsubseteq, \sqcup, \perp, \top)\)
  • Transfer function
    \(post_{\mathcal{L}} : \text{Basic Block} \times \mathcal{L} \rightarrow \mathcal{L}\)
  • Control flow graph \(G = (N, E, s)\)

• Compute: least function \(f\) such that
  \[\begin{align*}
  1 & \quad f(s) = \top \\
  2 & \quad \text{For all } p \rightarrow n \in E, post_{\mathcal{L}}(p, f(p)) \sqsubseteq f(n)
  \end{align*}\]

Invariants:
• \(work\) contains all \(n \in N\) that may violate their constraints \((post(p, f(p)) \nsubseteq f(n)\) for some \(p \rightarrow n \in E)\)
• Use \(f_i\) to denote \(f\) on the \(i\)th iteration and \(f^*\) to denote least solution to the constraint system. Then for all \(n\), \(f_i(n) \subseteq f^*(n)\).

Termination:
• Why does this algorithm terminate?
• Ascending chain condition \(\Rightarrow\) for each \(n, f_1(n) \subseteq f_2(n) \subseteq f_3(n) \subseteq \ldots\) must eventually stabilize