COS320: Compiling Techniques

Zak Kincaid

April 4, 2024

Generic (forward) dataflow analysis algorithm

- Given:
  - Abstract domain $(\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)$
  - Transfer function
    post$_{\mathcal{L}}$: *Basic Block* $\times \mathcal{L} \to \mathcal{L}$
  - Control flow graph $G = (N, E, s)$
- Compute: *least* annotation $\mathbf{IN}$, $\mathbf{OUT}$ such that
  ① $\mathbf{IN}[s] = \top$
  ② For all $n \in N$, $\mathit{post}_{\mathcal{L}}(n, \mathbf{IN}[n]) \sqsubseteq \mathbf{OUT}[n]$
  ③ For all $p \to n \in E$, $\mathbf{OUT}[p] \sqsubseteq \mathbf{IN}[n]$

```
$\mathbf{IN}[s] = \top$, $\mathbf{OUT}[s] = \bot$;
$\mathbf{IN}[n] = \mathbf{OUT}[n] = \bot$ for all other nodes $n$;
work $\leftarrow N$;
while work $\neq \emptyset$ do
    Pick some $n$ from work;
    work $\leftarrow$ work $\setminus \{n\}$ ;
    old $\leftarrow \mathbf{OUT}[n]$;
    $\mathbf{IN}[n] \leftarrow \mathbf{IN}[n] \sqcup \bigsqcup_{p \in \mathit{pred}(n)} \mathbf{OUT}[p]$;
    $\mathbf{OUT}[n] \leftarrow \mathit{post}_{\mathcal{L}}(n, \mathbf{IN}[n])$;
    if $\mathit{old} \neq \mathbf{OUT}[n]$ then
        work $\leftarrow$ work $\cup$ succ$(n)$
return $\mathbf{IN}$, $\mathbf{OUT}$
```

(Partial) Correctness

```
$\mathbf{IN}[s] = \top$, $\mathbf{OUT}[s] = \bot$;
$\mathbf{IN}[n] = \mathbf{OUT}[n] = \bot$ for all other nodes $n$;
$\textit{work} \leftarrow N$;
while $\textit{work} \neq \emptyset$ do
    Pick some $n$ from work;
    $\textit{work} \leftarrow \textit{work} \setminus \{n\}$ ;
    $\textit{old} \leftarrow \mathbf{OUT}[n]$;
    $\mathbf{IN}[n] \leftarrow \mathbf{IN}[n] \sqcup \bigsqcup_{p \in \textit{pred}(n)} \mathbf{OUT}[p]$;
    $\mathbf{OUT}[n] \leftarrow \textit{post}_{\mathcal{L}}(n, \mathbf{IN}[n])$;
    if $old \neq \mathbf{OUT}[n]$ then
        $\textit{work} \leftarrow \textit{work} \cup \textit{succ}(n)$
return $\mathbf{IN}, \mathbf{OUT}$
```

When algorithm terminates, all constraints are satisfied. Invariants:

- $\mathbf{IN}[s] = \top$
- For any $n \in N$, if $\textit{post}_{\mathcal{L}}(n, \mathbf{IN}[n]) \not\sqsubseteq \mathbf{OUT}[n]$, we have $n \in \textit{work}$
- For any $p \to n \in E$ with $\mathbf{OUT}[p] \not\sqsubseteq \mathbf{IN}(n)$, we have $n \in \textit{work}$

Optimality

Algorithm computes *least* solution.

- Invariant: $\mathbf{IN} \sqsubseteq^* \overline{\mathbf{IN}}$ and $\mathbf{OUT} \sqsubseteq^* \overline{\mathbf{OUT}}$, where
  - $\overline{\mathbf{IN}}/\overline{\mathbf{OUT}}$ denotes any solution to the constraint system
  - $\sqsubseteq^*$ is pointwise order on function space $N \to \mathcal{L}$

Optimality

Algorithm computes *least* solution.

- Invariant: $\mathbf{IN} \sqsubseteq^* \overline{\mathbf{IN}}$ and $\mathbf{OUT} \sqsubseteq^* \overline{\mathbf{OUT}}$, where
  - $\overline{\mathbf{IN}}/\overline{\mathbf{OUT}}$ denotes any solution to the constraint system
  - $\sqsubseteq^*$ is pointwise order on function space $N \to \mathcal{L}$
- Argument: let $\mathbf{IN}_i/\mathbf{OUT}_i$ be $\mathbf{IN}/\mathbf{OUT}$ at iteration i; n_i be workset item
  - Base case $\mathbf{IN}_0 \sqsubseteq^* \overline{\mathbf{IN}}$ and $\mathbf{OUT}_0 \sqsubseteq^* \overline{\mathbf{OUT}}$ is easy
  - Inductive step:
    - $\mathbf{IN}_{i+1}[n_i] = \mathbf{IN}_i[n_i] \sqcup \bigsqcup_{p \to n_i \in E} \mathbf{OUT}_i[p] \sqsubseteq \overline{\mathbf{IN}[n_i]} \sqcup \bigsqcup_{p \to n_i \in E} \overline{\mathbf{OUT}}[p] \sqsubseteq \overline{\mathbf{IN}}[n_i]$
    - $\mathbf{OUT}_{i+1}[n_i] = \textit{post}_{\mathcal{L}}(n_i, \mathbf{IN}_{i+1}[n_i]) \sqsubseteq \textit{post}_{\mathcal{L}}(n_i, \overline{\mathbf{IN}}[n_i]) \sqsubseteq \overline{\mathbf{OUT}}[n_i]$
    - For any $n \neq n_i$, $\mathbf{IN}_{i+1}[n] = \mathbf{IN}_i[n] \sqsubseteq \overline{\mathbf{IN}}[n_i]$

Termination

- Why does this algorithm terminate?

Termination

- Why does this algorithm terminate?
  - In general, it doesn’t

Termination

- Why does this algorithm terminate?
  - In general, it doesn't
- Ascending chain condition is sufficient.
  - A partial order $\sqsubseteq$ satisfies the ascending chain condition if any infinite ascending sequence

$$x_1 \sqsubseteq x_2 \sqsubseteq x_3 \sqsubseteq \dots$$

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

Termination

- Why does this algorithm terminate?
  - In general, it doesn't
- Ascending chain condition is sufficient.
  - A partial order $\sqsubseteq$ satisfies the ascending chain condition if any infinite ascending sequence

    $$x_1 \sqsubseteq x_2 \sqsubseteq x_3 \sqsubseteq \dots$$

    eventually stabilizes: for some i, we have $x_j = x_i$ for all $j \geq i$.
  - Fact: X is finite $\Rightarrow (2^X, \subseteq)$ and $(2^X, \supseteq)$ satisfy a.c.c. (*available expressions*)

Termination

- Why does this algorithm terminate?
  - In general, it doesn't
- Ascending chain condition is sufficient.
  - A partial order $\sqsubseteq$ satisfies the ascending chain condition if any infinite ascending sequence

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

    eventually stabilizes: for some i, we have $x_j = x_i$ for all $j \geq i$.
  - Fact: X is finite $\Rightarrow (2^X, \subseteq)$ and $(2^X, \supseteq)$ satisfy a.c.c. (*available expressions*)
  - Fact: X is finite and $(\mathcal{L}, \sqsubseteq)$ satisfies a.c.c. $\Rightarrow (X \rightarrow \mathcal{L}, \sqsubseteq^*)$ satisfies a.c.c. (*constant propagation*)

Termination

- Why does this algorithm terminate?
    - In general, it doesn't
- Ascending chain condition is sufficient.
    - A partial order $\sqsubseteq$ satisfies the ascending chain condition if any infinite ascending sequence

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

      eventually stabilizes: for some i, we have $x_j = x_i$ for all $j \geq i$.
    - Fact: X is finite $\Rightarrow (2^X, \subseteq)$ and $(2^X, \supseteq)$ satisfy a.c.c. (*available expressions*)
    - Fact: X is finite and $(\mathcal{L}, \sqsubseteq)$ satisfies a.c.c. $\Rightarrow (X \to \mathcal{L}, \sqsubseteq^*)$ satisfies a.c.c. (*constant propagation)*
- Termination argument:
    - If $(\mathcal{L}, \sqsubseteq)$ satisfies a.c.c., so does the space of annotations $(N \to \mathcal{L}, \sqsubseteq^*)$
    - $\mathbf{OUT}_0 \sqsubseteq^* \mathbf{OUT}_1 \sqsubseteq^* \ldots$, where $\mathbf{OUT}_i$ is the $\mathbf{OUT}$ annotation at iteration i
    - This sequence eventually stabilizes $\Rightarrow$ algorithm terminates

Local vs. Global constraints

- We had two specifications for available expressions
  - **Global**: e available at entry of n iff 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
  - **Local**: $\mathbf{IN}, \mathbf{OUT}$ is *least* annotation such that
    - ① $\mathbf{IN}[s] = \top$
    - ② For all $n \in N$, ***post***$_{AE}(n, \mathbf{IN}[n]) \sqsubseteq \mathbf{OUT}[n]$
    - ③ For all $p \rightarrow n \in E$, $\mathbf{OUT}[p] \sqsubseteq \mathbf{IN}(n)$
- *Why are these specifications the same?*

Coincidence

- Let $(\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)$ be an abstract domain and let $\mathit{post}_{\mathcal{L}}$ be a transfer function.
  - "Global specification" is formulated as *join over paths*:

$$\mathbf{JOP}[n] = \bigsqcup_{\pi \in \mathit{Path}(s,n)} \mathit{post}_{\mathcal{L}}(\pi, \top)$$

where $\mathit{Path}(s, n)$ denotes set of paths from s to n, and $\mathit{post}_{\mathcal{L}}$ is extended to paths by taking

$$\mathit{post}_{\mathcal{L}}(n_1 n_2 \ldots n_k, \top) = \mathit{post}_{\mathcal{L}}(n_k, \ldots, \mathit{post}_{\mathcal{L}}(n_1, \top))$$

Coincidence

- Let $(\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)$ be an abstract domain and let $\textit{post}_{\mathcal{L}}$ be a transfer function.
  - "Global specification" is formulated as *join over paths*:

$$\mathbf{JOP}[n] = \bigsqcup_{\pi \in \textit{Path}(s,n)} \textit{post}_{\mathcal{L}}(\pi, \top)$$

where $\textit{Path}(s, n)$ denotes set of paths from s to n, and $\textit{post}_{\mathcal{L}}$ is extended to paths by taking

$$\textit{post}_{\mathcal{L}}(n_1 n_2 \dots n_k, \top) = \textit{post}_{\mathcal{L}}(n_k, \dots, \textit{post}_{\mathcal{L}}(n_1, \top))$$

- **Coincidence theorem** (Kildall, Kam & Ullman): let $(\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)$ be an abstract domain satisfying the a.c.c., $\textit{post}_{\mathcal{L}}$ be a ***distributive*** transfer function, and $\mathbf{IN}/\mathbf{OUT}$ be least solution to
  1. $\mathbf{IN}[s] = \top$
  2. For all $n \in N$, $\textit{post}_{\mathcal{L}}(n, \mathbf{IN}[n]) \sqsubseteq \mathbf{OUT}[n]$
  3. For all $p \to n \in E$, $\mathbf{OUT}[p] \sqsubseteq \mathbf{IN}(n)$

  Then for all n, $\mathbf{JOP}[n] = \mathbf{IN}[n]$.

Coincidence

- Let $(\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)$ be an abstract domain and let $\mathit{post}_{\mathcal{L}}$ be a transfer function.
  - "Global specification" is formulated as *join over paths*:

$$\mathbf{JOP}[n] = \bigsqcup_{\pi \in \mathit{Path}(s,n)} \mathit{post}_{\mathcal{L}}(\pi, \top)$$

where $\mathit{Path}(s, n)$ denotes set of paths from s to n, and $\mathit{post}_{\mathcal{L}}$ is extended to paths by taking

$$\mathit{post}_{\mathcal{L}}(n_1 n_2 \dots n_k, \top) = \mathit{post}_{\mathcal{L}}(n_k, \dots, \mathit{post}_{\mathcal{L}}(n_1, \top))$$

- **Coincidence theorem** (Kildall, Kam & Ullman): let $(\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)$ be an abstract domain satisfying the a.c.c., $\mathit{post}_{\mathcal{L}}$ be a ***distributive*** transfer function, and $\mathbf{IN}/\mathbf{OUT}$ be least solution to
  1. $\mathbf{IN}[s] = \top$
  2. For all $n \in N$, $\mathit{post}_{\mathcal{L}}(n, \mathbf{IN}[n]) \sqsubseteq \mathbf{OUT}[n]$
  3. For all $p \to n \in E$, $\mathbf{OUT}[p] \sqsubseteq \mathbf{IN}(n)$

  Then for all n, $\mathbf{JOP}[n] = \mathbf{IN}[n]$.
- $\mathit{post}_{\mathcal{L}}$ is distributive if for all $x, y \in \mathcal{L}$, $\mathit{post}_{\mathcal{L}}(n, x \sqcup y) = \mathit{post}_{\mathcal{L}}(n, x) \sqcup \mathit{post}_{\mathcal{L}}(n, y)$

Available expressions

Recall transfer function *post*$_{AE}$ for available expressions:

$$\textit{post}_{AE}(x = e, E) = \{e' \in (E \cup \{e\}) : x \text{ not in } e'\}$$

post$_{AE}$ is distributive:

$$\begin{aligned}
\textit{post}_{AE}(x = e, E_1 \cap E_2) &= \{e' \in ((E_1 \cap E_2) \cup \{e\}) : x \text{ not in } e'\} \\
&= \{e' \in E_1 \cup \{e\}) : x \text{ not in } e'\} \cap \{e' \in (E_2 \cup \{e\}) : x \text{ not in } e'\} \\
&= \textit{post}_{AE}(x = e, E_1) \cap \textit{post}_{AE}(x = e, E_2)
\end{aligned}$$

Constant propagation

Is $post_{CP}$ distributive?

Constant propagation

Is *post*$_{CP}$ distributive?

$$\begin{aligned}\textit{post}_{CP}(x := x + y, \{x \mapsto 0, y \mapsto 1\} \sqcup \{x \mapsto 1, y \mapsto 0\}) &= \textit{post}_{CP}(x := x + y, \{x \mapsto \top, y \mapsto \top\}) \\ &= \{x \mapsto \top, y \mapsto \top\}\end{aligned}$$

Constant propagation

Is *post*$_{CP}$ distributive?

$$\textit{post}_{CP}(x := x + y, \{x \mapsto 0, y \mapsto 1\} \sqcup \{x \mapsto 1, y \mapsto 0\}) = \textit{post}_{CP}(x := x + y, \{x \mapsto \top, y \mapsto \top\})$$
$$= \{x \mapsto \top, y \mapsto \top\}$$

$$\textit{post}_{CP}(x := x + y, \{x \mapsto 0, y \mapsto 1\}) = \{x \mapsto 1, y \mapsto 1\}$$
$$\textit{post}_{CP}(x := x + y, \{x \mapsto 1, y \mapsto 0\}) = \{x \mapsto 1, y \mapsto 0\}$$
$$\{x \mapsto 1, y \mapsto 1\} \sqcup \{x \mapsto 1, y \mapsto 0\} = \{x \mapsto 1, y \mapsto \top\}$$

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*$_{\mathcal{L}}(n, F) = (F \setminus \textit{kill}(n)) \cup \textit{gen}(n)$.

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*$_{\mathcal{L}}(n, F) = (F \setminus \textit{kill}(n)) \cup \textit{gen}(n)$.
- The *order* on sets of facts may be $\subseteq$ or $\supseteq$
    - $\subseteq$ used for *existential* analyses: a fact holds at n if it holds along *some* path to n
        - E.g., a variable is possibly-uninitialized at n if it is possibly-uninitialized along some path to n.
    - $\supseteq$ used for *universal* analyses: a fact holds at n if it holds along *all* paths to n
        - E.g., an expression is available at n if it is available along all paths to n

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 $\textit{gen}(n)$ and *killed* facts $\textit{kill}(n)$
- Define $\textit{post}_{\mathcal{L}}(n, F) = (F \setminus \textit{kill}(n)) \cup \textit{gen}(n)$.
- The *order* on sets of facts may be $\subseteq$ or $\supseteq$
  - $\subseteq$ used for *existential* analyses: a fact holds at n if it holds along *some* path to n
    - E.g., a variable is possibly-uninitialized at n if it is possibly-uninitialized along some path to n.
  - $\supseteq$ used for *universal* analyses: a fact holds at n if it holds along *all* paths to n
    - E.g., an expression is available at n if it is available along all paths to n
- In either case, $\textit{post}_{\mathcal{L}}$ is monotone and distributive

$$\begin{aligned}
\textit{post}_{\mathcal{L}}(n, F \cup G) &= ((F \cup G) \setminus \textit{kill}(n)) \cup \textit{gen}(n) \\
&= ((F \setminus \textit{kill}(n)) \cup (G \setminus \textit{kill}(n))) \cup \textit{gen}(n) \\
&= ((F \setminus \textit{kill}(n)) \cup \textit{gen}(n)) \cup (((G \setminus \textit{kill}(n))) \cup \textit{gen}(n)) \\
&= \textit{post}_{\mathcal{L}}(n, F) \cup \textit{post}_{\mathcal{L}}(n, G)
\end{aligned}$$

Possibly-uninitialized variables analysis

- A variable x is possibly-uninitialized at a location n if there is some path from start to n along which x is never written to.
- If n *uses* an uninitialized variable, that could indicate undefined behavior
  - Can catch these errors at compile time using possibly-uninitialized variable analysis
  - E.g. `javac` does this by default
- Possibly-unintialized variables as a dataflow analysis problem:

Possibly-uninitialized variables analysis

- A variable x is possibly-uninitialized at a location n if there is some path from start to n along which x is never written to.
- If n *uses* an uninitialized variable, that could indicate undefined behavior
  - Can catch these errors at compile time using possibly-uninitialized variable analysis
  - E.g. `javac` does this by default
- Possibly-unintialized variables as a dataflow analysis problem:
  - Abstract domain: 2^{Var} (each $V \in 2^{\mathit{Var}}$ represents a set of possibly-uninitialized vars)
    - *Existential* $\Rightarrow$ order is $\subseteq$, join is $\cup$, $\top$ is *Var*, $\bot$ is $\emptyset$

Possibly-uninitialized variables analysis

- A variable x is possibly-uninitialized at a location n if there is some path from start to n along which x is never written to.
- If n *uses* an uninitialized variable, that could indicate undefined behavior
  - Can catch these errors at compile time using possibly-uninitialized variable analysis
  - E.g. `javac` does this by default
- Possibly-unintialized variables as a dataflow analysis problem:
  - Abstract domain: 2^{Var} (each $V \in 2^{\mathit{Var}}$ represents a set of possibly-uninitialized vars)
    - *Existential* $\Rightarrow$ order is $\subseteq$, join is $\cup$, $\top$ is *Var*, $\bot$ is $\emptyset$
  - $\mathit{kill}(x := e) = \{x\}$
  - $\mathit{gen}(x := e) = \emptyset$

Reaching definitions analysis

- A *definition* is a pair (n, x) consisting of a basic block n, and a variable x such that n contains an assignment to x.
- We say that a definitoin (n, x) *reaches* a node m if there is a path from start to m such that the latest definition of x along the path is at n
- Reaching definitions as a data flow analysis:

Reaching definitions analysis

- A *definition* is a pair (n, x) consisting of a basic block n, and a variable x such that n contains an assignment to x.
- We say that a definitoin (n, x) *reaches* a node m if there is a path from start to m such that the latest definition of x along the path is at n
- Reaching definitions as a data flow analysis:
  - Abstract domain: $2^{N \times \mathit{Var}}$
    - *Existential* ⇒ order is $\subseteq$, join is $\cup$, $\top$ is $N \times \mathit{Var}$, $\bot$ is $\emptyset$
  - $\mathit{kill}(n) = \{(m, x) : m \in N, (x := e) \text{ in } n\}$
  - $\mathit{gen}(n) = \{(n, x) : (x := e) \text{ in } n\}$

Wrap-up

- In a compiler, program analysis is used to inform optimization
  - Outside of compilers: verification, testing, software understanding…
- Dataflow analysis is a particular *family* of progam analyses, which operates by solving a constraint system over an ordered set
  - Gen/kill analysis are a sub-family with nice properties
  - The basic idea of solving constraints systems over ordered sets appears in lotss of different places!
    - Parsing – computation of first, follow, nullable
    - Networking – computing shortest parths
    - Automated planning – distance-to-goal estimation
    - …