COS320: Compiling Techniques

Zak Kincaid

April 16, 2019

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

Generic (forward) dataflow analysis algorithm

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

$f(s) \leftarrow \top$;
$f(n) = \bot$ for all other nodes;
work $\leftarrow N \setminus \{s\}$;
while *work* $\neq \emptyset$ **do**
    Pick some n from work;
    work $\leftarrow$ *work* $\setminus \{n\}$;
    $v \leftarrow \bigsqcup_{p \in \textit{pred}(n)} \textit{post}_{\mathcal{L}}(p, f(p))$;
    if $v \neq f(n)$ **then**
        $f(n) \leftarrow v$;
        work $\leftarrow$ *work* $\cup$ $\textit{succ}(n)$

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

$f(s) \leftarrow \top$;
$f(n) = \bot$ for all other nodes;
work $\leftarrow N \setminus \{s\}$;
while *work* $\neq \emptyset$ **do**
  Pick some n from work;
  work $\leftarrow$ *work* $\setminus \{n\}$;
  $v \leftarrow \bigsqcup_{p \in \textit{pred}(n)} \textit{post}_\mathcal{L}(p, f(p))$;
  if $v \neq f(n)$ **then**
    $f(n) \leftarrow v$;
    work $\leftarrow$ *work* $\cup$ *succ*(n)

Invariants:

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

Generic (forward) dataflow analysis algorithm

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

$f(s) \leftarrow \top$;
$f(n) = \bot$ for all other nodes;
work $\leftarrow N \setminus \{s\}$;
while *work* $\neq \emptyset$ **do**
  Pick some n from work;
  work $\leftarrow$ *work* $\setminus \{n\}$;
  $v \leftarrow \bigsqcup_{p \in \textit{pred}(n)} \textit{post}_{\mathcal{L}}(p, f(p))$;
  if $v \neq f(n)$ **then**
    $f(n) \leftarrow v$;
    work $\leftarrow$ *work* $\cup$ *succ*(n)

Invariants:

- *work* contains all $n \in N$ that may violate their constraints (*post*$(p, f(p)) \not\sqsubseteq f(n)$ for some $p \to n \in E$)
- Use f_i to denote f on the ith 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?

Generic (forward) dataflow analysis algorithm

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

$f(s) \leftarrow \top$;
$f(n) = \bot$ for all other nodes;
work $\leftarrow N \setminus \{s\}$;
while *work* $\neq \emptyset$ **do**
  Pick some n from work;
  work $\leftarrow$ *work* $\setminus \{n\}$;
  $v \leftarrow \bigsqcup_{p \in \mathit{pred}(n)} \mathit{post}_{\mathcal{L}}(p, f(p))$;
  if $v \neq f(n)$ **then**
    $f(n) \leftarrow v$;
    work $\leftarrow$ *work* $\cup\ \mathit{succ}(n)$

Invariants:

- *work* contains all $n \in N$ that may violate their constraints ($\mathit{post}(p, f(p)) \not\sqsubseteq f(n)$ for some $p \to n \in E$)
- Use f_i to denote f on the ith 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?
- Ascending chain condition $\Rightarrow$ for each n, $f_1(n) \sqsubseteq f_2(n) \sqsubseteq f_3(n) \sqsubseteq \ldots$ must eventually stabilize

Coincidence

- We had two specifications for available expressions
  - **Global**: $e \in \textit{ae}(n)$ 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**: *ae* is the *smallest* function such that
    - $\textit{ae}(s) = \emptyset$
    - For each $p \to n \in E$, $\textit{post}_{AE}(p, \textit{ae}(p)) \supseteq \textit{ae}(n)$
- *Why are these specifications the same?*

Coincidence

- We had two specifications for available expressions
  - **Global**: $e \in$ *ae*(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**: *ae* is the *smallest* function such that
    - *ae*$(s) = \emptyset$
    - For each $p \to n \in E$, *post*$_{AE}(p,$ *ae*$(p)) \supseteq$ *ae*(n)
- *Why are these specifications the same?*
- **Coincidence theorem** (Kildall, Kam & Ullman): for any abstract domain $(\mathcal{L}, \sqsubseteq, \sqcup, \bot, \top)$ and **distributive** transfer function *post*$_{\mathcal{L}}$, the least solution f to the constraint system
  1. $f(s) \sqsupseteq \top$
  2. For each $p \to n \in E$, *post*$_{\mathcal{L}}(p, f(p)) \sqsubseteq f(n)$

  coincides with the function $g(n) = \bigsqcup_{\pi \in Path(s,n)}$ *post*$_{\mathcal{L}}(\pi, \top)$, where *post*$_{\mathcal{L}}$ is extended to paths by taking

$$\textit{post}_{\mathcal{L}}(n_1 n_2 \ldots n_k, \top) = \textit{post}_{\mathcal{L}}(n_{k-1}, \ldots, \textit{post}_{\mathcal{L}}(n_1, \top))$$

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

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 $\textit{post}_{\mathcal{L}}(n, F) = (F \setminus \textit{kill}(n)) \cup \textit{gen}(n)$.
- 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 avaiable 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 *gen*(n) and *killed* facts *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 avaiable 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 $V \in 2^{Var}$ 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: $2^{N \times \textit{Var}}$
    - *Existential* $\Rightarrow$ order is $\subseteq$, join is $\cup$, $\top$ is $N \times$ *Var*, $\bot$ is $\emptyset$
  - *kill*$(n) = \{(m, x) : m \in N, (x := e)$ *in* $n\}$
  - *gen*$(n) = \{(n, x) : (x := e)$ *in* $n\}$