Dataflow Analysis: Part 2

October 6, 2023
Check List

Define semi-lattice

  Direction: forward / backward.
  
  Check the order (does it make sense?)
  
  Decide initial values.

Design the transfer functions.

  How does each statement affect the dataflow facts?
  
  Prove monotonicity.
### Review: Monotone Frameworks

<table>
<thead>
<tr>
<th>Available Expressions</th>
<th>Reaching Definitions</th>
<th>Very Busy Expressions</th>
<th>Live Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>$\mathcal{P}(\text{AExp}_*)$</td>
<td>$\mathcal{P}(\text{Var}_* \times \text{Lab}^?)$</td>
<td>$\mathcal{P}(\text{AExp}_*)$</td>
</tr>
<tr>
<td>$\subseteq$</td>
<td>$\supseteq$</td>
<td>$\subseteq$</td>
<td>$\subseteq$</td>
</tr>
<tr>
<td>$\sqcup$</td>
<td>$\cap$</td>
<td>$\cup$</td>
<td>$\cup$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>$\text{AExp}_*$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

| $\iota$               | $\emptyset$          | $\{(x, ?) \mid x \in \text{FV}(S_*)\}$ | $\emptyset$          |
| $E$                   | $\{\text{init}(S_*)\}$ | $\{\text{init}(S_*)\}$ | $\text{final}(S_*)$ |
| $F$                   | $\text{flow}(S_*)$    | $\text{flow}(S_*)$    | $\text{flow}^R(S_*)$ |

$\mathcal{F}$ $\{f : L \to L \mid \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g\}$

$f_\ell(l) = (l \setminus \text{kill}([B]^\ell)) \cup \text{gen}([B]^\ell)$ where $[B]^\ell \in \text{blocks}(S_*)$
Implementing Analyses in Tundra

Create new file in `analysis/` folder:

Define `AnalysisType`

Create a subclass of abstract `Analysis` class (`analysis/analysis.py`)

Define methods: `initial_in`, `initial_out`, and `get_new_values`

Useful functions in `lang/utils.py`
Example: Live Variables Analysis

A variable is **live** at the exit from a label if there exists a path from the label to a use of the variable that does not re-define the variable.
Check List: Live Variables Analysis

Define semi-lattice

Direction: forward / backward.

Check the order (does it make sense?)

Decide initial values.

Design the transfer functions.

How does each statement affect the dataflow facts?

Prove monotonicity.
### Example: Live Variables Analysis

<table>
<thead>
<tr>
<th></th>
<th>Live Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>(L)</td>
<td>(\mathcal{P}(\text{Var}_*))</td>
</tr>
<tr>
<td>(\subseteq)</td>
<td>(\subseteq)</td>
</tr>
<tr>
<td>(\cup)</td>
<td>(\cup)</td>
</tr>
<tr>
<td>(\bot)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(i)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(E)</td>
<td>(\text{final}(S_*))</td>
</tr>
<tr>
<td>(F)</td>
<td>(\text{flow}^R(S_*))</td>
</tr>
</tbody>
</table>

For analysis type `Set[Variable]`:

- 14 usages by Avery Laird

```python
class LiveVariables(Analysis[AnalysisType]):
    T = AnalysisType
```

- Avery Laird

```python
def initial_in(self, node: Node) -> T:
    return set()
```

- Avery Laird

```python
def initial_out(self, node: Node) -> T:
    return set()
```

\[\mathcal{F} = \{f : L \rightarrow L | \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g\}\]

\[f_\ell(l) = (l \setminus \text{kill}([B]^{\ell})) \cup \text{gen}([B]^{\ell})\text{ where } [B]^{\ell} \in \text{blocks}(S_*)\]
Tundra Demo
Example: Reaching Definitions

For each program point, which assignments may have been made and not overwritten, when program execution reaches this point along some path.

\[
\begin{align*}
L1: & \quad x = 0; \\
L2: & \quad y = 10;
\end{align*}
\]
Check List: Reaching Definitions

Define semi-lattice

  Direction: forward / backward.

  Check the order (does it make sense?)

  Decide initial values.

Design the transfer functions.

  How does each statement affect the dataflow facts?

  Prove monotonicity.
### Example: Reaching Definitions

#### Reaching Definitions

<table>
<thead>
<tr>
<th>$L$</th>
<th>$\mathcal{P}(\text{Var} \times \text{Lab})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\perp$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\sqcup$</td>
<td>$\cup$</td>
</tr>
<tr>
<td>$\sqsubseteq$</td>
<td>$\subseteq$</td>
</tr>
</tbody>
</table>

| $\iota$ | $\{ (x, ?) | x \in FV(S) \}$ |
|---------|-----------------------------|
| $E$     | $\{ \text{init}(S) \}$     |
| $F$     | $\text{flow}(S)$           |

| $\mathcal{F}$ | $\{ f : L \to L | \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g \}$ |
|---------------|------------------------------------------------------------------|
| $f_\ell$     | $f_\ell(l) = (l \setminus \text{kill}(B_\ell)) \cup \text{gen}(B_\ell)$ where $B_\ell \in \text{blocks}(S)$ |

*Coding Example*

```python
AnalysisType = Set[Tuple[Variable, Optional[Node]]]

6 usages ` Avery Laird*

class ReachingDefinitions(Analysis[AnalysisType]):
    ` Avery Laird
    def initial_in(self, node: Node) -> AnalysisType:
        return set()

    ` Avery Laird
    def initial_out(self, node: Node) -> AnalysisType:
        return set()
```
Tundra Demo
Example: Very Busy Expressions

An expression is very busy at the exit from a label if, no matter what path is taken from the label, the expression must always be used before any of the variables occurring in it are redefined.
Example: Very Busy Expressions

An expression is very busy at the exit from a label if, no matter what path is taken from the label, the expression must always be used before any of the variables occurring in it are redefined.

```c
if (a > b) {
    x = b - a;
    y = a - b;
} else {
    y = b - a;
    x = a - b;
}
```
Check List: Very Busy Expressions

Define semi-lattice

  Direction: forward / backward.

  Check the order (does it make sense?)

  Decide initial values.

Design the transfer functions.

  How does each statement affect the dataflow facts?

  Prove monotonicity.
## Example: Very Busy Expressions

<table>
<thead>
<tr>
<th>( L )</th>
<th>( \mathcal{P}(\text{AExp}_*) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sqsubseteq )</td>
<td>( \supseteq )</td>
</tr>
<tr>
<td>( \sqcup )</td>
<td>( \cap )</td>
</tr>
<tr>
<td>( \bot )</td>
<td>( \text{AExp}_* )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \iota )</th>
<th>( \emptyset )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E )</td>
<td>( \text{final}(S_*) )</td>
</tr>
<tr>
<td>( F )</td>
<td>( \text{flow}^R(S_*) )</td>
</tr>
</tbody>
</table>

\[ \mathcal{F} = \{ f : L \to L \mid \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g \} \]

\( f_{\ell} \) s.t.

\[ f_{\ell}(l) = (l \setminus \text{kill}([B]_{\ell})) \cup \text{gen}([B]_{\ell}) \text{ where } [B]_{\ell} \in \text{blocks}(S_*) \]

AnalysisType = Set[Expression]

```python
class VeryBusyExpressions(Analysis[AnalysisType]):
    T = AnalysisType

    def initial_in(self, node: Node) -> T:
        return set()

    def initial_out(self, node: Node) -> T:
        return set()
```

2 usages  Avery Laird
Tundra Demo
Example: Available Expressions

For each program point, which expressions must have already been computed, and not later modified, on all paths to the program point.
Example: Available Expressions

For each program point, which expressions must have already been computed, and not later modified, on all paths to the program point.

\[ x = a + b; \]
\[ l1: \ y = a + b; \]
Check List: Available Expressions

Define semi-lattice

- Direction: forward / backward.
- Check the order (does it make sense?)
- Decide initial values.

Design the transfer functions.

- How does each statement affect the dataflow facts?
- Prove monotonicity.
### Example: Available Expressions

<table>
<thead>
<tr>
<th>Available Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
</tr>
<tr>
<td>$\subseteq$</td>
</tr>
<tr>
<td>$\cup$</td>
</tr>
<tr>
<td>$\perp$</td>
</tr>
<tr>
<td>$\mathcal{P}(\text{AExp}_*)$</td>
</tr>
<tr>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${\text{init}(S_*)}$</td>
</tr>
<tr>
<td>$\text{flow}(S_*)$</td>
</tr>
</tbody>
</table>

$\mathcal{F} = \{f : L \rightarrow L \mid \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g\}$

$f_{\ell} = (l \setminus \text{kill}([B]_\ell)) \cup \text{gen}([B]_\ell)$ where $[B]_\ell \in \text{blocks}(S_*)$
Tundra Demo