Lecture Notes: Dataflow Analysis Notation

17-654/17-754: Analysis of Software Artifacts
Jonathan Aldrich (`jonathan.aldrich@cs.cmu.edu`)

Lectures 6-9

σ	Used to represent dataflow information (a lattice element)
$[x \mapsto NZ, y \mapsto Z]$	An element of a tuple lattice for zero analysis where the sub-lattice element for x is NZ and for y is Z
$[x \mapsto NZ]\sigma$	The tuple lattice that is the same as σ except that x is NZ
$\bot$	Bottom, the most precise element of a lattice. Bottom is assumed to be the initial dataflow value along back edges
$\top$	Top, the least precise element of a lattice
$\sigma_1 \sqsubseteq \sigma_2$	σ_1 is at least as precise as σ_2
$\sigma_1 \sqsupseteq \sigma_2$	σ_1 is at least as general as σ_2 (inverse of $\sqsubseteq$)
$\sigma_1 \sqcup \sigma_2$	The least upper bound (join) of σ_1 and σ_2
ι	The initial dataflow analysis information that is injected at the beginning of a function (the end for backwards analyses)
$\alpha_{ZI}(n)$	The abstraction function for the integer zero sub-lattice Maps an integer to a lattice element
$\alpha_{ZA}(\eta)$	The abstraction function for the integer zero sub-lattice Maps a program state η to a tuple lattice mapping each variable to an alement of the component lattice

$\{\ i \mid 0 \leq i \leq 5\}$	Set comprehension notation: denotes the set of all integers i such that the condition $0 \leq i \leq 5$ is true
$\emptyset$	The empty set
$s_1 \cup s_2$	The union of sets s_1 and s_2
$s_1 \cap s_2$	The intersection of sets s_1 and s_2
$s_1 - s_2$	The set difference of s_1 and s_2 (all elements in s_1 but not in s_2)
$s_1 \subseteq s_2$	s_1 is a subset of (or is equal to) s_2
$\mathcal{P}^S$	The powerset (set of all subsets) of S
$f^*_{DF}(\sigma, exp)$	Transitively applies the flow function f_{DF} to all subexpressions of *exp*

Below, we define the flow function $f_{\text{ZA}}(\sigma, [...]_k)$ for zero analysis (ZA). A flow function maps dataflow information σ from before a AST node $[...]_k$ to dataflow information afterwards. We define the flow function by cases, as might be done using pattern matching function definitions in the ML language. Think of the pattern matching as equivalent to an if statement testing whether the AST node matches each piece of syntax in turn–and the right hand side is used to compute the value of the flow function for the first syntax match.

To define separate cases for when a conditional evalutes the true and false cases, we just use tne notation f^T_{ZA} and f^F_{ZA}.

$$f_{\text{ZA}}(\sigma, [x]_k) = [t_k \mapsto \sigma(x)]\ \sigma$$

$$f_{\text{ZA}}(\sigma, [n]_k) = \text{if } n = 0 \text{ then } [t_k \mapsto Z]\ \sigma \text{ else } [t_k \mapsto NZ]\ \sigma$$

$$f_{\text{ZA}}(\sigma, [x := [...]_n]_k) = [x \mapsto \sigma(t_n)]\ \sigma$$

$$f^T_{\text{ZA}}(\sigma, [[x]_n = [0]_m]_k) = [x \mapsto Z]\ \sigma$$

$$f^F_{\text{ZA}}(\sigma, [[x]_n = [0]_m]_k) = [x \mapsto NZ]\ \sigma$$

$$f_{\text{ZA}}(\sigma, [[...]_n op [...]_m]_k) = [t_k \mapsto MZ]\ \sigma$$

$$f_{\text{ZA}}(\sigma, ...) = \sigma$$