Dataflow Frequency Analysis
CS701

Thomas Reps
[Based on notes taken by Anshul Purohit on November 3, 2015]

Abstract

This lecture continues the presentation of how to perform Dataflow Frequency Analysis [5] on the folded DAG structure generated via the Larus algorithm for collecting a whole-program path. The frequency-analysis technique can be applied to the class of bi-distributive dataflow-analysis problems. This lecture formalizes the technique of generating dataflow-frequency facts by traversing the DAG representation of a grammar that represents a whole-program path (without unfolding the DAG).

1 Whole Program Paths

The previous lecture explained Larus's algorithm to collect a "whole-program path" (WPP) [2]. The algorithm uses the Sequitur algorithm [3] to build a somewhat degenerate context-free grammar that accepts exactly one word, which corresponds to the path that was executed by an instrumented program. The terminal symbols of the context-free grammar are the acyclic Ball-Larus path fragments [1] of the program's control flow graph. The whole-program path is returned in the form of a dag, each of whose leaves is a path fragment.

Consider the control flow graph shown in Fig. 1(a) and the dag shown in Fig. 1(b), which is the dag created for the purpose of path-fragment numbering by the Ball-Larus algorithm. There are eight acyclic path fragments in the dag; we will refer to three of them, [s,1,2,4], [1,2,4], [1,3,4,5,6], as "a," "b," and "c," respectively. The Larus WPP algorithm returns a grammar whose one word

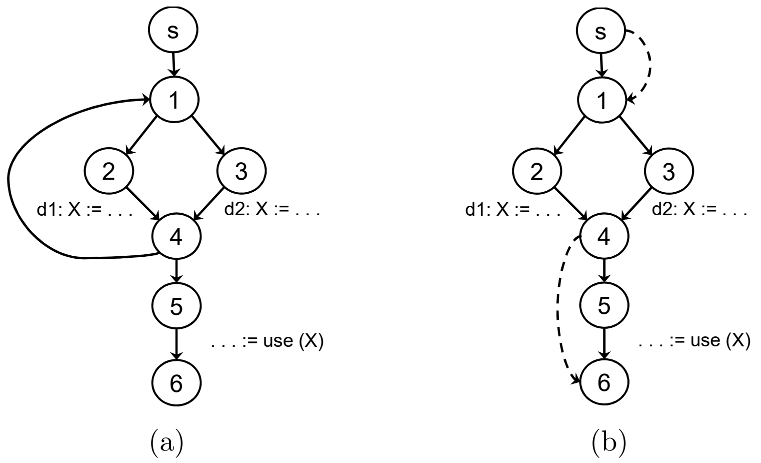

Figure 1: (a) Example control-flow graph. (b) dag created for the purpose of path-fragment numbering by the Ball-Larus algorithm.

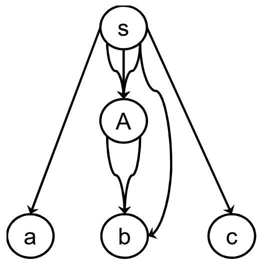

Figure 2: Example whole-program path.

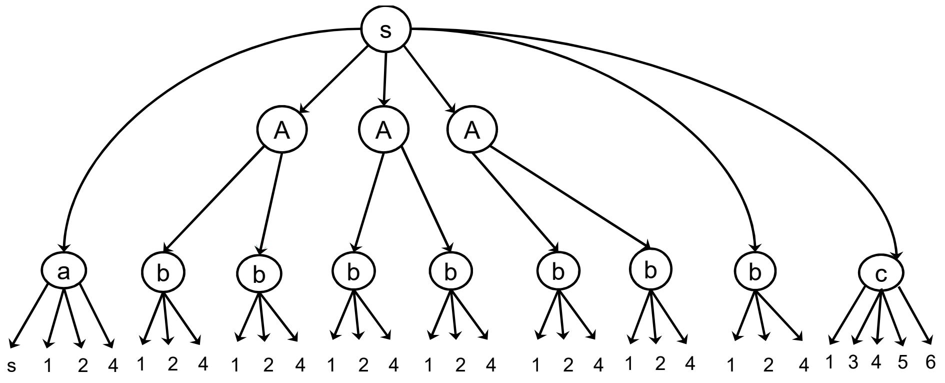

Figure 3: Expansion of the whole-program path from Fig. 2.

represents the trace of the program execution. For instance, consider the execution trace $\left[\text{s,1,2,4}\right]$ $\left[1, 2, 4\right]^7$ $\left[\text{1,3,4,5,6}\right]$. The WPP grammar for this trace is as follows:

$$\begin{aligned} S &\rightarrow aAAAbc \\ A &\rightarrow b\,b \end{aligned}$$

This grammar can be represented as a dag (see Fig. 2). It is instructive to consider the tree obtained by unfolding the dag (see Fig. 3).

2 Computing Dataflow-Fact Frequencies

The CFG in Fig. 1(a) is the CFG for a program that contains two definitions of variable X, at nodes 2 and 3, respectively. We will use the example of computing dataflow-fact frequencies for the most-recent definition-site of variable X. We will show how one can determine that (i) d1 reaches node 2 seven times; (ii) d2 never reaches node 2; (iii) d1 never reaches node 5; and (iv) d2 reaches node 5 once. The technique works for the class of bi-distributive dataflow-analysis problems [4]. This class of problem includes all gen/kill problems, as well as copy-constant propagation (which is not a gen/kill problem).

There is a dataflow transformer corresponding to every edge. The transformer can be represented as either a graph or a matrix (see Fig. 4 and the matrices shown below). For a bidirectional dataflow-analysis problem, the number of ones in each column of a dataflow-transformer matrix must be at most 1.

For edges $s \rightarrow 1$, $1 \rightarrow 2$, $1 \rightarrow 3$, $4 \rightarrow 5$, and $5 \rightarrow 6$, the transformer is merely the identity matrix. Fig. 4 shows the edge transformers in graphical form for all of the edges. The transformers

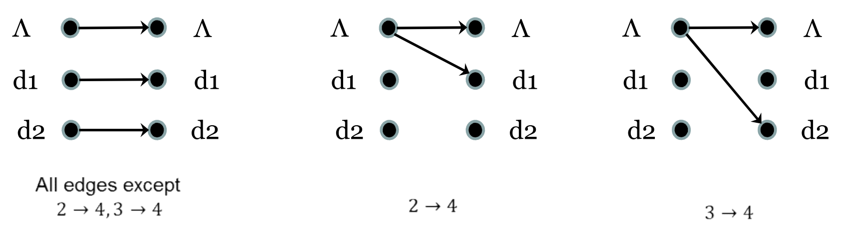

Figure 4: Dataflow transformers for propagating information about the most-recent definition-site of variable X.

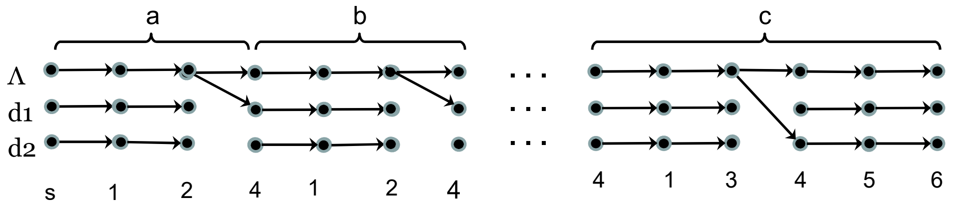

Figure 5: Concatenation of the dataflow transformers for propagating information about the most-recent definition-site of variable X.

can also be represented as the matrices M_1, M_2, and M_3, respectively.

$$M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad M_2 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad M_3 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The extend operator $\otimes$ in this problem is matrix multiplication. As noted above, the number of ones in each column of the dataflow-transformer matrix for a bidirectional dataflow-analysis problem must be at most 1. It is easy to argue that when two such matrices are multiplied, that property is preserved.

In the case of the graph representation, the $\otimes$ operation amounts to 2-step graph reachability. The edge transformer can also be written in functional form as

$$\lambda S.((N \times S) \cup \text{gen}),$$

where S is the characteristic vector of the input set, gen is a vector that represents reachability from Λ, and N is a matrix whose i^{th} row is a vector that indicates reachability from input fact d_i.

Using the definition of edge transformers, the number of times dataflow fact d_1 holds at a particular node can be calculated by performing reachability on the composition of the edge transformers. Fig. 5 shows the concatenation of the edge transformers for the path [s,1,2,4]$[1, 2, 4]^7$[1,3,4,5,6]. (Note that, for purposes of this computation, path-fragment b is considered to include the loop's back-edge $4 \to 1$.)

Matrices $M(i)$ can be defined for each of the terminal symbols i in the grammar. Terminal symbol b corresponds to the path [4,1,2,4]. Matrix $M(b)$ is merely the product the edge-transformer matrices for the edges in path-fragment b:

$$M(b) = M_{4,1} \times M_{1,2} \times M_{2,4}.$$

We also define *frequency-propagation matrices* for terminal-symbol/node pairs. For instance, for terminal symbol b and node i (where i may or may not be part of b), the frequency-propagation

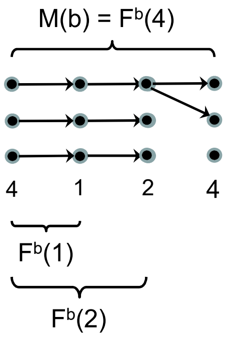

Figure 6: Partial products.

matrices are denoted by $F_b(i)$. $F_b(i)$ represents the frequency transformation from the beginning of a b symbol to just before node i in b. $F_b(1)$ is the edge transformer for the edge $4 \to 1$. $F_b(2)$ is the product of the matrices for edges $4 \to 1$ and $1 \to 2$. For all nodes not in the path represented by nonterminal b, the F_b matrix will contain all zeros, which represents the fact that we do not find these nodes in path-fragment b.

Using the definitions for the matrix transformers over path fragments and frequency matrices over terminal-symbols in the grammar, M and F matrices are defined for the non-terminal symbols of the WPP. (The F_S matrices, where S is the start symbol of the grammar, are used to calculate the final dataflow-frequency answers for the entire path.) Matrices are defined over the grammar symbols in a bottom-up fashion. In this case, $M(A)$ and $F_A(.)$ are defined in terms of $M(b)$ and $F_b(\cdot)$ according to the rule "$A \to bb$":

$$\begin{aligned} M(A) &= M(b) \times M(b) \\ F_A(i) &= F_b(i) + M(b) \times F_b(i) \end{aligned}$$

In particular, the two summands in $F_A(i)$ correspond to the two places in bb where node i may occur.

In general, the computation of $F_P(\cdot)$ for a nonterminal P resembles a prefix-sum computation over the symbols of the production with P on the left-hand side. For instance, in our example the production for S is "$S \to aAAAbc$," and thus $M(S)$ and $F_S(\cdot)$ are defined as follows:

$$\begin{aligned} M(S) &= M(a) \times M(A) \times M(A) \times M(A) \times M(b) \times M(c) \\ F_S(i) &= F_a(i) + M(a) \times F_A(i) + M(a) \times M(A) \times F_A(i) \\ &\quad + M(a) \times M(A) \times M(A) \times F_A(i) + M(a) \times M(A) \times M(A) \times M(A) \times F_b(i) \\ &\quad + M(a) \times M(A) \times M(A) \times M(A) \times M(b) \times F_c(i). \end{aligned}$$

Finally, the frequency vector for node i, denoted by Frequency(i), can be obtained from the first row of $F_S(i)$, in this case:

$$\text{Frequency}(i) = (1, 0, 0) \times F_S(i).$$

The first element of this vector is the frequency that Λ is reachable at node i from Λ at the beginning of the path (i.e., the beginning of execution). Because at *every* node of the path Λ is always reachable from Λ at the beginning of the path, the first element of Frequency(i) tells us how many times node i occurs in the path. In general, the j^{th} entry of Frequency(i) indicates the number of times that the j^{th} dataflow fact holds at node i. Thus, the frequency vector can also be used to compute the *fraction* of times a particular dataflow fact j holds at node i

$$\text{Fraction}(i, j) = \frac{\text{Frequency}(i)(j)}{\text{Frequency}(i)(1)}.$$

References

[1] T. Ball and J. Larus. Efficient path profiling. In *MICRO*, 1996.

[2] J.R. Larus. Whole program paths. In *Prog. Lang. Design and Impl.*, 1999.

[3] C.G. Nevill-Manning and I.H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. of Art. Intell. Research*, 7:67–82, 1997.

[4] G. Ramalingam. Data flow frequency analysis. In *Prog. Lang. Design and Impl.*, 1996.

[5] B. Scholz and E. Mehofer. Dataflow frequency analysis based on whole program paths. In *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2002.