Interprocedural Dataflow Analysis, Part II
CS701

Thomas Reps
[Based on notes taken by Cheng Su on October 20, 2015]

Abstract

This lecture continues the presentation of interprocedural dataflow analysis, describing the "functional approach" defined by Sharir and Pnueli [2]. We define the interface for their interprocedural dataflow-analysis framework, and show how the interface can be instantiated for so-called "gen/kill problems." We also begin to describe a second instantiation of the framework for so-called "interprocedural, finite, distributive, subset (IFDS) problems" [1], which can be analyzed via context-free-language reachability (CFL-reachability). In this lecture, we show how to use relations to represent distributive functions over a finite subset, which is the first step in reducing IFDS problems to CFL-reachability.

1 Review

Last lecture, we explained the path-problem framework as follows:

$$EAS[s,n] = \begin{cases} \top & \text{if } n = s \\ \bigoplus_{\langle m,n\rangle \in \text{Edges}} M^{\#}(\langle m,n\rangle)(EAS[s,m]) & \text{otherwise} \end{cases} \tag{1}$$

We pointed out that Eqn. (1) is not in exactly the right form for a path problem because it uses function application (i.e., $M^{\#}(\langle m,n\rangle)(\cdot)$) rather than the extend operation (i.e., $(\cdot) \bigotimes M^{\#}(\langle m,n\rangle)$). We fixed this anomaly, and generalized Eqn. (1) to a functional view of dataflow analysis by recasting Eqn. (1) using the following formulation:

$$EFAS[s,n] = \begin{cases} Id & \text{if } n = s \\ \bigoplus_{\langle m,n\rangle \in \text{Edges}} EFAS[s,m] \bigotimes M^{\#}(\langle m,n\rangle) & \text{otherwise} \end{cases} \tag{2}$$

We pointed out that Eqn. (2) is similar to the intraprocedural-propagation equation in Sharir and Pnueli's "functional approach" to intraprocedural dataflow analysis [2]. To support cross-procedure propagation of information, one more case needs to be considered, for procedure call. The equations in [2] are defined as follows:

$$\phi_{e,e} = Id \tag{3}$$

$$\phi_{e,n} = \bigoplus_{\langle m,n\rangle \in \text{Edges}} f_{m,n} \circ \phi_{e,m} \quad \text{if } n \notin \text{Ret} \tag{4}$$

$$\phi_{e,r} = \phi_{e',x'} \circ \phi_{e,c} \quad ((c,r) \in \text{CallRetPair}) \tag{5}$$

In Eqn. (3), e denotes an entry node of the interprocedural control flow graph. $\phi_{e,e}$ denotes the dataflow information propagated from the entry node to itself. In Eqn. (4), we calculate the

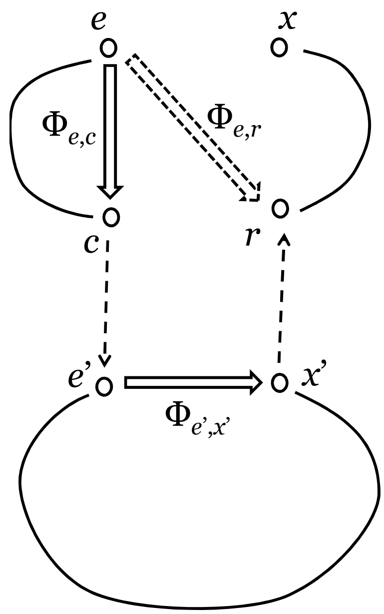

Figure 1: Illustration of the interprocedural equation from Phase I.

information for all the nodes except for return nodes. In Eqn. (5), c is the node to call procedure P. P has entry node e', and return node x'. r is the node after c. Eqn. (5) handles cross-propagation of information.

2 The Sharir and Pnueli Algorithm: Phase I

Sharir and Pnueli introduce the concept of *interprocedural valid paths* (IVP) for describing interprocedural information propagation. IVP can be viewed as the paths to satisfy a certain condition specified by a context-free language. Phase I calculates dataflow information based on the paths that satisfy the following context-free grammar:

$$\begin{array}{lcll} \mathit{matched} & ::= & \epsilon & \\ & | & \mathit{matched}\ \ e & \\ & | & \mathit{matched}\ \ (_i\ \ \mathit{matched}\ \ )_i & \quad \text{for } i \in \mathit{CallSites} \end{array} \tag{6}$$

For every call site i in every procedure p, we tag the call edge with "$(_i$", and the return edge with "$)_i$". The grammar above describes the paths whose calling stack returns to exactly the same level as it started, and never becomes shorter than the stack that we had when the path commenced.

To propagate dataflow information across procedures, Sharir and Pnueli gave Eqn. (5); however, it is a bit more precise—and better illustrates what is going on—to state it as follows:

$$\phi_{e,r} = \mathit{CallOut}_{x',r} \circ \phi_{e',x'} \circ \mathit{CallIn}_{c,e'} \circ \phi_{e,c}. \tag{7}$$

(Sharir and Pnueli make the assumption that the dataflow transformers for *CallIn* and *CallOut* edges are all *Id*.) It is also instructive to re-express Eqn. (7) with the extend operation:

$$\phi_{e,r} = \phi_{e,c} \otimes \mathit{CallIn}_{c,e'} \otimes \phi_{e',x'} \otimes \mathit{CallOut}_{x',r} \tag{8}$$

As shown in Fig. 1, to obtain the information $\phi_{e,r}$, we propagate information from entry node e to call site c in the caller, then from c to entry node e' of the callee, then from e' to return node x' of

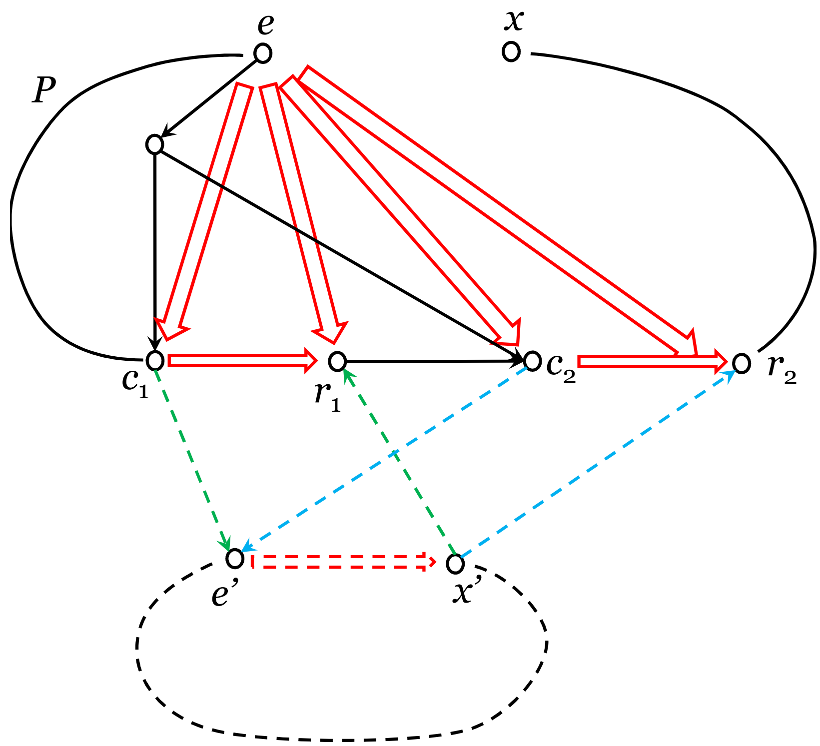

Figure 2: Propagation during Phase I when there are multiple calls to the same procedure.

the callee, and finally from x' to r. Note the similarity of Eqn. (8) to the form of grammar rule (6):

$$\underbrace{\phi_{e,r}}_{matched} = \underbrace{\phi_{e,c}}_{matched} \otimes \underbrace{CallIn_{c,e'}}_{(_i} \otimes \underbrace{\phi_{e',x'}}_{matched} \otimes \underbrace{CallOut_{x',r}}_{)_i}.$$

The situation where there are multiple calls to the same procedure is shown in Figure 2. (Phase I handles it automatically.)

Discussion. Fig. 2 can also be viewed as showing how *inter*procedural analysis is "reduced" to an *intra*procedural-analysis problem of analyzing separate procedures, once *summary functions* are introduced at call sites—such as call sites $\langle c_1, r_1 \rangle$ and $\langle c_2, r_2 \rangle$ in procedure P, as shown in Fig. 2. However, when a program has recursive procedures, the problem of finding the appropriate summary functions itself requires solving a dataflow-analysis problem. In a sense, the Sharir/Pnueli algorithm interleaves these two dataflow-analysis problems in a single analysis problem.

Recall the declarative specification of path problems, and what we have said previously about how the value defined by the declarative specification relates to the value obtained by solving the equational formulation—namely, they coincide as long as $\otimes$ distributes over $\oplus$. In this case, as long as $\otimes$ distributes over $\oplus$, the ϕ functions obtained by solving Eqns. (3), (4), and (5) correspond to the ψ functions specified by the following declaratively specified path problem, which is defined in terms of paths that respect the language $L(matched)$.

$$\psi_{e,n} = \bigoplus_{p \in matched(e,n)} apf_p,$$

where for $p = e_1, e_2, \cdots, e_k$,

$$apf_p = M^{\#}(e_1) \otimes M^{\#}(e_2) \otimes \cdots \otimes M^{\#}(e_k).$$

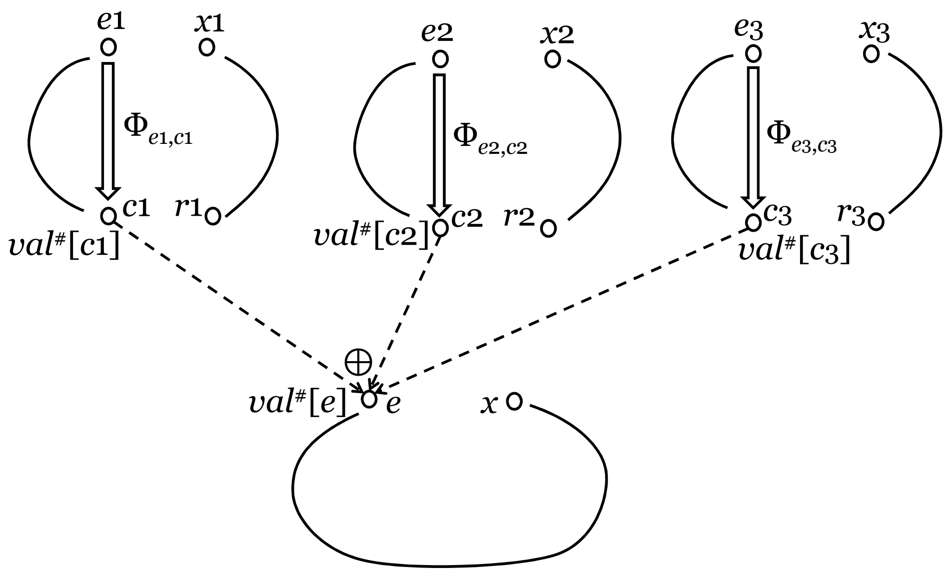

Figure 3: Propagation during Phase II when there are multiple calls to the same procedure.

3 The Sharir and Pnueli Algorithm: Phase II

Sharir and Pnueli describe their algorithm as a two-phase algorithm; however, I prefer to break up their second phase into two separate phases. Thus, part of Sharir and Pnueli's Phase II will be described as "Phase III" (see §4).

In both Phase II and Phase III, we need to handle interprocedural valid paths that end with the stack containing a sequence of called procedures that have not yet returned. Also, we are mainly interested in paths that begin at e_{main}, the entry node of the main procedure. The context-free grammar that we need to describe such paths is as follows:

$$\begin{array}{rcll} \mathit{unbalLeft} & ::= & \epsilon & \\ & | & \mathit{unbalLeft}\ \ \mathit{matched}\ \ (_i & \quad \text{for } i \in \mathit{CallSites} \\ & | & \mathit{unbalLeft}\ \ \mathit{matched} & \end{array}$$

Suppose that we want to calculate the dataflow information that should hold along all paths from entry node e_{main} to n, where n is a node in some procedure p (and p is not necessarily procedure *main*). Suppose that e is the entry node of p. In Phase III, we will use the fact that if we have the dataflow information $\mathrm{Val}^{\#}[e]$ for e, then we can obtain $\mathrm{Val}^{\#}[n]$ merely by applying $\phi_{e,n}$:

$$\mathrm{Val}^{\#}[n] = \phi_{e,n}(\mathrm{Val}^{\#}[e]).$$

This observation leaves us with the question, "How can we obtain $\mathrm{Val}^{\#}[e]$ (for each of the entry nodes in the program?" This problem can itself be formulated as a system of dataflow equations. The propagation of dataflow information from entry node e_{main} to entry node e can be expressed by the following system of equations:

$$\begin{array}{rcl} \mathrm{Val}^{\#}[e_{main}] & = & \top \\ \mathrm{Val}^{\#}[e] & = & \displaystyle\bigoplus_{\langle c_i, e\rangle \in \mathit{CallIn}} \phi_{e_j,c_i}(\mathrm{Val}^{\#}[e_j]) \end{array} \tag{9}$$

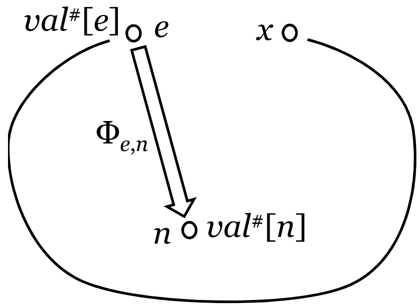

Figure 4: Intraprocedural propagation during Phase III.

where e_j is the entry node of the procedure that contains c_i, and *CallIn* is the set of all edges from a call node to an entry node. Note that the functions of the form ϕ_{e_j,c_i} were all calculated during Phase I, so all the necessary information is available at the start of Phase. A graphical depiction of these equations is shown in Fig. 3.

In the discussion of Phase I, we noted a similarity between Eqn. (8) and the form of grammar rule (6). For Phase II, there is an analogous similarity between Eqn. (9) and the second grammar rule that defines $L(\mathit{unbalLeft})$:

$$\mathit{unbalLeft} \quad ::= \quad \mathit{unbalLeft} \;\; \mathit{matched} \;\; (_i.$$

The similarity is concealed because (i) Sharir and Pnueli make the assumption that the dataflow transformers for *CallIn* edges are all *Id*, and (ii) the values propagated during Phase II are abstractions of *sets of concrete states*—not abstractions of *transition relations*—hence, the equations use function *application* rather than $\circ$ or $\otimes$. Thus, it is a bit more precise—and better illustrates what is going on—to state Eqn. (9) as follows:

$$\underbrace{\mathrm{Val}^{\#}[e]}_{\mathit{unbalLeft}} = \bigoplus_{\langle c_i,e\rangle \in \mathit{CallIn}} \underbrace{\mathit{CallIn}_{c_i,e}}_{(_i}(\underbrace{\phi_{e_j,c_i}}_{\mathit{matched}}(\underbrace{\mathrm{Val}^{\#}[e_j]}_{\mathit{unbalLeft}})), \tag{10}$$

where e_j is the entry node of the procedure that contains c_i. (The symbols on the right-hand side of the grammar rule appear in reverse order on the right-hand side of Eqn. (10) because functions that are called *later* as one traverses a path appear *earlier* in a chain of function applications.)

4 The Sharir and Pnueli Algorithm: Phase III

In Phase II, we calculated the information from e_{main} to e for each entry node e of the program. Now we show how to calculate the information from e to n. n is a node in the (intraprocedural) CFG for which e is the entry node. We merely use the result from Phase I, as shown in following equation:

$$\mathrm{Val}^{\#}[n] = \phi_{e,n}(\mathrm{Val}^{\#}[e])) \tag{11}$$

This equation is depicted graphically in Fig. 4.

The combination of Phases I, II, and III provides an algorithm for solving an interprocedural dataflow-analysis problem.

Discussion. It can be shown that as long as $\otimes$ distributes over $\oplus$, the values $\{\mathrm{Val}^{\#}[n] \mid n \in \mathrm{Nodes}\}$ obtained by solving the equations of Phases I, II, and III correspond exactly to the values $\{\mathrm{Val}^{\#}[n] \mid n \in \mathrm{Nodes}\}$ defined via the following declaratively specified path problem, which is defined in terms of paths that respect the language $L(\mathit{unbalLeft})$:

$$\mathrm{DVal}^{\#}[n] = \bigoplus_{p \in \mathit{unbalLeft}(e,n)} \mathit{apf}_p(\top),$$

where for $p = e_1, e_2, \cdots, e_k$,

$$\mathit{apf}_p = M^{\#}(e_1) \otimes M^{\#}(e_2) \otimes \cdots \otimes M^{\#}(e_k).$$

5 The Interface(s) for the Sharir/Pnueli Dataflow-Analysis Framework

There are two different interfaces that one needs to supply to be able to use the Sharir/Pnueli framework:

Phase I: $\mathit{Id}, \bigotimes, \bigoplus, M^{\#}, =, \sqsubseteq$.

Phases II and III: $\top, \oplus_{\mathrm{AStore}}$, function application, $=, \subseteq$.

One family of dataflow-analysis problems for which we can satisfy our obligations is the family of gen/kill problems. Recall that each gen/kill abstract transformer has the form $\lambda S.(S - \mathit{Kill}_p) \cup \mathit{Gen}_p$, where Kill_p and Gen_p are set-valued constants associated with some program point p. Each such function can be represented using two sets: $(\mathit{Kill}_p, \mathit{Gen}_p)$. For the Phase I interface, we have

$$\begin{array}{rl} \mathit{Id}: & (\phi, \phi) \\ \oplus: & (k_1 \cap k_2, g_1 \cup g_2) \\ \otimes: & (k_1 \cup k_2, (g_1 - k_2) \cup g_2) \end{array}$$

For the interface for Phases II and III, we have

$$\begin{array}{rl} \top: & \mathbb{U} \\ \oplus_{\mathrm{AStore}}: & \cup \\ \text{function application}: & (\lambda S.(S - \mathit{Kill}_p) \cup \mathit{Gen}_p)(T) = (T - \mathit{Kill}_p) \cup \mathit{Gen}_p) \end{array}$$

where $\mathbb{U}$ is the universe of elements that can go into *Kill* and *Gen* sets.

6 Relations to Represent Distributive Functions

We now begin to describe a second instantiation of the Sharir and Pnueli framework for so-called "interprocedural, finite, distributive, subset (IFDS) problems" [1].

Suppose that we have a function $f : 2^D \to 2^D$ that distributes over union—i.e., $f(S_1 \cup S_2) = f(S_1) \cup f(S_2)$. We can construct a relation $R \subseteq (D \cup \{\Lambda\}) \times (D \cup \{\Lambda\})$ to represent f as follows:

$$R_f = \{(\Lambda, \Lambda)\} \cup \{(\Lambda, y) \mid y \in f(\varnothing)\} \cup \{(x, y) \mid y \in f(\{x\}) \wedge y \notin f(\varnothing)\} \tag{12}$$

In essence, R_f tracks f's behavior "pointwise" (i.e., on the singleton sets), with the special symbol Λ representing f's behavior on the empty set.

Remark. The construction is sensible because distributivity means that f's behavior on some set S depends only on its behavior on subsets of S. We can keep breaking the subsets into smaller subsets until we are left with the singleton sets and the empty set. □

All gen/kill functions distribute over $\cup$, so let us consider a gen/kill problem as an example. Suppose that the function is

$$f \stackrel{\text{def}}{=} \lambda S.(S - \{d_4\}) \cup \{d_1, d_2\}. \tag{13}$$

The relation R_f that represents f is shown in Fig. 5. The graph of R_f illustrates a characteristic of all gen/kill functions:

- The pre-state Λ is connected to the post-state Λ.

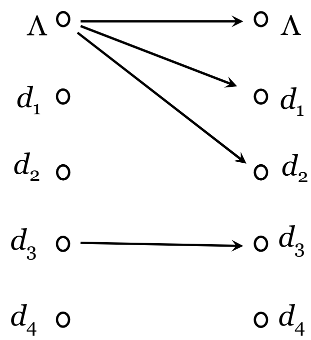

Figure 5: Representation relation for a gen-kill function.

- There are edges from the pre-state Λ to zero or more nodes $\{d_i \mid d_i \neq \Lambda\}$.
- For all $d_i \neq \Lambda$, either there is a gap from the pre-state d_i node to the post-state d_i node, or there is an edge from the pre-state d_i node to the post-state d_i node.
- There are no other kinds of edges.

In other words, if the representation relation R_f for a function f has been constructed according to Eqn. (12), you can immediate tell whether f is a gen/kill function by looking at the graph of R_f. In particular, if there is an edge from a pre-state node $d_i \neq \Lambda$ to a post-state node $d_j \neq \Lambda$ such that $i \neq j$, then f is *not* a gen/kill function.

One should note that the mapping from (syntactic) definitions of functions to representation relations is many-one. For instance, the representation relation shown in Fig. 5 could also be interpreted as the representation relation R_g for the function

$$g \stackrel{\text{def}}{=} \lambda S.(S - \{d_1, d_2, d_4\}) \cup \{d_1, d_2\}.$$

That is, the representation relation shown in Fig. 5 is both R_g and R_f for the function f defined in Eqn. (13). However, even though f and g are defined in terms of different kill sets, f and g both compute the same *mathematical function*. Two functions F_1 and F_2 are said to be *extensionally equal* iff, for all x, $F_1(x) = F_2(x)$. Thus, although the definitions of f and g are syntactically different, f and g are extensionally equal. In general, a given representation relation is a canonical form for the set of distributive functions over a finite set that are extensionally equal.

References

[1] T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Princ. of Prog. Lang.*, pages 49–61, 1995.

[2] M. Sharir and A. Pnueli. Two approaches to interprocedural data flow analysis. In *Program Flow Analysis: Theory and Applications*. Prentice-Hall, 1981.