CHAPTER 2

SYMBOLIC EVALUATION AND THE GLOBAL VALUE GRAPH

2.0 Summary.

As in Chapter 1, we assume a global flow model in which the expressions computed are specified, but the flow of control is indicated only by a directed graph whose nodes are blocks of assignment statements. We develop a direct (non-iterative) method for finding general symbolic values for expressions in the text of the program. Our method gives results similar to an iterative method due to Kildall[Ki] and a direct method due to Fong, Kam, and Ullman[FKU]. By means of a structure called a global value graph which compactly represents both symbolic values and the flow of these values through the program, we are able to obtain results that are as strong as either of these algorithms at a lower time and space cost, while retaining applicability to all flow graphs.

2.1 *Introduction*.

Let us review the basic definitions of the global flow model defined in Section 1.2. Let P be the program which we wish to analyze and improve. The flow of control through P is represented by the *control flow graph* $F = (N, A, s)$ where N is a set of blocks of assignment statements, A is a set of edges specifying possible flow of control immediately between blocks, and $s \in N$ is the *start block* from which all flow of control begins. A *control path* is a path in F. Let $\rightarrow$, $\overset{+}{\rightarrow}$, $\overset{*}{\rightarrow}$ denote respectively the immediate dominator relation, proper dominator relation, and the dominator ordering, which is a partial ordering.

Let $\{X, Y, Z, \ldots\}$ be the set of program variables, and let Σ be the set of program variables occurring within more than one block of N. For each $n \in N-\{s\}$ and program variable $X \in \Sigma$ we introduce the *input variable* $X^{\rightarrow n}$ to denote the value of X on *entry* to block n. Also, $X^{\rightarrow s}$ represents the value of program variable $X \in \Sigma$ on entry to the program P at the start block s; $X^{\rightarrow s}$ is considered to be a constant sign rather than an input variable. Let EXP be a set of expressions built from input variables and fixed sets of constant and k-adic function signs. For each program variable $X \in \Sigma$ and block $n \in N$ such that X is assigned to at n, let $X^{n\rightarrow}$ denote the expression in EXP for the value of X on *exit* from n in terms of constants and input variables at

n. $X^{n\rightarrow}$ is called the _output expression for X at n_. The _text expressions_ of P are the output expressions plus their subexpressions. Note that input variable $X^{\rightarrow n}$ is a text expression only if X occurs in the right hand side of an assignment statement of block n before X is assigned to. In this section and the next, it will be useful to assume that the text expressions include _all_ input variables; for each X ϵ Σ and block n ϵ N-{s} such that $X^{\rightarrow n}$ is not an input variable, add at n the dummy assignment X := X.

An _interpretation_ was defined in Section 1.2 to contain a universe U of values and mappings from contant signs to U, and from k-adic function signs to mappings from U^k to U. An expression in EXP is _reduced_ relative to an interpretation by repeatedly substituting constant signs for constant subexpressions.

For each α ϵ EXP, origin(α) is the earliest block n relative to the domination ordering $\overset{*}{\rightarrow}$ (with the start block s first) such that every block referred to in α is contained on all control paths from s to n. For each control path p from s and containing origin(α), EXEC(α,p) is intuitively the expression in EXP for the value of α relative to p. An expression α ϵ EXP _covers_ text expression t if

$$\text{EXEC}(t,p) = \text{EXEC}(\alpha,p)$$

for all control paths from s to origin(t). A _cover_ ψ is a mapping from text expressions to EXP such that ψ(t) covers t for each text expression t. We use origin to induce the partial ordering $\overset{*}{\rightarrow}$ of covers; for each pair of covers ψ and ψ', $\psi \overset{*}{\rightarrow} \psi'$ iff origin(ψ(t)) $\overset{*}{\rightarrow}$ origin(ψ'(t)) for all text expressions t.

In Chapter 1 we showed that the problem of computing covers minimal with respect to $\overset{*}{\rightarrow}$ over arithmetic domains is unsolvable; hence we consider a simple class of covers that might be computed by an algorithm due to Kildall. To construct this class of covers, Kildall's algorithm would first take a pass through the program and construct a mapping ψ_0 from text expressions to EXP; ψ_0 may not be a cover but has the property that for all text expressions t,

$$\text{EXEC}(\psi_0(t),p) = \text{EXEC}(t,p)$$

for some (rather than <u>all</u>) control paths p from s to origin(t). The algorithm would then iteratively compare possible covering expressions of input variables at particular blocks to the corresponding output expressions of preceding blocks, and propagate the results to predecessor blocks. More precisely, for any mapping ψ from text expressions to EXP, let $\Psi(\psi)$ be the mapping ψ' from text expressions to EXP such that for each input variable $X^{\rightarrow n}$,

$$\psi'(X^{\rightarrow n}) = \alpha \text{ if } \alpha = \psi(X^{m\rightarrow}) \text{ for all blocks m immediately preceding n in the control flow graph F,}$$
$$= X^{\rightarrow n}\text{, otherwise.}$$

and ψ'(t) is the reduced expression derived from text expression t after substituting $\psi'(X^{\rightarrow n})$ for each input variable $X^{\to n}$ occurring in t. Kildall's algorithm computes $\Psi^k(\psi_0)$ for $k = 1,2,\ldots$ until a fixed point of Ψ is obtained. Note that Ψ maps covers to covers; but Ψ need not be monotonic, i.e. for some cover ψ and text expression t, it may happen that $\Psi(\psi)(t) \overset{*}{\not\to} \psi(t)$.

<u>Theorem 2.1</u>. Each ψ which is a fixed point of Ψ is a cover, i.e. $\mathrm{EXEC}(\psi(t),p) = \mathrm{EXEC}(t,p)$ for all text expressions t and control paths p from s to the block where t is located.

<u>Proof</u> by construction. Let p be the shortest control path from s to a block n where there is located a text expression t such that

$$\mathrm{EXEC}(\psi(t),p) \neq \mathrm{EXEC}(t,p).$$

Thus t must contain an input variable $X^{\to n}$ such that

$$\mathrm{EXEC}(\psi(X^{\to n}),p) \neq \mathrm{EXEC}(X^{\to n},p).$$

Clearly, $\psi(X^{\to n}) \neq X^{\to n}$. Let m be the next to last block in p, so $p = p' \cdot (m,n)$. By definition of Ψ, $\psi(X^{\to n}) = \psi(X^{m\to})$. Since $\psi(X^{\to n})$ contains no input variables at n,

$$\begin{aligned}\mathrm{EXEC}(\psi(X^{\to n}),p) &= \mathrm{EXEC}(\psi(X^{\to n}),p')\\ &= \mathrm{EXEC}(\psi(X^{m\to}),p'), \text{ since } \psi(X^{\to n}) = \psi(X^{m\to}).\\ &= \mathrm{EXEC}(X^{m\to},p') \text{ by the induction hypothesis,}\\ &= \mathrm{EXEC}(X^{\to n},p) \text{ by definition of EXEC.} \quad \square\end{aligned}$$

In Section 2.2, we show that Ψ has a unique minimal fixed point ψ^*. We then show (Sections 2.3-2.7) that while the problem of finding minimal covers is hopeless, that of finding ψ^* is not only solvable but can be done efficiently. Thus we provide an efficient algorithm for finding the minimal cover among those of the type computed iteratively by Kildall's algorithm.

In fact the rest of this chapter is presented in a more general setting than is suggested above, so as to lay the foundation for related algorithms in Chapter 3 which deal with programs that operate on structured data. The overall plan is to introduce (in Section 2.2) a special class of graphs called _global value graphs_ which represent the flow of _values_ (rather than _control_) through the program P; and we define, for each global value graph GVG, a set Γ_{GVG} of approximate covers associated with it. Γ_{GVG} is in each case a finite semilattice which thus has a unique minimal element ψ_{GVG}, and which is efficiently calculated by the algorithm presented in Sections 2.3-2.7. As we show in Sections 2.2 and 2.8, for a particular choice of GVG, ψ_{GVG} is actually ψ^*, the minimal fixed point of the functional Ψ, so our general algorithm can be used to find ψ^* efficiently. (Indeed, the whole presentation could be made uniform by replacing the functional Ψ in an apppropriate way by a functional Ψ_{GVG} that depends on the particular global value graph; then ψ_{GVG}, the minimal element of Γ_{GVG}, would be in each case the minimal fixed point of Ψ_{GVG}. We have chosen not to do so, since only Ψ as defined here has any historical significance.)

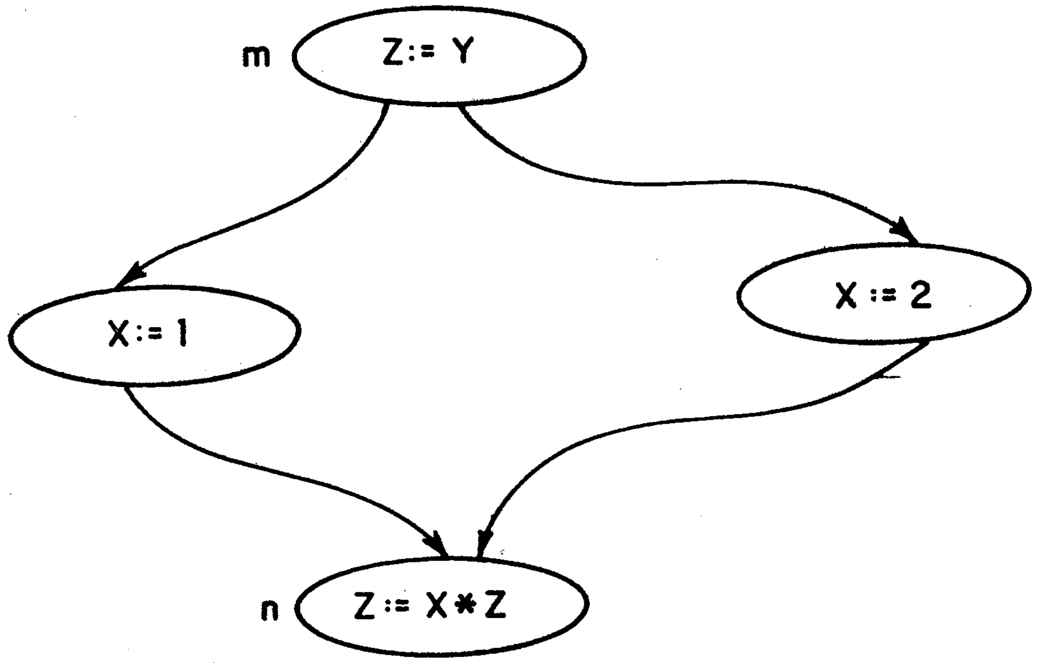
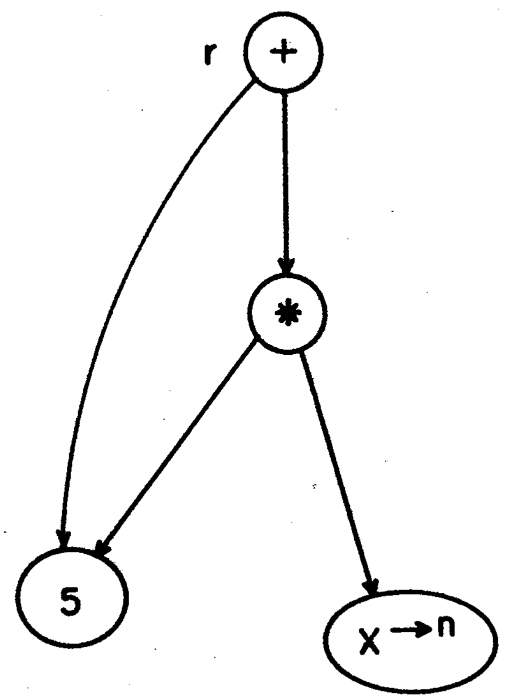


Figure 2.1. A fixed point of Ψ covers Z^{n+} with the expression $X^{\to n} * Y^{\to m}$.

2.2 Dags and Global Value Graphs.

A labeled dag D = (V, E, L) is a labeled, acyclic, oriented digraph with a node set V, an edge list E giving the order of edges departing from nodes, and a labeling L of the nodes in V. A rooted labeled dag (D,r) represents an expression α if α is the parenthesized listing of the labels of the subgraph of D rooted at r in topological order from r to the leaves and from left to right. (Where D is fixed, we simply say r represents α if (D,r) so represents α).

The dag D is minimal if each node $r \in V$ represents a distinct expression. Any expression or set of expressions may be represented, with no redundancy, by a minimal labeled dag D. In particular, we use the minimal dag D(n) to represent efficiently the set of text expressions located at block n. We have assumed that each block is reduced, so each node in D(n) corresponds to a unique text expression. Aho and Ullman[AU1] describe the use of dags for representing computations within blocks. Kildall[Ki] and Fong, Kam, and Ullman[FKU] have applied dags to various global flow problems.



<u>Figure 2.2</u>. (D,r) represents $(5+(5*X^{\rightarrow n}))$ (or more properly in prefix notation $(+\ 5\ (*\ 5\ X^{\rightarrow n})))$ where D is the above dag.

We now come to the central definition. To model the flow of values through a program P, we introduce a class of labeled digraphs called _global value graphs_ derived by combining the dags of all the blocks in N and adding a set of edges called _value edges_ which pair nodes labeled with input variables to other nodes. More precisely, a global value graph is a possibly cyclic, labeled, oriented digraph GVG = (V, E, L) such that:

(1) the node set V is the union of the node sets of the dags of N,

(2) E is an edge list containing (a) the edge list of each D(n) and (b) a set of pairs in V^2 (the _value edges_ of GVG) such that (i) the first node of each pair is labeled with an input variable and (ii) for each $v \in V$ labeled with an input variable $X^{\rightarrow n}$, and control path p from s to n, there is some value edge departing from v and entering a node located at a block in p and distinct from n.

(3) L is a labeling of V compatible with the labeling of each D(n).

Note that for each $v \in V$, if v represents a constant sign c then v is labeled with c and has no departing edges; if v represents a function application $(\theta\ t_1 \ldots t_k)$ then v is labeled with the k-adic function sign θ and $u_1, \ldots, u_k$ are the immediate successors of v in GVG representing $t_1, \ldots, t_k$ respectively; if v represents an input variable $X^{\rightarrow n}$ then v is labeled with $X^{\rightarrow n}$ and all the edges departing from v are value edges. For each node $v \in V$, let loc(v) be the block in N where the text expression which v represents is located.

A <u>value path</u> is a path in GVG traversing only nodes linked by value edges; a value path is <u>maximal</u> relative to a fixed beginning node if its last node has no departing value edges.

<u>Lemma 2.2.1</u> For any $v \in V$ labeled with an input variable and any control path p from the start block s to loc(v), there is a maximal value path q from v such that all the nodes in q have distinct loc values in p.

<u>Proof</u>. We consider (t) to be a trivial value path. Suppose we have constructed a value path $(v=u_1,\ldots,u_i)$ such that $loc(u_i)$, $loc(u_{i-1}),\ldots,loc(u_1)$ are distinct blocks occurring in this order in p. If u_i is not labeled with an input variable (and thus has no departing value edges) then $(t=u_1,\ldots,u_i)$ is a maximal value path. Otherwise, let p_i be the subpath of p from s to the first occurrence of block $loc(u_i)$ and let (u_i,u_{i+1}) be a value edge such that $loc(u_{i+1})$ occurs strictly before $loc(u_i)$ in p. Then $(t=u_1,\ldots,u_i,u_{i+1})$ is a value path and $loc(u_{i+1})$ is distinct from blocks $loc(u_1),\ldots,loc(u_i)$. The result thus follows from induction on the length of p. □

We assume here, as in Section 2.1, that the set of text expressions of each block $n \in N$ include all input variables at n. Let Γ_{GVG} be the set of mappings ψ from V to EXP such that for all $v \in V$,

(1) if L(v) is a constant sign c then $\psi(v) = c$, or

(2) if L(v) is a function sign θ and v has immediate successors $u_1,\ldots,u_k$ (in this order) then $\psi(v)$ is the reduced expression derived from $(\theta\ \psi(u_1)\ldots\psi(u_k))$, or

(3) if L(v) is an input variable then either (a) $\psi(v) = L(v)$ or (b) $\psi(v) = \psi(u)$ for all value edges (v,u) departing from v.

Note that for any node v satisfying (2), $\psi(v)$ is determined from the input variables occurring in the text expression which v represents. Hence any $\psi \in \Gamma_{GVG}$ is uniquely specified by the set of input variables satisfying case (3a), so Γ_{GVG} has at most $2^{|N||\Sigma|}$ elements.

<u>Lemma</u> <u>2.2.2</u>. For any $\psi \in \Gamma_{GVG}$ and $v \in V$, origin(ψ(v)) $\overset{*}{\rightarrow}$ loc(v).

<u>Proof</u> by contradiction. Suppose for some $v \in V$,

origin(ψ(v)) $\overset{*}{\not\rightarrow}$ loc(v).

Hence, there must be an input variable $X^{\rightarrow n}$ occurring in $\psi(v)$ such that n $\overset{*}{\not\rightarrow}$ loc(v), and so there is an n-avoiding path p from the start block s to loc(v). Also, there must exist some $u \in V$ labeled with an input variable and also located at block n, such that $\psi(u) = X^{\rightarrow n}$. By Lemma 2.2.1, we can construct a maximal value path $(u=u_1,\ldots,u_k)$ such that $loc(u_1),\ldots,loc(u_k)$ are distinct blocks in p. Let j be the maximal integer $\leq k$ such that $\psi(u_1) = \ldots = \psi(u_j)$. If $L(u_j)$

is an input variable, then $\psi(u_1) = L(u_j) = X^{\to n}$, so $loc(u_j) = n$ is contained in p, contradicting the assumption that p contains n. Otherwise, if $L(u_j)$ is not an input variable then neither is $\psi(v) = \psi(u_j)$, a contradiction with the assumption that $\psi(u) = X^{\to n}$. □

We shall show that Γ_{GVG} is a finite semilattice with ordering $\overset{*}{\to}$, and hence has a minimal element. Then we shall define a global value graph GVG* such that the minimal fixed point of Ψ, the functional defined in the last section, is the minimal element of Γ_{GVG^*}.

We define a partial mapping <u>min</u>: EXP^2 -> EXP such that for all $\alpha, \alpha' \in$ EXP,

α <u>min</u> $\alpha' = \alpha$ if origin(α) $\overset{+}{\to}$ origin(α')

$= \alpha'$ if origin(α') $\overset{+}{\to}$ origin(α)

or if origin(α) = origin(α') and

(i) if $\alpha = \alpha'$ then α <u>min</u> $\alpha' = \alpha = \alpha'$, or

(ii) if α is a constant sign and α' is a function application, then α <u>min</u> α' = α' <u>min</u> $\alpha = \alpha$, or

(iii) if α, α' are function applications $(\theta\ \alpha_1 \ldots \alpha_k)$, $(\theta\ \alpha'_1 \ldots \alpha'_k)$ respectively, and $\bar{\alpha}_i = \alpha_i$ <u>min</u> α'_i is defined for $i = 1, \ldots, k$ then α <u>min</u> $\alpha' = (\theta\ \bar{\alpha}_1 \ldots \bar{\alpha}_k)$,

and otherwise, α <u>min</u> α' is undefined.

We extend <u>min</u> to the partial mapping from pairs of elements of Γ_{GVG} to Γ_{GVG} defined thus: for $\psi, \psi' \in \Gamma_{GVG}$, if for all $v \in V$ $\psi(v)$ <u>min</u> $\psi'(v) = \bar{\psi}(v)$ is defined then ψ <u>min</u> ψ'

$= \overline{\Psi}$ and otherwise ψ min ψ' is undefined.

Theorem 2.2.1. Γ_{GVG} is a semilattice with ordering $\overset{*}{\rightarrow}$.

Proof It is sufficient to show min is well defined over Γ_{GVG}. We proceed by induction. Suppose for $\psi, \psi' \in \Gamma_{GVG}$ and some α in the domain of Ψ, $\psi(u)$ min $\psi'(u)$ is defined for all $u \in V$ such that $\psi(u)$ is a proper subexpression of α. Consider some text expression v such that $\psi(v) = \alpha$. By Lemma 2.2.2, both origin($\psi(v)$) and origin($\psi'(v)$) are contained on all control paths from the start block s to loc(v), so we may assume without loss of generality that origin($\psi(v)$) $\overset{*}{\rightarrow}$ origin($\psi'(v)$). Observe that $\psi(v)$ min $\psi'(v)$ = $\psi(v)$ if origin($\psi(v)$) $\overset{+}{\rightarrow}$ origin($\psi'(v)$) so we further assume that origin($\psi(v)$) = origin($\psi'(v)$).

Case 1. If L(v) is a constant sign c then $\psi(v) = \psi'(v) = c$ so $\psi(v)$ min $\psi'(v) = c$.

Case 2. Suppose L(v) is a function sign θ and v has immediate successors $u_1,\ldots,u_k$. By the induction hypothesis $\alpha_i' = \psi(u_i)$ min $\psi'(u_i)$ is defined for $i = 1,\ldots,k$. Hence $\psi(v)$ min $\psi'(v)$ is the reduced expression derived from (θ $\alpha_1' \ldots \alpha_k'$).

Case 3. Otherwise, suppose L(v) is an input variable. Let p be a control path from the start block s to loc(v). By Lemma 2.2.1, we can construct a maximal a value path $(v=u_1,\ldots,u_k)$ such that for $i = 1,\ldots,k$ each loc(u_i) is contained in p. Let j be the maximal integer such that $\psi(u_1)=\ldots=\psi(u_j)$.

Case 3a. If $\psi'(v) = \psi(u_1) = \ldots = \psi(u_i) \neq \psi'(u_{i+1})$ for some i, $1 \leq i < j$, then by the definition of Γ_{GVG}, $\psi(v) = \psi'(u_i) = L(u_i)$. Hence $\text{origin}(\psi'(v)) = n_i \neq n_j = \text{origin}(\psi(v))$, contradicting our assumption that $\text{origin}(\psi'(v)) = \text{origin}(\psi(v))$.

Case 3b. Otherwise, suppose $\psi'(v) = \psi'(u_1) = \ldots = \psi'(u_j)$ so we have $\psi(v) = \psi(u_j)$ and $\psi'(v) = \psi'(u_j)$. Applying Cases 1 and 2, $\psi(v)$ min $\psi'(v) = \psi(u_j)$ min $\psi'(u_j)$ is defined if $L(u_j)$ is either a constant sign or function application, so we assume $L(u_j)$ is an input variable. Since j is maximal, $\psi(v) = \psi(u_j) = L(u_j)$. If $\psi'(v) = \psi'(u_j) = L(u_j)$ then $\psi(v)$ min $\psi'(v) = L(u_j)$. Otherwise, suppose $\psi'(u_j) \neq L(u_j)$. For each value edge (u_j, v'), by the definition of Γ_{GVG}, $\psi'(u_j) = \psi'(v')$ and by Lemma 2.2.2, $\text{origin}(\psi'(v')) \overset{*}{\rightarrow} \text{loc}(v')$. Hence $\text{origin}(\psi'(v)) = \text{origin}(\psi'(u_j))$ is distinct from $\text{origin}(\psi(v))$, contradicting our assumption that $\text{origin}(\psi'(v)) = \text{origin}(\psi(v))$. □

Theorem 2.2.1 immediately implies that:

Corollary 2.2. Γ_{GVG} has an unique minimal element min Γ_{GVG}.

Now we shall define a special global value graph such that ψ^*, the minimal fixed point of Ψ, the functional defined in Section 2.1, is the minimal element of Γ applied to this graph. Again we assume that the text expressions include all the input variables, and add dummy assignments to satisfy this assumption. Let GVG^* be the global value graph containing the value edges $\{(v,u) \mid v$ represents input variable $X^{\rightarrow n}$ and u represents the output expression $X^{m\rightarrow}$ for each program variable $X \in \Sigma$ and edge $(m,n) \in A$ of the control flow graph $F\}$.

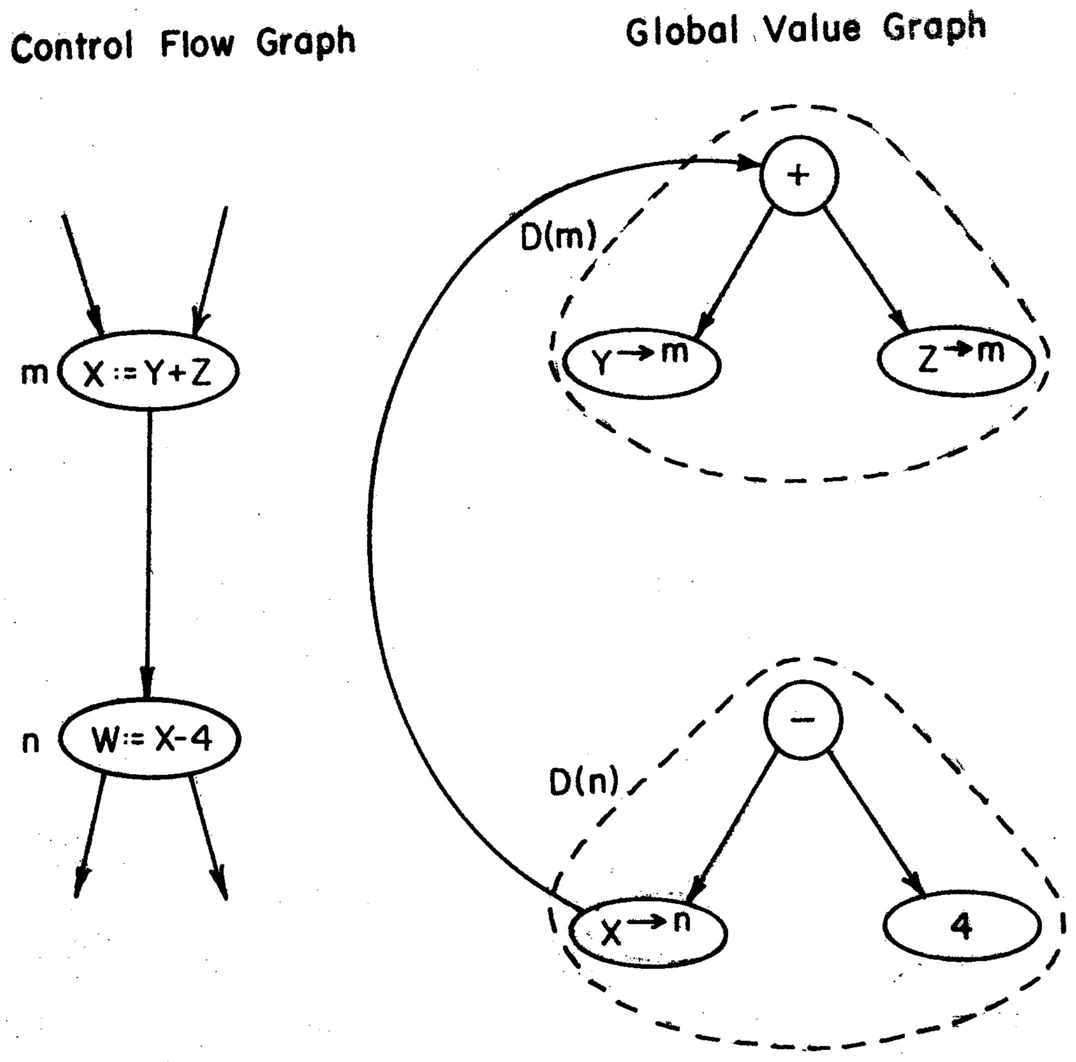


Figure 2.3. The global value graph GVG*.

We have shown that Γ_{GVG^*} is a finite semilattice and hence has a minimal element; we now show that this minimal element is the unique minimal fixed point of Ψ.

Theorem 2.2.2 ψ^*, the minimal fixed point of Ψ, is identical to $\hat{\psi}$, the unique minimal element of Γ_{GVG^*}.

Proof. Observe that any fixed point of Ψ is an element of Γ_{GVG^*}. By Corollary 2.2, Γ_{GVG^*} has a unique minimal element $\hat{\psi} = \underline{\min}\ \Gamma_{GVG^*}$. Suppose $\hat{\psi}$ is not a fixed point of Ψ. Observe that since $\hat{\psi} \in \Gamma_{GVG^*}$, for each input variable $X^{\rightarrow n}$, if $\psi(X^{\rightarrow n}) \neq X^{\rightarrow n}$ then $\Psi(\hat{\psi})(X^{\rightarrow n}) = \hat{\psi}(X^{\rightarrow n})$. Hence there is an input variable $X^{\rightarrow n}$ such that $\hat{\psi}(X^{\rightarrow n}) = X^{\rightarrow n}$ but $\Psi(\hat{\psi})(X^{\rightarrow n}) = \alpha$ where $\alpha = \hat{\psi}(X^{m\rightarrow})$ for all blocks m immediately preceding block n in the control flow graph F.

We are going to construct a mapping $\psi \in \Gamma_{GVG^*}$ distinct from $\hat{\psi}$ such that $\psi \overset{*}{\rightarrow} \hat{\psi}$, which will contradict our assumption that $\hat{\psi}$ is the minimal element of Γ_{GVG^*}. For each text expression t, let $\psi(t)$ be derived from $\hat{\psi}(t)$ by substituting α for each occurrence of $X^{\rightarrow n}$, and then reducing the resulting expression. We now show $\psi \in \Gamma_{GVG^*}$. Consider any input variable $Y^{\rightarrow n'}$.

Case a. Suppose $\hat{\psi}(Y^{\rightarrow n'}) = Y^{\rightarrow n'}$. If $Y^{\rightarrow n'} \neq X^{\rightarrow n}$ then $\psi(Y^{\rightarrow n'}) = Y^{\rightarrow n'}$. Otherwise, if $Y^{\rightarrow n'} = X^{n\rightarrow}$ then for each block m immediately preceding block n' = n, $\psi(Y^{\rightarrow n'}) = \hat{\psi}(Y^{m\rightarrow}) = \alpha$, and since $X^{\rightarrow n}$ is not contained in α, $\psi(Y^{\rightarrow n'}) = \psi(Y^{m\rightarrow}) = \alpha$.

Case b. If $\hat{\psi}(Y^{\rightarrow n'}) \neq Y^{\rightarrow n}$ then for each block m immediately preceding n' in F, $\hat{\psi}(Y^{\rightarrow n'}) = \hat{\psi}(Y^{m\rightarrow})$ so $\psi(Y^{\rightarrow n'}) = \psi(Y^{m\rightarrow})$.

Thus $\psi \in \Gamma_{GVG}^{*}$. For each block m immediately preceding n in F, $\alpha = \psi(X^{\rightarrow n}) = \psi(X^{m\rightarrow})$ so

$$\begin{aligned} \text{origin}(\psi(X^{\rightarrow n})) &= \text{origin}(\psi(X^{m\rightarrow})) \\ &\overset{*}{\rightarrow} \text{loc}(X^{m\rightarrow}), \text{ by Lemma 2.2.2} \\ &= m, \end{aligned}$$

and hence $\text{origin}(\psi(X^{\rightarrow n})) \overset{+}{\rightarrow} n = \text{origin}(\hat{\psi}(X^{\rightarrow n}))$. This implies that $\hat{\psi}$ is not the minimal element of Γ_{GVG}^{*}, a contradiction.

□

2.3 Propagation of Constants

Let ψ be a minimal element of Γ_{GVG} where GVG is an arbitrary global value graph (V, E, L). We wish to compute a new labeling L' of V such that for each $v \in V$, if $\psi(v)$ is a constant sign then L'(v) = c and otherwise L'(v) = L(v). Nodes thus relabeled with constant signs may be discovered by propagating possible constants through GVG, starting from nodes originally labeled with constant signs, and then testing for conflicts. This leads to an algorithm for constant propagation with time cost linear in the size of the GVG.

Recall that a <u>spanning tree</u> of the control flow graph F = (N, A, s) is a tree rooted at s, with node set N, and edge set contained in A. A <u>preordering</u> of a tree orders fathers before sons. Let < be a preordering of some spanning tree of F. We construct an acyclic subgraph of GVG by deleting value edges which are oriented between nodes in V whose loc values are compatable with <. More formally, let $E_<$ be the set of all value edges (v,u) such that loc(v) < loc(v). Observe that $(V, E-E_<)$ is acyclic. We shall propagate constants in a topological order of $(V, E-E_<)$, from leaves to roots.

Our algorithm for computing the new labeling L' is given below.

<u>Algorithm</u> <u>2A</u>.

<u>INPUT</u> GVG = (V, E, L), F.

<u>OUTPUT</u> L'.

```
begin
  declare L' := array of length |V|;
  Let < be a preordering of a spanning tree of F;
  Q := E< := the empty set {};
  for all value edges (v,u) ε E such that loc(v) < loc(u)
      do add (v,u) to E<;
  comment propagate constants;
 L0:for each v ε V in topological order of (V, E-E<)
   from leaves to roots do
     if L(v) is a constant sign c then L1: L'(v) := c;
     else if L(v) is a k-adic function sign θ,
        u1,...,uk are the immediate successors of v in
       GVG, and (θ L'(u1)...L'(uk)) reduces to a
       constant sign c then L2: L'(v) := c;
       else if L(v) is an input variable and there
         is a constant sign c such that L'(u) = c
         for all value edges (v,u) departing from v
         then L3: L'(v) := c;
       else begin add v to Q;L'(v):=L(v) end;
     end;
   comment test for conflicts;
 L4:for each v ε V labeled with an input variable do
     if v has a departing value edge (v,u) such that
     L'(v)≠L'(u) then add v to Q;
   till Q = the empty set {} do
     begin
       delete some node v from Q;
       if L'(v) is a constant sign then
     L5: begin
           L'(v) := L(v);
           add all immediate predecessors of v in GVG to Q;
         end;
     end;
 end.
```

Lemma 2.3.1. If $\psi(v)$ is a constant sign then $L'(v)$ is set to $\psi(v)$ at L1, L2, or L3.

Proof, by induction on the topological order of $(V, E-E<)$.

Basis step. Suppose v is a leaf of $(V, E-E<)$. Then $L(v)$ is a constant sign and so $L'(v)$ is set to $L(v) = \psi(v)$ at L1.

Induction step. Suppose v is in the interior of $(V, E-E<)$ and $L'(u)$ has been set to $\psi(u)$ for all u occurring before v in the topological order where $\psi(u)$ is a constant sign. Then v represents either a function application or an input variable.

Case 1. Suppose $L(v)$ is a k-adic function sign θ and $u_1,\ldots,u_k$ are the immediate successors of v in $(V, E-E<)$. If $\psi(v)$ is a constant sign c then by definition of Γ, $\psi(u_1),\ldots,\psi(u_k)$ are constant signs $c_1,\ldots,c_k$ respectively and $(\theta\ c_1\ldots c_k)$ reduces to c. By the induction hypothesis $L'(u_1),\ldots,L'(u_k)$ have been previously set to $c_1,\ldots,c_k$ and so $L'(v)$ is set to $\psi(v) = c$ at L2.

Case 2. Otherwise, $L(v)$ is an input variable X^{+n}. If $\psi(v)$ is a constant sign c then $\psi(v) \neq X^{+n}$ so by definition of Γ_{GVG}, $c = \psi(u)$ for all value edges (v,u) departing from v. By the induction hypothesis, $L'(u)$ has been set to $c = \psi(u)$ for each value edge $(v,u) \in E-E<$. Now we must show v has some departing value edge $(v,u) \in E-E<$. Let T be the spanning tree of F with preorder <. Consider the path p in T from the start block s to n. By definition of GVG, there is a value edge (v,u) such that $loc(u)$ is distinct from n and is contained in p. Hence $(v,u) \in E-E_<$ and L(v) is set to c at L3. $\square$

Let $\bar{Q}$ be the value of Q just after L4. Then $v \in V$ is eventually added to Q and L'(v) reset to L(v) iff some element of $\bar{Q}$ is reachable in GVG from v. If $v \in V$ is labeled by L' with a constant sign at L4, then we show

Lemma 2.3.2. $\psi(v)$ is not a constant sign iff some element of $\bar{Q}$ is reachable in GVG from v.

Proof. IF. Suppose $\psi(v)$ is not a constant, but no element of $\bar{Q}$ is reachable from v. Then let $\bar{\Psi}$ be the mapping from V to EXP such that for each $u \in V$, $\bar{\Psi}(u)$ is the reduced expression derived from $\psi(u)$ after substituting $\psi(w)$ for each input variable represented by a node w (i.e. w is the unique node labeled with that input variable) from which an element of $\bar{Q}$ is reachable. Then $\bar{\Psi} \in \Gamma_{GVG}$ but origin($\bar{\Psi}(v)$) = $s \overset{+}{\to}$ origin($\psi(v)$), contradicting the assumption that ψ is the minimal element of Γ_{GVG}.

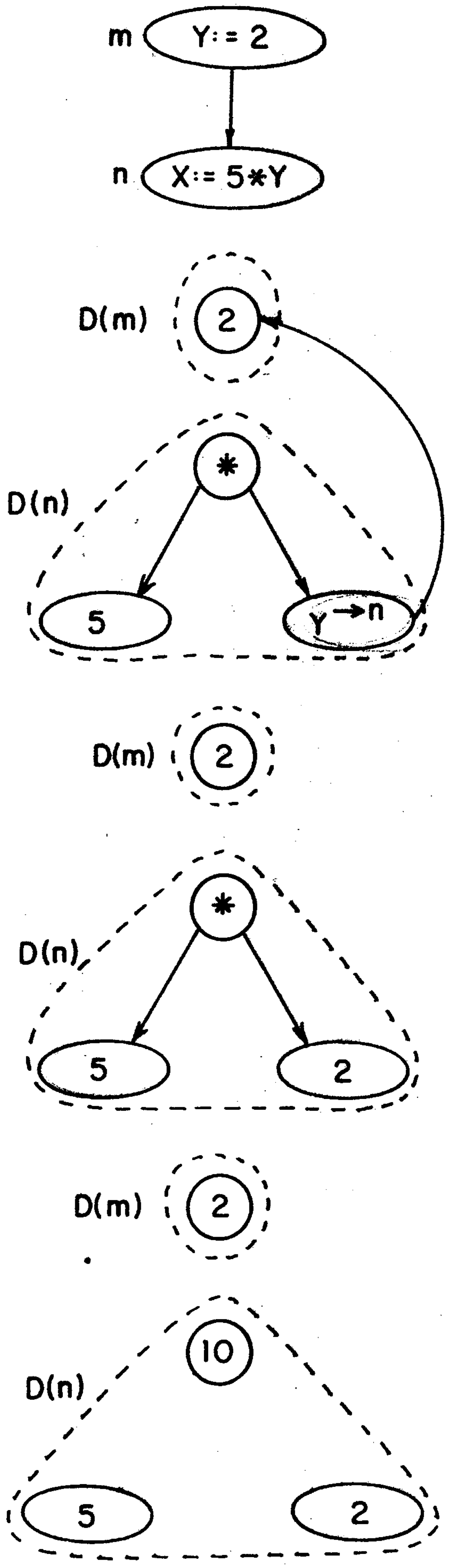
ONLY IF. Suppose some element of $\bar{Q}$ is reachable from v in GVG. Clearly if $v \in \bar{Q}$, then $\psi(v)$ is not a constant sign. Assume for some $k > 0$, if there is a path of length less than k in GVG from some $u \in V$ to an element of $\bar{Q}$, then $\psi(u)$ is not a constant sign. Suppose there is a path $(v=w_0,w_1,\ldots,w_k)$ of length k from v to $w_k \in \bar{Q}$. If $k = 1$, then $w_1 \in \bar{Q}$, and otherwise if $k > 1$, then $(w_1,\ldots,w_k)$ is a path of length k-1. By the induction hypothesis, $\psi(w_1)$ is not a constant sign. But $(v,w_1) \in E$ and by the definition of Γ_{GVG}, $\psi(v)$ is not a constant sign. □

Theorem 2.3. Algorithm 2A is correct and has time cost linear in the size of the GVG.

Proof. The correctness of Algorithm 2A follows directly from Lemmas 2.3.1 and 2.3.2.

In addition we must show Algorithm 2A has time cost linear in $|V|+|E|$. The initialization costs time linear in $|V|$. The preordering $<$ may be computed in time linear in $|N|+|A|$ by the depth first search algorithm of [T1]. The time to process each $v \in V$ at steps L0 and L4 is $O(1+\text{outdegree}(v))$. Step L5 can be reached at most $|V|$ times and the time cost to process each node v at step L5 is $O(1+\text{indegree}(v))$. Thus, the total time cost is linear in $|V|+|E|$. □

<u>Figure 2.4</u>. A simple example of constant propagation through the global value graph.



In some cases, we may improve the power of Algorithm 2A for particular interpretations by applying algebraic identities to reduce expressions in EXP more often to constant symbols. For example, in the arithmetic domain we can use the fact that 0 is the identity element under integer multiplication to modify Algorithm 2A so that if node v is labeled by L with the multiplication sign and a successor of v in GVG is covered by 0, then at step L3 we may set L'(v) to the constant sign corresponding to 0.

From the new labeling L' and GVG = (V, E L), we construct a _reduced_ _global_ _value_ _graph_ GVG' = (V, E', L') with labeling L' and with edge set E' derived from E by deleting all edges departing from nodes labeled by L' with constant signs. This corresponds to substituting constant signs for constant text expressions in the program P. We assume throughout the next three sections that GVG is so reduced.

2.4 A Partial Characterization of ψ, the Minimal Element of Γ_{GVG}

Let GVG = (V, E, L) be a reduced global value graph as constructed by Algorithm 2A of the last section and let ψ be the minimal element of Γ_{GVG}. Let $\hat{V}$ be the set of nodes in V representing constant signs and function applications (i.e. nodes labeled with constant and function signs). Observe that Γ_{GVG} characterizes exactly the values of any such ψ over nodes in $\hat{V}$ in terms of the values of ψ over the nodes in $V-\hat{V}$, i.e. in terms of the nodes labeled with input variables. The following Theorem characterizes ψ over $V-\hat{V}$ in terms of ψ over $\hat{V}$.

We require first a few additional definitions. Recall that a value path is a path p in GVG traversing only nodes linked by value edges and is maximal relative to a fixed beginning node if the last node of p has no departing value edges. For any node $v \in V$ labeled with an input variable, let H(v) be the set of nodes in V lying at the end of maximal value paths from v. Note that H(v) is a subset of $\hat{V}$. Call two paths almost disjoint if they have exactly one node in common.

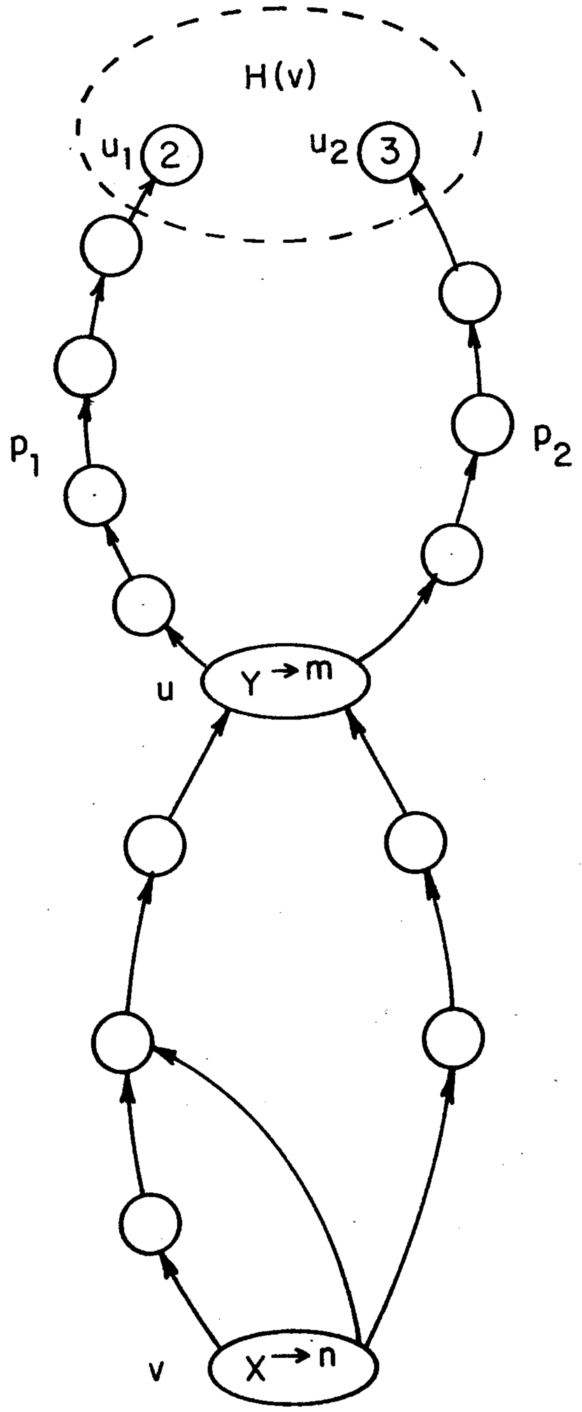


Figure 2.5. Case (b) of Theorem 2.4: all maximal value paths from v contain u and p_1, p_2 are almost disjoint maximal value paths from u to $u_1, u_2 \in H(v)$.

<u>Theorem</u> <u>2.4</u>. If v is labeled with an input variable, then either

(a) $\psi(v) = \psi(u)$ for all $u \in H(v)$, or

(b) $\psi(v) = L(u)$, where u is the unique node such that

(i) u lies on all maximal value paths from v but

(ii) there are almost disjoint maximal value paths from u to nodes $u_1, u_2 \in H(v)$ such that $\psi(u_1) \neq \psi(u_2)$.

<u>Proof</u>. Suppose $\psi(v)$ is <u>not</u> an input variable, so there exists a maximal value path p from v to some $u_1 \in H(v)$ such that $\psi(v) = \psi(u_1)$. Assume there exists another maximal value path p' from v to some $u_2 \in H(v)$ such that $\psi(v) \neq \psi(u_2)$. Let z be the first element of p' such that $\psi(z) \neq \psi(u)$ and let z' be the immediate predecessor of z in p', so $\psi(z') = \psi(v)$. Then by definition of Γ_{GVG}, $\psi(v) = \psi(z') = L(z')$ is an input variable, contradiction.

Suppose $\psi(v)$ is an input variable, so $\psi(v) = L(u)$ for some $u \in V$. For any maximal value path p from v, let z be the first element of p such that $\psi(z) \neq L(u)$ and let z' be the immediate predecessor of z in p. Then by definition of Γ_{GVG}, $\psi(z') = L(z') = L(u)$ so $z' = u$ is contained on p. Now suppose that there is a node $w \in V$ distinct from u and contained on all maximal value paths from u. Let loc map from nodes in V to the respective blocks in the control from graph where they are located.

Consider any control path q from the start block s to block loc(u). By Lemma 2.2.1, we can construct a maximal value path $(u=w_1,\ldots,w_k)$ such that $\mathrm{loc}(w_1),\ldots,\mathrm{loc}(w_k)$ are distinct blocks in q. Hence, $\mathrm{loc}(w) \overset{+}{\rightarrow} \mathrm{loc}(u)$.

Let ψ' be the mapping from V to EXP such that for all $v' \in V$, $\psi'(v')$ is derived from $\psi(v')$ by substituting L(w) for each input variable labeling a node from which all maximal value paths contain w. Then $\psi' \in \Gamma_{GVG}$. But $\mathrm{origin}(\psi'(v)) = \mathrm{loc}(w) \overset{+}{\rightarrow} \mathrm{loc}(u) = \mathrm{origin}(\psi(v))$, contradicting our assumption that ψ is minimal over Γ_{GVG}. □

Theorem 2.4 suggests a procedure for calculating ψ, but there is an implicit circularity since the calculation (using Theorem 2.4) of $\psi(v)$ for $v \in V-\hat{V}$ requires the determination (using the definition of Γ_{GVG}) of $\psi(u)$ for $u \in H(v)$; but since $u \in \hat{V}$, the calculation of $\psi(u)$ may require the determination of $\psi(w)$ for some other $w \in V-\hat{V}$. The way out is by the rank decomposition discussed in the next section. There will remain the problem of finding almost disjoint paths, which we consider in Section 2.5. This allows us to apply Theorem 2.4 without circularity.

2.5 Rank Decomposition of a Reduced GVG

This section describes a decomposition of the nodes of a reduced GVG = (V, E, L) into sets for which we may completely characterize the minimal $\psi \in \Gamma_{GVG}$. This leads to an algorithm for the construction of ψ.

Fong, Kam, and Ullman[FKU] describe the rank decomposition of a dag; this provides a topological ordering of a dag from leaves to roots over which the dag may be efficiently reduced. Here we generalize the rank decomposition to a possibly cyclic GVG; this provides us a method of partitioning V into sets of text expressions over which ψ may have the same value; it also allows us to apply Theorem 2.4 without circularity, characterizing completely the minimal $\psi \in \Gamma_{GVG}$. In Section 2.7 we apply the rank decomposition to implement our direct method for symbolic evaluation.

The rank of a node $v \in V$ is defined:

rank(v) = 0 if v is labeled with a constant sign

= 1 + MAX{rank(u) | $(v,u) \in E$} for v labeled with a function sign

= MIN{rank(u) | $u \in H(v)$} for v labeled with an input variable.

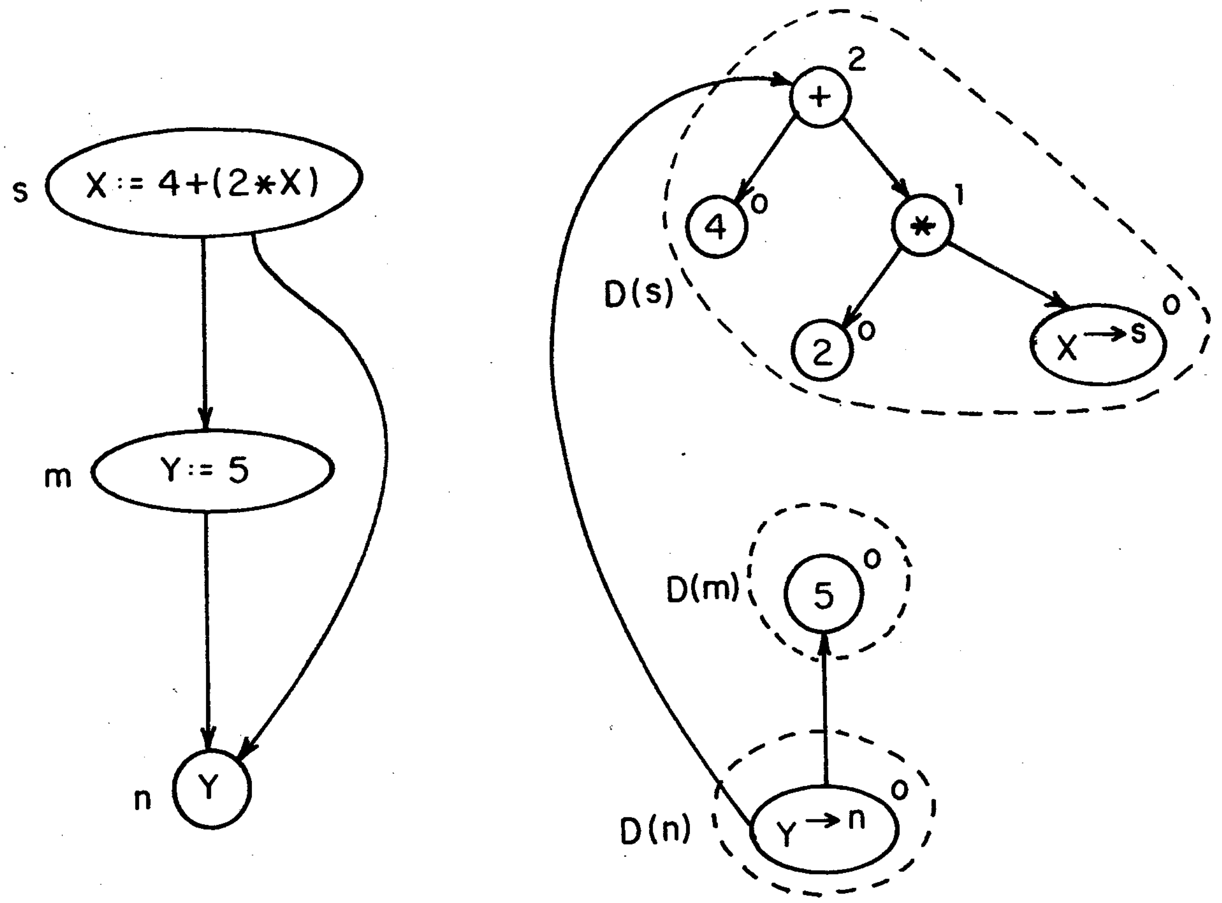


Figure 2.6. Rank decomposition of a global value graph. (The integer on the upper right hand side of each node is its rank.)

Observe that in the very simple case where P contains only a single block of code, (i.e. the start block s) then GVG consists of the dag D(s). Hence the rank of a node $v \in V$ is the length of a maximal path from v to a leaf of the dag D(s); inducing a topological ordering of the dag D(s) from leaves to roots.

Lemma 2.5. $\psi(v) = \psi(v')$ implies rank(v) = rank(v').

Proof. We proceed by induction on rank of v.

Basis step. Suppose v is of rank 0, so $\psi(v) = \psi(v')$ is a constant sign c. But since GVG is reduced, L(v') = c and v' is also of rank 0.

Inductive step. Suppose for some $r > 0$, rank(w) = rank(w') for all $w,w' \in V$ such that rank(w) $< r$ and $\psi(w) = \psi(w')$. Consider some $v,v' \in V$ such that rank(v) = r.

Case a. Suppose $\psi(v) = \psi(v')$ is the function application $(\theta\ \alpha_1 \ldots \alpha_k)$. Then by Theorem 2.4, $\psi(v) = \psi(u)$ for all $u \in H(v)$, and similarily, $\psi(v') = \psi(u')$ for all $u' \in H(v')$. Fix some $u \in H(v)$ and $u' \in H(u')$. By definition of Γ_{GVG}, L(u) = L(u') = θ and if $w_1,\ldots,w_k$ are the immediate successors of u and $w_1',\ldots,w_k'$ are the immediate successors of u', then $\alpha_i = \psi(w_i) = \psi(w_i')$ for $i = 1,\ldots,k$. By the induction hypothesis, rank(w_i) = rank(w_i') for $i = 1,\ldots,k$.

Hence, rank(v) = rank(u)

= 1 + MAX{rank(w_1),...,rank(w_k)}

= 1 + MAX{rank(w_1',...,rank(w_k')}

= rank(u')

= rank (v').

Case b. Suppose $\psi(v) = \psi(v')$ is an input variable. By Theorem 2.4, $\psi(v) = \psi(v') = L(u)$ for some $u \in V$ contained on all value paths from v and v'. Hence, rank(v) = rank(v') = rank(u). $\square$

To compute the rank of all nodes in GVG we use a modified version of the depth first search developed by Tarjan[T1]. Because the search proceeds backwards, we require reverse adjacency lists to store edges in E. Note that the RANK(v) is used in two different ways; first to store the number of successors of node v which have not been visited, and later RANK(v) is set to rank(v). Let V_r, $\hat{V}_r$ be the nodes in V, $\hat{V}$ of rank r. We initially compute $\hat{V}_0$ and on the r'th execution of the main loop we compute $V_r - \hat{V}_r$ and $\hat{V}_{r+1}$.

Algorithm 2B.

INPUT GVG = (V, E, L)

OUTPUT RANK

```
begin
  declare RANK:= an array of integers of length |V|;
  for all v ε V do
    RANK(v) := - outdegree(v);
  r := 0;
  Q' := {v |  L(v) is a constant sign };
  untill  Q' = the empty set {} do
    begin
      Q := Q'; Q' := the empty set {};
      comment Q = $\hat{V}_r$;
    L:  untill Q = the empty set {} do
        begin
          delete v from Q;
          for each immediate predecessor u of v do
            if L(v) is a function sign then
              if RANK(u) = -1 then
                begin
                  comment u ε $\hat{V}_{r+1}$;
                  RANK(u) := r+1;
                  add u to Q'
                end
              else RANK(u) := RANK(u) + 1;
            else if RANK(u) < 0 then
              begin
                comment u ε $V_r - \hat{V}_r$;
                RANK(u) := r;
                add u to Q
              end;
        end;
      r := r + 1;
    end;
 end.
```

<u>Theorem 2.5</u>. Algorithm 2B is correct and has time cost linear in $|V|+|E|$.

<u>Proof</u> by induction on r.

<u>Basis step</u>. Initially, RANK(v) is set to - (outdegree of v) for each $v \in V$. So if L(v) is labeled with a constant sign then RANK(v) is set to 0. Also, Q is initially set to $\hat{V}_0$ just before label L.

<u>Inductive step</u>. Suppose for some $r > 0$, we have on entering the inner loop at label L on the r'th time:

(1) $Q = \hat{V}_r$,

(2) For each $v \in V$, RANK(v) = rank(v) if rank(v) < r or $v \in \hat{V}_r$, and RANK(v) = - (number of successors of v with rank > r) if rank(v) > r or $v \in V_r-\hat{V}_r$.

In the inner loop we add to Q exactly the nodes $V_r-\hat{V}_r = \{v \in V-\hat{V} \mid$ some element of $\hat{V}_r$ is reachable by a value path from $v\}$. For each such $v \in V_r-\hat{V}_r$ added to Q, RANK(v) is set to r. Also, for each $v \in \hat{V}$, if rank(v) > r+1 then RANK(v) is incremented by 1 for each immediate successor of v of rank r; if rank(v) = r+1 then all immediate successors of v are of rank $\leq$ r so RANK(v) is set to r+1 and v is added to Q. Thus, (1) and (2) are satisfied entering the loop on the r+1 time.

Now we show that Algorithm 2B may be implemented in linear time. For each node $v \in V$ we keep a list (the reverse adjacency list), giving all predecessors of v. To process any $v \in Q'$ requires time O(1 + indegree(v)). Since each node is added to Q' exactly once, the total time cost is linear in $|V|+|E|$. □

This suffices for the construction of ψ; $\psi(v)$ for $v \in \hat{V}_0$, $V_0-\hat{V}_0$, $\hat{V}_1$, $V_1-\hat{V}_1, \ldots$ may be determined by alternately applying the definition of Γ_{GVG} and Theorem 2.4.

Using this method could be inefficient, since Theorem 2.4 could be expensive to apply and the representations of the values could grow rapidly in size. The first problem is solved by reducing it to the problems of P-graph completion and decomposition as described in the next section. The second problem is solved by constructing a special labeled dag; the construction of this dag and the final algorithm are given in Section 2.7.

2.6 P-graph Completion and Decomposition.

Let GVG = (V, E, L) be a reduced global value graph as above. This section presents an efficient method for applying Theorem 2.4 to nodes in $V_r - \hat{V}_r$ (i.e. nodes of rank r labeled with input variables). Now to compute ψ, the minimal element of Γ_{GVG}, it suffices to find the partitioning of V such that $\psi(v) = \psi(u)$ iff v, u are in the same block of the partition. To represent such a partitioning, we distinguish one node of each block of the partitioning to be the value source of all other nodes of that block. We require that if $v \in V - \hat{V}$ (i.e. v is labeled with an input variable) then $\psi(v) = L(v)$ iff v is a value source. Let V^* be the set of value sources and let VS be a mapping from nodes in V to their value sources. Hence the fixed points of VS are the value sources and $VS^{-1}[V^*]$ is a partitioning of V. Note that, in general, the definition of "value source" is not uniquely determined, so the definition of V^* and VS depends on our particular choice of value sources.

We shall find value sources by reducing this problem to the problems of P-graph completion and decomposition stated below.

Let $G = (V_G, E_G)$ be any directed graph and let $S \subseteq V_G$ be a set of vertices of G such that for each vertex $v \in V_G$ there is some vertex $u \in S$ from which v is reachable.

P-Graph Completion Problem. Find

$S^+ = S \cup \{v \in V_G \mid$ there are almost disjoint paths from distinct elements of S to v not containing any other element of $S\}$.

This form of the problem is due to Karr[K], who shows that it is equivalent to the original formulation due to Shapiro and Saint[SS]. (Actually, this form is slightly more general than Karr's; Karr satisfies our restriction on S by stipulating that there is a single $r \in S$ from which every $v \in V_G$ is reachable.) Karr proves that for each $v \in V_G$ there is one and only one element of S^+ from which v is reachable (and his proof extends directly to our slightly more general problem).

P-Graph Decomposition Problem. Given G and S^+, find, for each $v \in V_G$, the unique $u \in S^+$ from which v is reachable.

We first show these problems can be solved efficiently. Shapiro and Saint give an $O(|V_G|^2)$ algorithm, while Karr gives a more complex $O(|V_G|\log|V_G|+|E_G|)$ algorithm. Here we reduce these problems to the computation of a certain dominator tree, for which there is an almost linear time algorithm as noted in Section 1.2. (This construction was discovered independently by Tarjan[T6].)

Let h be a new node not in V_G, and let G' be the rooted directed graph

$(V_G \cup \{h\}, E_G \cup \{(h,v) \mid v \in S\} - \{(u,v) \mid u \in V_G, v \in S\}, h)$.

Thus G' is derived from G by adding a new root h, linking h to every node in S, and removing the edges of G which lead to nodes in S. Let T be the dominator tree of G'.

<u>Lemma 2.6.1</u>. The members of S^+ are the sons of h in T.

<u>Proof</u>. <u>IF</u>. Let $v \in S^+$. If $v \in S$ then h is a predecessor of v in G' so h is the father of v in T. If $v \in S^+-S$ then by definition of S^+ there are almost disjoint paths p_1, p_2 in G from distinct elements of S to v not containing any other element of S. Clearly p_1 and p_2 are also paths in G' since they contain no edge entering a member of S. Then (h,p_1) and (h,p_2) are paths from h to v in G' which have only their endpoints in common, so v is a son of h in T.

<u>ONLY IF</u>. Suppose v is a son of h in T. If h is a predecessor of v in G' then $v \in S \subseteq S^+$. Otherwise there are in G' paths (h,p_1) and (h,p_2) from h to v which have only their endpoints in common. Moreover these paths contain no element of S except for the first nodes of p_1, p_2, since no edge of G' enters an element of S except from h. Hence p_1, p_2 are almost disjoint paths in G' from distinct members of S to v not containing any other element of S, and hence $v \in S^+$. □

<u>Theorem 2.6.1</u>. For each $v \in V_G$, the unique node in S^+ from which v is reachable in G is the unique node which is a son of h and an ancestor of v in T.

<u>Proof</u>. Let w be that ancestor of v which is a son of h in T. By Lemma 2.6.1, $w \in S^+$, and clearly v is reachable from

w in G since it is reachable from w in T. Conversely, if w ϵ S^+ is reachable from v in G then w is a son of h in T by Lemma 2.6.1, and w must be an ancestor of v since otherwise v would be reachable from some other member of S^+. □

Now we establish the relation of these problems to the problem of finding V^* and VS as stated above. Fix some V^* and VS by choosing one node of GVG for each value of ψ on V consistent with our definition of value sources. For each rank r, let $G_r = (V_r, E_r)$, where V_r is the set of all nodes of a reduced GVG of rank r as defined in Section 2.5 and E_r is the edge set derived from E by

(1) deleting all edges except value edges between nodes of rank r,

(2) for those remaining value edges (v,u) entering u ϵ $\hat{V}_r$, substituting instead the edge (v,VS(u)),

(3) finally reversing all edges.

Note that any edge of GVG departing from a member of $\hat{V}_r$ enters a node of rank r-1. Let S_r be the set of all value sources of $\hat{V}_r$ plus all nodes of rank r labeled with input variables which have a departing value edge entering a node of rank greater than r. Note that for each node v of G_n, there is a node in S_r from which v is reachable in G_r. Finally, let S_r^+ be defined from S_r as in the statement of the P-graph completion problem.

<u>Lemma 2.6.2</u>. The members of S_r^+ are the value sources of

rank r.

__Proof__. __IF__. Suppose $v \in S_r^+$.

__Case 1__. By definition, all elements of $\{VS(v) \mid v \in \hat{V}_r\}$ are value sources. Hence we need only consider the case where v is a node of rank r labeled with an input variable which has a departing value edge (v,z) entering a node z of rank greater than r. Since v is of rank r, v must also have a departing value edge (v,u) leading to a node of rank r. By Lemma 2.5, $\psi(z) \neq \psi(u)$, so by the definition of Γ_{GVG}, $\psi(v) = L(v)$ and v is a value source.

__Case 2__. Suppose there are in G_r almost disjoint paths $(x_1,x_2,\ldots,x_j)$ and $(y_1,y_2,\ldots,y_k)$ in G_r from distinct x_1, y_1 $\in S_r$ to v. By construction of G_r, there exist distinct $\bar{x}_1,\bar{y}_1 \in H(v)$ such that $VS(\bar{x}_1) = x_1$, $VS(\bar{y}_1) = y_1$, and $(x_2,\bar{x}_1)$ and $(y_2,\bar{y}_1)$ are value edges, and so $p_1 = (v{=}x_j,x_{j-1},\ldots,x_2,\bar{x}_1)$ and $p_2 = (v{=}y_k,y_{k-1},\ldots,y_2,\bar{y}_1)$ are almost disjoint value paths. Now suppose v is not a value source. Applying Theorem 2.4, there is a value source u (distinct from v) such that $\psi(v) = \psi(u) = L(u)$. Since p_1 and p_2 are almost disjoint they can not both contain u. Suppose, without loss of generality, that p_1 avoids u. Then all maximal value paths from $\bar{x}_1$ contain u. Also, by definition of S_r, $\bar{x}_1 = x_1$ and there is a value edge (v,z) such that z is not of rank r. Since any maximal value path from z must contain u, rank(z) = rank(u) implying that u is not of rank r. But, by hypothesis, all maximal value paths from v contain u, so rank(v) = rank(u). This implies that v is not of rank r, contradicting our assumptions. □.

By Karr's proof [K] of the uniqueness of the P-graph decomposition of G_r on S_r, we have

<u>Theorem</u> <u>2.6.2</u>. For all nodes $v \in V$ of rank r and labeled with an input variable, VS(v) is the unique value source contained on all paths in G_r from elements of S_r to v.

Thus the problem of computing VS reduces to the problem of decomposing the reduced global value graph by rank and then constructing dominator trees. The former can be done in linear time by Algorithm 2B of Section 2.5, the latter in almost linear time by Tarjan's Algorithm[T4].

2.7 The Algorithm for Symbolic Evaluation.

In this section we pull together the various pieces developed in Sections 2.3-2.6 to give a unified presentation of our algorithm for symbolic evaluation. Instead of using the GVG directly to represent ψ, as suggested in the beginning of Section 2.6, we more economically represent ψ by a dag D^* derived from GVG by collapsing nodes into their value sources; more precisely $D^* = (V^*, E^*, L^*)$ where

$V^* = \{VS(v) | v \in V\}$ = the set of value sources,

$E^* = \{(VS(v), VS(u)) | (v,u) \in E$ and $L(v)$ is a function sign$\}$,

L^* is the restriction of L to V^*.

Recall from Section 2.2 that rooted dags may be used to represent expressions in EXP.

Lemma 2.7. for each node $v \in V$,

$(D^*, VS(v))$ represents $\psi(v)$.

Proof. Note that by definition of VS, for each $v \in V$

$\psi(VS(v)) = \psi(v)$

for each $v \in V$, so we need only show for $v \in V^*$

(D^*, v) represents $\psi(v)$.

We proceed by induction on a topological ordering of D^*, from leaves to roots.

Basis step. If v is a leaf of D^*, then (D^*, v) represents the contant sign $L(v) = \psi(v)$.

Induction step. Suppose v is in the interior of D^* and (D^*,u) represents $\psi(u)$ for all sons u of v. Thus v must be labeled in L with a function sign θ and have immediate successors $u_1,\ldots,u_2$ in GVG. Then $VS(u_1),\ldots,VS(u_k)$ are the sons of v in D^* and for $i = 1,\ldots,k$ by the induction hypothesis $(D^*,VS(u_i))$ represents $\psi(VS(u_i))$

$$= \psi(u_i).$$

Thus (D^*,v) represents $(\theta\ \psi(u_1)\ldots\psi(u_k))$

$$= \psi(v) \text{ by definition of } \Gamma_{GVG}.$$

□

Our algorithm for symbolic evaluation is given below. As in Section 2.6, we compute ψ and VS in the order of the rank of nodes in V. The array COLOR is used to discover nodes with the same ψ.

Algorithm 2C.

INPUT GVG = (V, E, L)

OUTPUT VS and $D^* = (V^*,E^*,L^*)$.

```
begin
initialize:
  declare VS,COLOR := arrays of length |V|;
  procedure COLLAPSE(S,u):
    for all v ε S do
      begin
        VS(v) := u;
        if u≠ v then
          begin
            for each edge (w,v) entering v do
              substitute (w,u);
            for each edge (v,w) departing from v do
              substitute (u,w);
            delete v from the edge set;
          end;
      end;
  Compute new labeling L' of V by Algorithm 2A
    and reduce GVG as described in Section 2.3;
  Compute rank of nodes in V by Algorithm 2B
    of Section 2.4;
  for r := 0 to MAX{rank(v) | v $\in$ V} do
    begin
      Let $V_r$, $\hat{V}_r$ be the nodes in V, $\hat{V}$ of rank r;
      for all v $\in$ $\hat{V}_r$ do
        if r = 0 then COLOR(v) := L'(v)
        else COLOR(v) := <L(v),$u_1$,...,$u_k$> where
        $u_1$,...,$u_k$ are the current immediate
        successors of v;
      radix sort nodes in $\hat{V}_r$ by their COLOR;
      for each maximal set S $\subseteq$ $\hat{V}_r$ containing nodes
      with the same COLOR do
        begin
          choose some u $\in$ S;
          comment u is made a value source;
          COLLAPSE(S,u);
        end;
      Let h be some node not in $V_r$;
      $E_r$ := $S_r$ := the empty set {};
      for all v $\in$ $\hat{V}_r$ do add VS(v) to $S_r$;
      for all v $\in$ $V_r$-$\hat{V}_r$ do
        for each node u which is currently an
        immediate successor of v do
          if u is of rank r then
            add (u,v) to $E_r$;
          else add u to $S_r$;
      Let $T_r$ be the dominator tree of $G_r$ =
      ($V_r$ $\cup$ {h}, $E_r$ $\cup$ {(h,v)|v $\in$ $S_r$}, h);
      for all sons u of h in $T_r$ do
        begin
          comment By Theorem 2.6.1 and Lemma 2.6.2,
            u is a value source;
          COLLAPSE({the descendants of u in $T_r$},u);
          delete all edges departing from u;
        end;
    end;
  Let $V^*$, $E^*$ be the node set and edge list derived from V, E
    by the above collapses;
  for all v $\in$ $V^*$ do $L^*$(v) := L'(v);
end.
```

<u>Theorem 2.7</u>. Algorithm 2C is correct and can be implemented in almost linear time.

<u>Proof</u>. The correctness of Algorithm 2C follows directly from Theorems 2.6.1, 2.6.2 and Lemmas 2.6.2, 2.7.

In addition, we must show that Algorithm 2C can be implemented in almost linear time. The storage cost of GVG is linear in $|V|+|E|$. The initialization of Algorithm 2C costs time linear in $|N| + |A|$. Algorithms 2A and 2B cost linear time by Theorems 2.3 and 2.5, respectively. The time cost of the r'th execution of the main loop, exclusive of the computation of T_r, is linear in $|V_r| + |E_r|$, plus the sum of the outdegree of all $v \in V_r - \hat{V}_r$. (Here we assume that elements in the range of L' are representable in a fixed number of machine words and that the number of argument-places of function signs is bounded by a fixed constant, so a radix sort can be used to partition $\hat{V}_r$ by COLOR.) The computation of the dominator tree T_r requires by [T4] time cost almost linear in $|V_r| + |E_r|$. Thus, the total time cost is almost linear in $|V| + |E|$. □

This completes the presentation of our algorithm. The next section explains how with the aid of the preprocessing stage of Chapter 4 costing $O((|A|+\ell)\alpha(|A|+\ell))$ bit vector operations, we may construct a global value graph GVG^+ of size $O(d|A|+\ell)$ where d is often of order 1 for block-structured programs but may grow to $|\Sigma|$. (Thus this preprocessing stage offers no theoretical advantage but in practice often leads to a glbal value graph of size linear in the program and flow graph.) GVG$^+$ has the property that the minimal element of ΓGVG$^+$ is the minimal fixed point of the functional Ψ defined in Section 2.1. In contrast to Kildall's iterative method, which for a large class of programs has storage cost $\Omega(\ell|N|)$ and time cost $\Omega(\ell|N|^2)$, our direct method has storage cost linear in the size of GVG$^+$ and time cost almost linear in the size of GVG$^+$. Although either method may be improved somewhat through the use of domain-specific identities, as shown in Section 1.4, there is in general no algorithm for computing an optimal symbolic evaluation.

In Chapter 3 these methods are extended to programs which operate on structured data in a language such as PASCAL or LISP 1.0.

2.8 Improving the Efficiency of our Algorithm for Symbolic Evaluation

The primary goal of this Chapter was to construct the minimal fixed point ψ^* of the functional Ψ. Actually, Ψ was defined relative to a program P' derived from the original program P by adding dummy assignments of the form X := X at every block where some program variable $X \in \Sigma$ is not assigned (recall that Σ is the set of global program variables occurring in P). This does not change the semantics of the program but requires the addition of $O(|\Sigma||N|)$ text expressions whose covers we are not actually concerned with; in practice we need the covers given by ψ^* only over the domain of the original text expressions of P.

The methods of Sections 2.3-2.7 allow us to construct, for any global value graph GVG, the unique minimal element of Γ_{GVG} in space linear in the size of GVG and time almost linear in the size of GVG. Section 2.2 defines a global value graph GVG* of size $O(|\Sigma||A|+\ell)$ and with the property that ψ^* is the minimal element of Γ_{GVG^*}. Now we shall define a global value graph GVG$^+$ of size often considerably less but with the property that a restriction of ψ^* is the minimal element of Γ_{GVG^+}.

A path is m-avoiding if the path does not contain node m. Consider blocks m, n in the control flow graph such that m dominates n. A program variable $X \in \Sigma$ is definition-free

<u>between</u> <u>m</u> <u>and</u> <u>n</u>, if (1) $m = n$ or (2) $m \overset{+}{\rightarrow} n$ and X is not assigned to on any m-avoiding control path from an immediate successor of m to an immediate predecessor of n (otherwise X is <u>defined</u> between m and n). Let W be a function from text expressions which are input variables to blocks of the control flow graph; for each input variable $X^{\rightarrow n}$ which is an text expression, $W(X^{\rightarrow n}) = m$, where m is the first block on the dominator chain of the control flow graph from the start block s to n such that X is definition-free between m and n. An algorithm in Chapter 4 computes W in a number of bit vector steps almost linear in $|N|+\ell$.

It will be convenient to assume that for each text expression which is an input variable $X^{\rightarrow n}$ such that $W(X^{\rightarrow n}) = n$, X is assigned to at each block m immediately preceding n. We must add $O(d|N|)$ dummy assignments to accomplish this; d is often constant for block structured programs but may grow to $|\Sigma|$. Let VE be the set of pairs of text expressions (t,t') such that

(1) t is an input variable $X^{\rightarrow n}$

(2) t' is an output expression $X^{m\rightarrow}$

(3) either (a) $W(X^{\rightarrow n}) = n$ and m is an immediate predecessor of n in F, or (b) $W(X^{\rightarrow n}) = m \overset{+}{\rightarrow} n$.

Note that VE contains $O(d|A|+\ell)$ edges. Let GVG^{+} be the global value graph with value edges VE. The nodes in GVG^{+} will be identified with the text expression which they represent. Let $d = |VE|/|A|$ and observe that $d \leq |\Sigma|$. Then GVG+ is of size $O(|VE|+\ell) = O(d|A|+\ell)$.

Let $\psi+$ be the minimal element of Γ_{GVG+} and let ψ^* be the minimal fixed point of Ψ.

<u>Theorem 2.8</u>. $\psi+ = \overline{\Psi}^*$, where $\overline{\Psi}^*$ is the restriction of ψ^* to the domain of $\psi+$.

<u>Proof</u> Suppose $\overline{\Psi}^* \notin \Gamma_{GVG+}$, so there must be an input variable $X^{\rightarrow n}$ such that $\psi^*(X^{\rightarrow n}) \neq X^{\rightarrow n}$ and $\psi^*(X^{\rightarrow n}) \neq \psi(t)$ for some value edge $(X^{\rightarrow n},t) \in VE$. Then $n' = W(X^{\rightarrow n}) \overset{+}{\rightarrow} n$ and furthermore, $n' \overset{+}{\rightarrow}$ birthpoint($\psi(X^{\rightarrow n})$). Let ψ be the mapping from text expressions to EXP such that for each text expression t, $\psi(t)$ is derived from $\psi(t)$ by substituting $X^{n'\rightarrow}$ for each input variable $X^{\rightarrow m}$ such that $W(X^{\rightarrow m}) = n'$. Thus ψ is an element of Γ_{GVG+}, a contradiction with the assumption that ψ^* is the minimal element of Γ_{GVG+}.

Let $\overline{\Psi}+$ be the extension of $\psi+$ to the domain of ψ^* defined thus: for each input variable $X^{\rightarrow n}$ not in the domain of $\psi+$, let $\overline{\Psi}+(X^{\rightarrow n}) = \psi+(X^{m\rightarrow})$ where $m = W(X^{\rightarrow n})$. We claim $\overline{\Psi}+ \in \Gamma_{GVG+}$. Suppose $\overline{\Psi}+ \notin \Gamma_{GVG+}$, so there is an input variable $X^{\rightarrow n}$ such that $\overline{\Psi}+(X^{\rightarrow n}) \neq X^{\rightarrow n}$ and $\overline{\Psi}+(X^{\rightarrow n}) \neq \overline{\Psi}+(X^{m\rightarrow})$ for some m immediately preceeding n in the control flow graph F. Hence, $\overline{\Psi}+(X^{\rightarrow n}) = \overline{\Psi}+(X^{n'\rightarrow})$ where $n' = W(X^{\rightarrow n}) \overset{+}{\rightarrow} n$. But X is definition-free between n' and n, hence $(X^{\rightarrow m},X^{\rightarrow n'})$ is the unique value edge in VE departing from $X^{\rightarrow m}$. Let ψ be the mapping from text expressions to EXP such that for each text xpression t, $\psi(t)$ is the reduced expression derived from $\psi(t)$ by substituting $X^{n'\rightarrow}$ for each input variable $X^{\rightarrow\overline{n}}$ such hat $W(X^{\rightarrow\overline{n}}) = n'$. Then $\psi \in \Gamma_{GVG+}$, implying that $\psi+$ is not he minimal element of Γ_{GVG+}, a contradiction. □

2.9 Further Applications of Global Value Graphs: Live-Dead Analysis

A concept related to the value edges of VE defined in Section 2.8 is due to Schwartz[Sw2]: a pair of text expressions (t,t') is a use-definition link if t is an input variable $X^{\to n}$, t' is an output variable $X^{m\to}$, and X is not defined on some (rather than all) m-avoiding control path from an immediate successor of m to an immediate predecessor of n. Unfortunately there may be $\Omega(\ell^2)$ use-definition pairs whereas there are at most $O(|\Sigma||A|+\ell)$ value edges in VE.

A method for live-dead analysis has been described by Schwartz[Sc2]; his method uses use-definition links and would require $\Omega(\ell^2)$ operations even if implemented using efficient depth first search techniques. The global value graph GVG^+ with value edges VE may be also applied to live-dead analysis thus: We distinguish a set $\bar{V} \subseteq V$ which are text expressions corresponding to vital computations (for example all text expressions appearing in print statements). Then text expression t is live if t is involved in the computation of EXEC(t',p) for some vital text expression $t' \in \bar{V}$ and control path p from s to the block where t is located (otherwise t is dead). Deleting all dead text expressions would thus not interfere with the vital computations of the program. It follows that text expression t is live iff some element of $\bar{V}$ is reachable from

t. Live text expressions may thus be discovered in time linear in the size of GVG^+ by a reverse depth first search backwards from elements of V. All text expressions <u>not</u> reached are dead, and may be deleted.