If we define a meet operation in L^* as a pointwise meet on Γ, i.e., if for $\xi_1, \xi_2 \in L^*$, $\gamma \in \Gamma$, we define $(\xi_1 \wedge \xi_2)(\gamma) = \xi_1(\gamma) \wedge \xi_2(\gamma)$, then L^* becomes a semilattice. The smallest element in L^* is 0^*, where $0^*(\gamma) = 0$ for each $\gamma \in \Gamma$. The largest element in L^* is Ω^*, where $\Omega^*(\gamma) = \Omega$ for each $\gamma \in \Gamma$. Note that unless Γ is finite, L^* need not be bounded. However, if $\xi_1 \geq \xi_2 \geq \ldots \geq \xi_n \geq \ldots$ is an infinite decreasing chain in L^*, its limit is well-defined and can be computed as follows: For each $\gamma \in \Gamma$, the chain $\xi_1(\gamma) \geq \xi_2(\gamma) \geq \ldots$ must be finite (since L is bounded). Define $(\lim \xi_n)(\gamma)$ as the final value of that chain. Obviously $\lim \xi_n = \bigwedge_n \xi_n$ and in the same manner it can be shown that $\bigwedge_n \xi_n$ exists for any sequence $\{\xi_i\}_{i\geq 1}$ in L^*.

In order to describe F^*, we first need to define a certain operation in Γ.

Definition. $\circ : \Gamma \times E^* \longrightarrow \Gamma$ is a partially defined binary operation such that for each $\gamma \in \Gamma$ and $(m, n) \in E^*$ such that $\mathrm{CM}^{-1}\{\gamma\} \cap \mathrm{IVP}(r_1, m) \neq \varnothing$ we have

$$\gamma \circ (m, n) = \begin{cases} \gamma & \text{if } (m, n) \in E^0 \\ \gamma \,\|\, [m] & \text{if } (m, n) \text{ is a call edge in } E^1 \\ & \text{(i.e., if } m \text{ is a call block)} \\ \gamma(1 : \#\gamma - 1) & \text{(i.e., } \gamma \text{ without its last component)} \\ & \text{if } (m, n) \text{ is a return edge in } E^1 \text{ such} \\ & \text{that } \gamma(\#\gamma) \text{ is its corresponding call edge} \end{cases}$$

in all other cases, $\gamma \circ (m, n)$ is undefined. Here $\#\gamma$ denotes the length of γ.

The following lemma can be proved in an obvious and straightforward way.

Lemma 7-4.1. Let $\gamma \in \Gamma$, $(m, n) \in E^*$, $q \in \mathrm{IVP}(r_1, m)$ such that $\mathrm{CM}(q) = \gamma$. Then $\gamma_1 = \gamma \circ (m, n)$ is defined iff $q_1 = q \,\|\, (m, n)$ is in $\mathrm{IVP}(r_1, n)$, in which case $\mathrm{CM}(q_1) = \gamma_1$.

The operation $\circ$ defines the manner in which call strings are updated as data is propagated along an edge of the flow graph. Loosely put, the above lemma states that path incrementation is transformed into $\circ$ by the "homomorphism" CM.

Example 6. In Example 1 of Section 7-3, we have

$$\lambda \circ (c_1, r_2) = (c_1)$$
$$(c_1) \circ (c_2, r_2) = (c_1 c_2)$$
$$(c_1 c_2) \circ (e_2, n_2) = (c_1)$$

and

$$(c_1 c_2) \circ (e_2, n_1)$$

is undefined, indicating that after p had called itself once, the return from p must be to the block following c_2 in p; it is illegal to return then to the main program.

Next, let $(m, n) \in E^*$, and let $f_{(m,n)} \in F$ be the data propagation map associated with (m, n). Note that by our assumptions $f_{(m,n)} = \mathbf{id}_L$ if $(m, n) \in E^1$, since in these cases m is a block containing only a jump which in itself does not affect data attributes. Define $f^*_{(m,n)} : L^* \longrightarrow L^*$ as follows: For each $\xi \in L^*$, $\gamma \in \Gamma$,

$$f^*_{(m,n)}(\xi)(\gamma) = \begin{cases} f_{(m,n)}(\xi(\gamma_1)) & \text{if there exists (a necessarily unique)} \\ & \gamma_1 \text{ such that } \gamma_1 \circ (m, n) = \gamma \\ \Omega & \text{otherwise} \end{cases}$$

The intuitive interpretation of this formula is as follows: $f^*_{(m,n)}(\xi)$ represents information at the start of n which is obtained by propagation of the information ξ, known at the start of m, along the edge (m, n). For each $\gamma_1 \in \Gamma$ for which $\xi(\gamma_1)$ is defined, we propagate $\xi(\gamma_1)$, the γ_1-selected data available at the start of m, to the start of n in standard intraprocedural fashion (that is, using $f_{(m,n)}$). However, this propagated data is now associated not with γ_1 but with $\gamma_1 \circ (m, n)$, which "tags" the set of paths obtained by concatenating (m, n) to all paths which are "tagged" by γ_1, which lead to m, and along which $\xi(\gamma_1)$ has been propagated. If $\gamma_1 \circ (m, n)$ is undefined, then, by Lemma 7-4.1, $\xi(\gamma_1)$ should not be propagated through (m, n) since no path which leads to m and is tagged by γ_1 can be concatenated with (m, n) in an interprocedurally valid manner. In this case, we simply discard $f_{(m,n)}(\xi(\gamma_1))$, as indicated by the above formula.

Example 7. In Example 1 of Section 7-3, let $\xi_0 = \{(\lambda, 1)\} \in L^*$. Then (for notational convenience, call strings are written without enclosing parentheses): $\xi_1 = f^*_{(c_1,r_2)}(\xi_0) = \{(c_1, 1)\}$, since $\lambda \circ (c_1, r_2) = c_1$ and ξ_0 is defined only at λ. Note that $f_{(c_1,r_2)} = \mathbf{id}$, as is the case for all interprocedural jumps.

$$\xi_2 = f^*_{(r_2,c_2)}(\xi_1) = \{(c_1, f_{(r_2,c_2)}(1))\} = \{(c_1, 0)\}$$

This edge is intraprocedural, so that call strings need not be modified.

$$\xi_3 = f^*_{(c_2,r_2)}(\xi_2) = \{(c_1 c_2, 0)\}$$
$$\xi_4 = f^*_{(r_2,e_2)}(\xi_3) = \{(c_1 c_2, 0)\}$$
$$\xi_5 = f^*_{(e_2,n_2)}(\xi_4) = \{(c_1, 0)\}$$

[But note that, e.g., $f^*_{(e_2,n_1)}(\xi_4)$ = totally undefined map (Ω^*) in L^*, since the only $\gamma_1 \in \Gamma$ for which $\gamma_1 \circ (e_2, n_1)$ is defined is the string c_1, but $\xi_4(c_1)$ is undefined.]

$$\xi_6 = f^*_{(n_2,e_2)}(\xi_5) = \{(c_1, 1)\}$$
$$\xi_7 = f^*_{(e_2,n_1)}(\xi_6) = \{(\lambda, 1)\} \qquad \text{(compare with } f^*_{(e_2,n_1)}(\xi_4)\text{!)}$$

To summarize, we have traced one possible interprocedurally valid path from c_1 to n_1, starting with the information that $a * b$ is available at c_1 and obtaining the fact that it is still available at n_1 (considering just this path, of course). An attempt to return to the main program prematurely resulted in completely discarding the information.

F^* is now defined as the smallest subset of maps acting in L^* which contains $\{f^*_{(m,n)} : (m, n) \in E^*\}$ and the identity map in L^* and which is closed under functional composition and meet.

Lemma 7-4.2.

1. If F is monotone in L, then F^* is monotone in L^*.
2. If F is distributive in L, then F^* is distributive in L^*.
3. If F is distributive in L, then for each $(m, n) \in E$, $f^*_{(m,n)}$ is *continuous* in L^*, that is, $f^*_{(m,n)}(\bigwedge_k \xi_k) = \bigwedge_k f^*_{(m,n)}(\xi_k)$, for each collection $\{\xi_k\}_{k\geq 1} \subseteq L^*$.

Proof. It is easily seen that it is sufficient to prove (1) or (2) for the set $\{f^*_{(m,n)} : (m, n) \in E^*\}$, and this is straightforward from the definitions.

To prove (3), note that for each $\gamma \in \Gamma$ for which there exists $\gamma_1 \in \Gamma$ such that $\gamma_1 \circ (m, n) = \gamma$ we have

$$f^*_{(m,n)}\Big(\bigwedge_{k\geq 1} \xi_k\Big)(\gamma) = f_{(m,n)}\Big(\bigwedge_{k\geq 1} \xi_k(\gamma_1)\Big)$$

But since L is bounded, there exists $k_0(\gamma_1)$ such that the last expression equals $f_{(m,n)}\Big(\bigwedge_{1\leq k\leq k_0(\gamma_1)} \xi_k(\gamma_1)\Big)$, which in turn, by the distributivity of $f_{(m,n)}$, equals

$$\bigwedge_{1\leq k\leq k_0(\gamma_1)} f_{(m,n)}(\xi_k(\gamma_1)) = \bigwedge_{1\leq k\leq k_0(\gamma_1)} f^*_{(m,n)}(\xi_k)(\gamma) \geq \Big(\bigwedge_{k\geq 1} f^*_{(m,n)}(\xi_k)\Big)(\gamma)$$

Thus $f^*_{(m,n)}\Big(\bigwedge_{k\geq 1} \xi_k\Big) \geq \bigwedge_{k\geq 1} f^*_{(m,n)}(\xi_k)$. The converse inequality is immediate from the monotonicity of $f^*_{(m,n)}$. ■

Remark: Note that interprocedural, as distinct from intraprocedural, data flow frameworks depend heavily on the flow graph (Γ itself may vary from one flow graph to another). Thus, for example, there is no simple way to obtain F^* directly from F without any reference to the flow graph. This will not create any problems in the sequel, and we argue that even in the intraprocedural case it is a better practice to regard data flow frameworks as graph-dependent.

We can now define a data flow problem for G^*, using the new framework (L^*, F^*), in which we seek the maximal fixed-point solution of the following equations in L^*:

$$\begin{aligned} x^*_{r_1} &= \{(\lambda, 0)\} \qquad \text{where } \lambda \text{ is the null call string} \\ x^*_n &= \bigwedge_{(m,n)\in E^*} f^*_{(m,n)}(x^*_m) \qquad n \in N^* - \{r_1\} \end{aligned} \tag{7-11}$$

We can show the existence of a solution to these equations in the following manner: Let $x^{*(0)}_{r_1} = \{(\lambda, 0)\}$, $x^{*(0)}_n = \Omega^*$ for all $n \in N^* - \{r_1\}$. Then apply Eqs. (7-11) iteratively to obtain new approximations to the x^*'s. Let $x^{*(i)}_n$ denote the ith approximation computed in this manner.

Since $x^{*(0)}_n \geq x^{*(1)}_n$ for all $n \in N^*$, it follows inductively, from the monotonicity of $f^*_{(m,n)}$ for each $(m, n) \in E^*$, that $x^{*(i)}_n \geq x^{*(i+1)}_n$ for all $i \geq 0$, $n \in N^*$. Thus, for each $n \in N^*$, $\{x^{*(i)}_n\}_{i\geq 0}$ is a decreasing chain in L^*, having a limit, and we define $x^*_n = \lim_i x^{*(i)}_n$. It is rather straightforward to show that $\{x^*_n\}_{n\in N^*}$ is indeed a solution to (7-11) and that in fact it is the maximal fixed-point solution of (7-11).

Having defined this solution, we will want to convert its values to values in L, because L^* has been introduced only as an auxiliary semilattice, and our aim is really to obtain data in L for each basic block. Since there is no longer a need to split the data at node n into parts depending on the interprocedural flow leading to n, we can combine these parts together, i.e., take their meet. For each $n \in N^*$, we can then simply define

$$x'_n = \bigwedge_{\gamma\in\Gamma} x^*_n(\gamma) \tag{7-12}$$

A detailed example of applying this technique to our running example (Example 1 of Section 7-3) will be given in the next section.

In justifying the approach that we have just outlined, our first step is to prove that x'_n coincides with the meet-over-all-interprocedurally-valid-paths solution y_n defined in the preceding section. This can be shown as follows:

Definition. Let $\text{path}_{G^*}(r_1, n)$ denote the set of all execution paths (whether interprocedurally valid or not) leading from r_1 to $n \in N^*$. For each $p = (r_1, s_2, \ldots, s_k, n) \in \text{path}_{G^*}(r_1, n)$ define $f^*_p = f^*_{(s_k,n)} \circ f^*_{(s_{k-1},s_k)} \circ \cdots \circ f^*_{(r_1,s_2)}$. For each $n \in N^*$ define $y^*_n = \bigwedge\{f^*_p(x^*_{r_1}) : p \in \text{path}_{G^*}(r_1, n)\}$.

Since $\text{path}_{G^*}(r_1, n)$ is at most countable, this (possibly infinite) meet in L^* is well defined.

Theorem 7-4.3. If (L, F) is a distributive data flow framework, then, for each $n \in N^*$, $x^*_n = y^*_n$.

Proof. The proof follows (it is quite similar to the proof of an analogous theorem of Kildall for a bounded semilattice [Kild73]):

1. Let $n \in N^*$ and $p \equiv (r_1, s_2, \ldots, s_k, n) \in \text{path}_{G^*}(r_1, n)$. By (7-11) we have
$$x^*_{s_2} \le f^*_{(r_1,s_2)}(x^*_{r_1})$$
$$x^*_{s_3} \le f^*_{(s_2,s_3)}(x^*_{s_2})$$
$$\vdots$$
$$x^*_n \le f^*_{(s_k,n)}(x^*_{s_k})$$
Combining all these inequalities, and using the monotonicity of the f^*'s, we obtain $x^*_n \le f^*_p(x^*_{r_1})$, and therefore $x^*_n \le y^*_n$.

2. Conversely, we will prove by induction on i that
$$x^{*(i)}_n \ge y^*_n \qquad \text{for all } i \ge 0,\ n \in N^*$$
Indeed, let $i = 0$. If $n \ne r_1$, then $x^{*(0)}_n = \Omega^* \ge y^*_n$. On the other hand, the null execution path $p_0 \in \text{path}_{G^*}(r_1, r_1)$, so that $y^*_{r_1} \le f^*_{p_0}(x^*_{r_1}) = x^*_{r_1} = x^{*(0)}_{r_1}$. Thus the assertion is true for $i = 0$. Suppose that it is true for some $i \ge 0$. Then $x^{*(i+1)}_{r_1} = x^{*(i)}_{r_1} \ge y^*_{r_1}$, and for each $n \in N^* - \{r_1\}$ we have
$$x^{*(i+1)}_n = \bigwedge_{(m,n)\in E^*} f^*_{(m,n)}(x^{*(i)}_m) \ge \bigwedge_{(m,n)\in E^*} f^*_{(m,n)}(y^*_m)$$
by the induction hypothesis.

We now need the following:

Lemma 7-4.4. For each $(m, n) \in E^*, f^*_{(m,n)}(y^*_m) \ge y^*_n$.

Proof. Since $f^*_{(m,n)}$ is distributive and continuous on L^* (Lemma 7-4.2), we have
$$\begin{aligned} f^*_{(m,n)}(y^*_m) &= f^*_{(m,n)}(\bigwedge \{f^*_p(x^*_{r_1}); p \in \text{path}_{G^*}(r_1, m)\}) \\ &= \bigwedge \{f^*_{(m,n)}(f^*_p(x^*_{r_1})) : p \in \text{path}_{G^*}(r_1, m)\} \\ &\ge \bigwedge \{f^*_q(x^*_{r_1}) : q \in \text{path}_{G^*}(r_1, n)\} = y^*_n \quad \blacksquare \end{aligned}$$

Now returning to Theorem 7-4.3, it follows by Lemma 7-4.4 that $x^{*(i+1)}_n \ge \bigwedge_{(m,n)\in E^*} y^*_n = y^*_n$ (each $n \in N^*$ is assumed to have predecessors). Hence assertion (2) is established, and it follows that for each $n \in N^*$, $x^*_n = \lim_i x^{*(i)}_n = \bigwedge_{i\ge 1} x^{*(i)}_n \ge y^*_n$, so that $x^*_n = y^*_n$. $\blacksquare$

Lemma 7-4.5. Let $n \in N^*$, $p = (r_1, s_1, \ldots, s_k, n) \in \text{path}_{G^*}(r_1, n)$ and $\gamma \in \Gamma$. Then $f^*_p(x^*_{r_1})(\gamma)$ is defined iff $p \in \text{IVP}(r_1, n)$ and $\text{CM}(p) = \gamma$. If this is the case, then $f^*_p(x^*_{r_1})(\gamma) = f_p(0)$.

Proof. The proof is by induction on $l(p)$, the length of p (i.e., the number of edges in p). If p is the null path, then n must be equal to r_1. Moreover, $\text{CM}(p) = \lambda, p \in \text{IVP}(r_1, r_1)$ and $f^*_p(x^*_{r_1}) = x^*_{p_1}$ is defined only at λ and equals $0 = f_p(0)$. Thus our assertion is true if $l(p) = 0$.

Suppose that this assertion is true for all $n \in N^*$ and $p \in \text{path}_{G^*}(r_1, n)$such that $l(p) < \lambda$. Let $n \in N^*$ and $p = (r_1, s_2, \ldots, s_k, n)$ be a path of length k in $\text{path}_{G^*}(r_1, n)$. Let $p_1 = (r_1, s_2, \ldots, s_k)$. By definition, for each $\gamma \in \Gamma$ we have
$$\begin{aligned} f^*_p(x^*_{r_1})(\gamma) &= f^*_{(s_k,n)}[f^*_{p_1}(x^*_{r_1})](\gamma) \\ &= \begin{cases} f_{(s_k,n)}[f^*_{p_1}(x^*_{r_1})(\gamma_1)] & \text{if there exists } \gamma_1 \in \Gamma \text{ such that } \gamma_1 \circ (s_k, n) = \gamma \\ \Omega & \text{otherwise} \end{cases} \end{aligned}$$

Thus $f^*_p(x^*_{r_1})(\gamma)$ is defined iff there exists $\gamma_1 \in \Gamma$ such that $\gamma_1 \circ (m, n) = \gamma$ and $f^*_{p_r}(x^*_{r_1})(\gamma_1)$ is defined. By our inductive hypothesis, this is the case iff $p_1 \in \text{IVP}(r_1, s_k)$, $\text{CM}(p_1) = \gamma_1$ and $\gamma_1 \circ (s_k, n) = \gamma$. By Lemma 7-4.1, these last conditions are equivalent to $p \in \text{IVP}(r_1, n)$ and $\text{CM}(p) = \gamma$.

If this is the case, then again, by our inductive hypothesis, $f^*_{p_1}(x^*_{r_1})(\gamma_1) = f_{p_1}(0)$ and so
$$f^*_p(x^*_{r_1})(\gamma) = f_{(s_k,n)}[f_{p_1}(0)] = f_p(0) \quad \blacksquare$$

Now we can prove the main result of this section:

Theorem 7-4.6. For each $n \in N^*$, $x'_n = y_n$.

Proof. Let $\gamma \in \Gamma$. By Theorem 7-4.3,
$$x^*_n(\gamma) = \{f^*_p(x^*_{r_1})(\gamma) : p \in \text{path}_{G^*}(r_1, n)\}$$
and by Lemma 7-4.5,
$$= \bigwedge \{f_p(0) : p \in \text{IVP}(r_1, n) \text{ such that } \text{CM}(p) = \gamma\}$$
Thus, by (7-12),
$$x'_n = \bigwedge_{\gamma\in\Gamma} x^*_n(\gamma) = \bigwedge \{f_p(0) : p \in \text{IVP}(r_1, n)\} = y_n \quad \blacksquare$$

Corollary 7-4.7. If the flow graph G^* is nonrecursive, then the iterative solution of Eqs. (7-11) that we have described will converge and yield the desired meet-over-all-interprocedurally-valid-paths solution of these equations.

Proof. Convergence is assured since Γ is finite, and hence L^* is bounded. Thus (L^*, F^*) is a distributive data flow framework, and by standard arguments the iterative solution of (7-11) must converge

[Kild73, Hech77]. Therefore, Theorem 7-4.6 implies that the limiting solution coincides with the meet-over-all-paths solution. ■

The call-strings approach is of questionable feasibility if Γ is infinite, i.e., if G^* contains recursive procedures. Moreover, as for the functional approach, it is rather hopeless to convert it into an effective algorithm for handling the most general cases of certain data flow problems such as constant propagation. However, as we shall see in the following section, a fairly practical variant of the call-strings approach can be devised for data flow frameworks with a finite semilattice L.

Let us also observe the similarity between the call-string approach and the inline expansion method (discussed, e.g., in [Alle77]). Indeed, tagging data by call strings amounts essentially to creating virtual copies of each procedure, one copy for each possible calling sequence reaching that procedure. Indeed let $\gamma = c_1c_2 \ldots c_j \in \Gamma$. Then, if c_j calls procedure p from procedure p', we can substitute the virtual copy of p corresponding to γ at the place of c_j in the virtual copy of p' corresponding to $\gamma' = c_1c_2 \ldots c_{j-1}$. Doing so, c_j and the return from p become no-ops, and we get a full inline expansion of procedures.

7-5. DATA FLOW ANALYSES USING A FINITE SEMILATTICE

Let (L, F) be a distributive data flow framework such that L is finite. As we have seen, the functional approach described in Section 7-3 converges for such a framework. We will show in this section that it is also possible to construct a call-strings algorithm which converges for these frameworks. As noted in the previous section, convergence is ensured if Γ is finite. The idea behind our modified approach is to replace Γ by some finite subset Γ_0 and allow propagation only through interprocedurally valid paths which are mapped into elements of Γ_0. Such an approach is not generally feasible because it can lead to an overestimated (and unsafe) solution, since it does not trace information along all possible paths. However, using the finiteness of L, we will show that Γ_0 can be chosen in such a way that no information gets lost and the algorithm produces the same solution as defined in Section 7-4.

We begin to describe our approach without fully specifying Γ_0. Later we will show how Γ_0 should depend on L in order to guarantee the above solution.

Definitions.

1. Let Γ_0 be some finite subset of Γ with the property that if $\gamma \in \Gamma_0$ and γ_1 is an initial subtuple of γ, then $\gamma_1 \in \Gamma_0$ too.

2. For each $n \in N^*$, let $\mathrm{IVP}'(r_1, n)$ denote the set of all $q \in \mathrm{IVP}(r_1, n)$ such that for each initial subpath q_1 of q (including q), $\mathrm{CM}(q_1) \in \Gamma_0$.
3. We also modify the $\circ$ operation so that it acts in Γ_0 rather than in Γ, as follows: If $\gamma \in \Gamma_0$, $(m, n) \in E^*$ such that there exists $q \in \mathrm{IVP}'(r_1, m)$ where $\mathrm{CM}(q) = \gamma$, then

$$\gamma \circ (m, n) = \begin{cases} \gamma & \text{if } (m, n) \in E^0 \\ \gamma \,\|\, [m] & \text{if } (m, n) \text{ is a call edge in } E^1 \text{ and } \\ & \gamma \,\|\, [m] \in \Gamma_0 \\ \gamma(1 : \#\gamma - 1) & \text{if } (m, n) \text{ is a return edge in } E^1 \text{ and} \\ & \gamma(\#\gamma) \text{ is the call block preceding } n \\ \text{undefined in all other cases} & \end{cases}$$

The only difference between this definition of $\circ$ and the previous one is that it will not add a call block m to a call string γ unless the resulting string is in Γ_0. When this is not the case, information tagged by γ will be lost when propagating through (m, n), unless it is also tagged by some other call string to which m can be concatenated.

The following lemma is analogous to Lemma 7-4.1:

Lemma 7-5.1. Let $\gamma \in \Gamma_0$, $(m, n) \in E^*$, $q \in \mathrm{IVP}'(r_1, m)$ such that $\mathrm{CM}(q) = \gamma$. Then $\gamma_1 = \gamma \circ (m, n)$ is defined iff $q_1 = q \,\|\, (m, n)$ is in $\mathrm{IVP}'(r_1, n)$, in which case $\mathrm{CM}(q_1) = \gamma_1$.

We now define a data flow framework (L^*, F^*) in much the same way as in Section 7-4, but replace Γ by Γ_0. This leads to a bounded semilattice $L^* = L^{\Gamma_0}$ and to a distributive data flow framework (L^*, F^*).

Hence, Eqs. (7-11) come to be effectively solvable by any standard iterative algorithm which yields their maximal fixed-point solution. To this solution we will want to apply the following final calculation, which is a variant of (7-12):

$$x_n'' = \bigwedge_{\gamma \in \Gamma_0} x_n^*(\gamma) \tag{7-13}$$

Careful scrutiny of the analysis of the previous section reveals that the only place where the nature of Γ_0 and the operation $\circ$ are referred to is in Lemma 7-4.1, and it is easily seen that if we replace Γ and $\circ$ by Γ_0 and the modified $\circ$, throughout the previous analysis, and also replace $\mathrm{IVP}(r_1, n)$ by $\mathrm{IVP}'(r_1, n)$ for all $n \in N^*$, then by proofs completely analogous to those presented in Section 7-4 (but with one notable difference, i.e., that there is now no need to worry about continuity of F^* or infinite meets in L^*, since L^* is now known to be bounded) we obtain the following:

Theorem 7-5.2. For each $n \in N^*$

$$x''_n = y''_n \equiv \bigwedge \{f_p(0): p \in \text{IVP}'(r_1, n)\}$$

Up to this point, our suggested modifications have been quite general and do not impose any particular requirements upon L or upon Γ_0. On the other hand, Theorem 7-5.2 implies that x''_n is an overestimated solution, and as such is useless for purposes of our analysis, as it can yield unsafe information (e.g., it may suggest that an expression is available whereas it may actually be unavailable), unless we can show that x''_n coincides with the meet-over-all-interprocedurally-valid-paths solution of the attribute propagation equations which concern us. As will be shown below, this is indeed the case if L is finite.

Definition. Let $M \geq 0$ be an integer. Define Γ_M as the (finite) set of all call strings whose lengths do not exceed M. Γ_M obviously satisfies the conditions of part (1) of the previous definition.

Lemma 7-5.3. Let (L, F) be a data flow framework with L a finite semilattice, and let $M = K(|L| + 1)^2$, where K is the number of call blocks in the program being analyzed and $|L|$ is the cardinality of L. Let $\Gamma_0 = \Gamma_M$. Then, for each $n \in N^*$ and each execution path $q \in \text{IVP}(r_1, n)$ there exists another path $q' \in \text{IVP}'(r_1, n)$ such that $f_q(0) = f_{q'}(0)$.

Proof. By induction on the length of q. If the length is 0, then $n = r_1$ and q is the null execution path, which belongs to both $\text{IVP}(r_1, r_1)$ and $\text{IVP}'(r_1, r_1)$, so that our assertion is obviously true in this case.

Suppose that the lemma is true for all paths whose length is less than some $k \geq 1$, and let $n \in N^*$, $q \in \text{IVP}(r_1, n)$ be a path of length k. If $q \in \text{IVP}'(r_1, n)$ then there is nothing to prove, so assume that this is not the case, and let q_0 be the shortest initial subpath of q such that $\text{CM}(q_0) \notin \Gamma_0$. Then q_0 can be decomposed according to Eq. (7-3) as follows:

$$q_0 = q_1 \| (c_1, r_{p_2}) \| q_2 \| \ldots \| (c_j, r_{p_{j+1}}) \| q_{j+1}$$

Hence $j > M$. Next, consider the sequence $\{(c_i, \alpha_i, \beta_i)\}_{i=1}^{j}$, where, for each $i \leq j$, $\alpha_i = f_{q_i} \circ f_{q_{i-1}} \circ \ldots \circ f_{q_1}(0)$, and β_i is either Ω if the call at c_i is not completed in q (this call is certainly not completed in q_0), or $f_{\hat{q}_i}(0)$ if the call at c_i is completed in q, and $\hat{q}_i$ is the initial subpath of q ending at the return which completes the call. Thus, for each call, the sequence records the calling block, the value propagated along this path up to the call, and the value propagated up to the correspond-

ing return, if it materializes. The number of distinct elements of such a sequence is at most $K(|L| + 1)^2 = M$ (we do not count Ω as an element of L; if we did, then the bound can be reduced to $K|L|^2$). Since $j > M$, this sequence must contain at least two identical components (c_a, α_a, β_a) and (c_b, α_b, β_b), where $a < b \leq j$.

Now, if $\beta_a = \beta_b = \Omega$, then neither of the calls c_a, c_b is completed in q. If we rewrite q as

$$q = q'_1 \| (c_a, r_{p_{a+1}}) \| q'_2 \| (c_b, r_{p_{b+1}}) \| q'_3$$

then it is easily seen that the shorter path $\hat{q} = q'_1 \| (c_a, r_{p_{a+1}}) \| q'_3$ is also in $\text{IVP}(r_1, n)$. Moreover

$$\alpha_a = f_{q_1'}(0) = \alpha_b = f_{q_2'} \circ f_{q_1'}(0)$$

so that

$$f_q(0) = f_{q_3'} \circ f_{q_2'} \circ f_{q_1'}(0) = f_{q_3'} \circ f_{q_1'}(0) = f_{\hat{q}}(0)$$

By our induction hypothesis there exists $\hat{q}' \in \text{IVP}'(r_1, n)$ such that $f_{\hat{q}'}(0) = f_{\hat{q}}(0) = f_q(0)$, which proves the lemma for q.

On the other hand, if $\beta_a = \beta_b \neq \Omega$, then it follows that both calls c_a and c_b are completed in q, with c_b necessarily completed first. Thus we can write

$$q = q'_1 \| (c_a, r_{p_{a+1}}) \| q'_2 \| (c_b, r_{p_{b+1}}) \| q'_3 \| (e_{p_{b+1}}, n_b) \| q'_4 \| (e_{p_{a+1}}, n_a) \| q'_5$$

where $n_a = n_b$ is the block immediately following c_a. Again it follows that $\hat{q} = q'_1 \| (c_a, r_{p_{a+1}}) \| q'_3 \| (e_{p_{a+1}}) \| q'_5$ is in $\text{IVP}(r_1, n)$. Moreover

$$\alpha_a = f_{q_1'}(0) = \alpha_b = f_{q_2'} \circ f_{q_1'}(0)$$
$$\beta_a = f_{q_4'} \circ f_{q_3'} \circ f_{q_2'} \circ f_{q_1'}(0) = \beta_b = f_{q_3'} \circ f_{q_2'} \circ f_{q_1'}(0)$$

from which one easily obtains $f_q(0) = f_{\hat{q}}(0)$, and the proof can now continue exactly as before. ■

The main result of this section now follows immediately:

Theorem 7-5.4. Let (L, F) be a distributive data flow framework with a finite semilattice L, and let $\Gamma_0 = \Gamma_M$, with M as defined above. Then, for each $n \in N^*$, $x''_n = y_n$. That is, the modified algorithm described at the beginning of the present section yields a valid interprocedural solution.

Proof. Since $\text{IVP}'(r_1, n) \subset \text{IVP}(r_1, n)$ we have $x''_n \geq y_n$. On the other hand, let $q \in \text{IVP}(r_1, n)$. By Lemma 7-5.3 there exists $q' \in \text{IVP}'(r_1, n)$ such that $f_q(0) = f_{q'}(0) \geq \bigwedge \{f_p(0): p \in \text{IVP}'(r_1, n)\} = x''_n$. Hence $y_n = \bigwedge \{f_q(0): q \in \text{IVP}(r_1, n)\} \geq x''_n$. ■

Remark: Note that in Lemma 7-5.3 and Theorem 7-5.4, K can be replaced by the maximal number K' of distinct calls in any sequence of nested calls in the program being analyzed. In most cases this gives a significant improvement of the bound on M appearing in those two results.

We have now shown that finite data flow frameworks are solvable by a modified call-strings approach. However, the size of Γ_0 can be expected to be large enough to make this approach as impractical as the corresponding functional approach. But in several special cases we can reduce the size of Γ_0 still further. The following definition is taken from [Rose78b], rewritten in our notation:

Definition. A data flow framework (L, F) is called *decomposable* if there exists a finite set A and a collection of data flow frameworks $\{(L_\alpha, F_\alpha)\}_{\alpha \in A}$, such that

1. $L = \prod_{\alpha \in A} L_\alpha$, ordered in a pointwise manner induced by the individual orders in each L_α.
2. $F \subseteq \sum_{\alpha \in A} F_\alpha$. That is, for each $f \in F$ there exists a collection $\{f^\alpha\}_{\alpha \in A}$ where $f^\alpha \in F$ for each $\alpha \in A$, such that for each $x = (x_\alpha)_{\alpha \in A} \in L$ we have

$$f(x) = (f^\alpha(x_\alpha))_{\alpha \in A}$$

In the cases covered by this definition we can split our data flow framework into a finite number of "independent" frameworks, each inducing a separate data flow problem, and obtain the solution to the original problem simply by grouping all the individual solutions together.

For example, the standard framework (L, F) for available expressions analysis is decomposable into subframeworks each of which is a framework for the availability of a single expression. Formally, let A be the set of all program expressions. For each $\alpha \in A$ let $L = \{0, 1\}$ where 1 indicates that α is available and 0 that it is not. Then $\{0, 1\}^A$ is isomorphic with L (which is more conveniently represented as the power set of A). It is easily checked that each $f \in F$ can be decomposed as $\sum_{\alpha \in A} f^\alpha$, where for each $\alpha \in A$, $f^\alpha \in F_\alpha$, and is either the constant $\mathbf{0}$ if α can be killed by the propagation step described by f, is the constant $\mathbf{1}$ if α is unconditionally generated by that propagation step, and is the identity map in all other cases. The frameworks used for use-definition chaining and live variables have analogous decompositions.

A straightforward modification of Lemma 7-5.3, applied to each (L_α, F_α) separately yields the following improved result for decomposable frameworks:

Theorem 7-5.5. Let (L, F) be a decomposable distributive data flow framework with a finite semilattice. Define $M = K \cdot \max_{\alpha \in A} (|L_\alpha| + 1)^2$ and let $\Gamma_0 = \Gamma_M$. Then, for each $n \in N^*$, $y''_n = y_n$.

In the special case of available expressions analysis this is certainly an improvement of Theorem 7-5.4, since it reduces the bound on the length of permissible call strings from $K \cdot O(4^{|A|})$ to $9K$. For this analysis we can do even better since available expression analysis has the property appearing in the following definition.

Definition. A decomposable data flow framework (L, F) is called *1-related* if, for each $\alpha \in A$, F_α consists only of constant functions and identity functions.

This property is characteristic of situations in which there exists at most one point along each path which can affect the data being propagated. Indeed, consider a framework having this property; let $\alpha \in A$ and let $p = (s_1, s_2, \ldots, s_k)$ be an execution path. Let $j \leq k$ be the largest index such that $f^\alpha_{(s_{j-1}, s_j)}$ is a constant function. Then clearly $f^\alpha_p = f^\alpha_{(s_{j-1}, s_j)}$ and is therefore also a constant. Hence in this case the effect of propagation in L_α through p is independent of the initial data and is determined by the edge (s_{j-1}, s_j) alone. If no such j exists, then $f_p = \mathbf{id}|_{L_\alpha}$, in which case no point along p affects the final data.

Note also that since each F_α is assumed to be closed under functional meet, it follows that if (L, F) is 1-related then the only constant functions that F_α can contain are $\mathbf{0}$ (the smallest element in L_α) and $\mathbf{1}$ (the largest element). Hence we can assume, with no loss of generality, that L_α is the trivial lattice $\{0, 1\}$ for each $\alpha \in A$. All the classical data flow analyses mentioned above have 1-related frameworks. It is therefore easily seen that, under these assumptions, 1-related frameworks are those having a semilattice of effective height 1 [Rose78b, Section 7].

For frameworks having the 1-related property it is easy to replace an execution path q by a shorter subpath q such that $f^\alpha_{\hat{q}}(0) = f^\alpha_q(0)$ for some $\alpha \in A$. Indeed, to obtain such a $\hat{q}$ we have only to ensure that $\hat{q}$ is also interprocedurally valid and that the last edge (s, s') in q for which $f_{(s,s')}$ is constant belongs to $\hat{q}$. This observation allows us to restrict the length of permissible call strings still further. The following can then be shown:

Theorem 7-5.6. Let (L, F) be a 1-related distributive data flow framework. Put $\Gamma_0 = \Gamma_{3K}$. Then, for each $n \in N^*$, $x''_n = y_n$.

The analysis developed in this section and the previous one can be modified to deal with nondistributive data flow problems. In the nondistributive case, Theorems 7-4.6 and 7-5.2 guarantee only inequalities of the form

$x'_n \le y_n$ (resp. $x''_n \le y''_n$) for all $n \in N^*$. The arguments in this section show that under appropriate conditions $y''_n = y_n$ for each $n \in N^*$, so that assuming these conditions Theorems 7-5.4, 7-5.5, 7-5.6 all yield the inequalities $x''_n \le y_n$ for each $n \in N^*$. Thus, in the nondistributive case, our approach leads to an underestimated solution, as is the case for intraprocedural iterative algorithms for nondistributive frameworks [Kam77].

Example 8. We return to Example 1 studied in Section 7-3. Since available expressions analysis uses a 1-related framework, and since the flow graph appearing in that example satisfies $K = K' = 2$, we can take $\Gamma_0 = \Gamma_6$, and apply Kildall's iterative algorithm [Kild73] to solve Eqs. (7-11). Table 7-3 summarizes the steps which are then performed (for notational convenience call strings are written without enclosing parentheses):

Table 7-3

Propagate			*Workpile of nodes*
From	*To*	*Updated x* value*	*from which further propagation is required*
(initially)		$x^*_{r_1} = \{(\lambda, 0)\}$	$\{r_1\}$
r_1	c_1	$x^*_{c_1} = \{(\lambda, 1)\}$	$\{c_1\}$
c_1	r_2	$x^*_{r_2} = \{(c_1, 1)\}$	$\{r_2\}$
r_2	c_2	$x^*_{c_2} = \{(c_1, 0)\}$	$\{c_2\}$
r_2	e_2	$x^*_{e_2} = \{(c_1, 1)\}$	$\{c_2, e_2\}$
c_2	r_2	$x^*_{r_2} = \{(c_1, 1), (c_1c_2, 0)\}$	$\{e_2, r_2\}$
e_2	n_2	$x^*_{n_2} = \Omega^*$(unchanged)	$\{r_2\}$
e_2	n_1	$x^*_{n_1} = \{(\lambda, 1)\}$	$\{r_2, n_1\}$
r_2	c_2	$x^*_{c_2} = \{(c_1, 0), (c_1c_2, 0)\}$	$\{n_1, c_2\}$
r_2	e_2	$x^*_{e_2} = \{(c_1, 1), (c_1c_2, 0)\}$	$\{n_1, c_2, e_2\}$
n_1	e_1	$x^*_{e_1} = \{(\lambda, 1)\}$	$\{c_2, e_2, e_1\}$
c_2	r_2	$x^*_{r_2} = \{(c_1, 1), (c_1c_2, 0), (c_1c_2c_2, 0)\}$	$\{e_2, e_1, r_2\}$
e_2	n_2	$x^*_{n_2} = \{(c_1, 0)\}$	$\{e_1, r_2, n_2\}$
e_2	n_1	—	$\{e_1, r_2, n_2\}$
e_1	—	—	$\{r_2, n_2\}$
		. . .	

The next steps of the algorithm update $x^*_{r_2}$, $x^*_{c_2}$, $x^*_{n_2}$, $x^*_{e_2}$ in similar fashion, adding new entries with increasingly longer call strings, up to a string $c_1c_2c_2c_2c_2c_2$, but none of $x^*_{r_1}$, $x^*_{c_1}$, $x^*_{n_1}$, or $x^*_{e_1}$ is ever modified. Final x^* values for the blocks appearing in our example are:

$$x^*_{r_2} = x^*_{e_2} = \{(c_1, 1), (c_1c_2, 0), (c_1c_2c_2, 0), \ldots, (c_1c_2c_2c_2c_2c_2, 0)\}$$
$$x^*_{c_2} = \{(c_1, 0), (c_1c_2, 0), \ldots, (c_1c_2c_2c_2c_2c_2, 0)\}$$
$$x^*_{n_2} = \{(c_1, 0), (c_1c_2, 0), \ldots, (c_1c_2c_2c_2c_2, 0)\} \qquad (\ne x^*_{c_2}, \text{ by the way})$$

An x'' solution can now easily be computed; of course, this is identical to the solutions obtained by previous methods.

Note that in this example there was no need to maintain call strings of length up to 6 (length 2 would have sufficed). However, to derive correct information in the example depicted as Fig. 7-2, we need call strings in which one call appears three times.

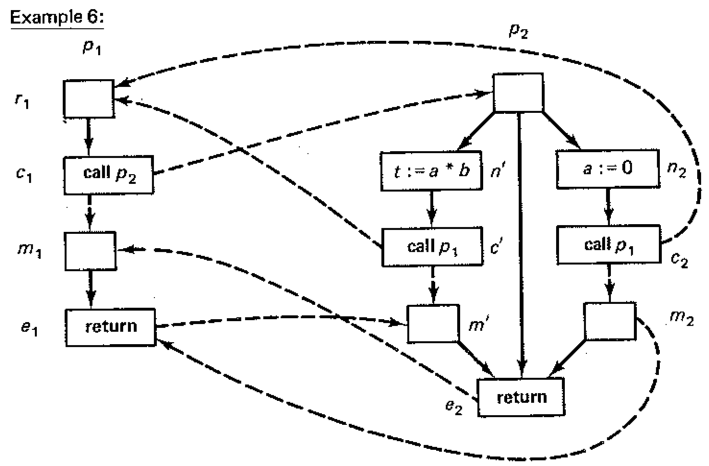

Figure 7-2

The shortest path in Fig. 7-2 showing that $a * b$ is *not* available at m' is $q = (r_1, c_1, r_2, n', c', r_1, c_1, r_2, n_2, c_2, r_1, c_1, r_2, e_2, m_1, e_1, m_2, e_2, m_1, e_1, m')$, in which c_1 appears three times before any of the calls in q is completed.

It is an interesting and challenging problem to find, for a given flow graph, by some preliminary analysis, an optimal set Γ_0 of call strings needed to perform some particular interprocedural data flow analysis without losing information.

7-6. AN APPROXIMATIVE CALL-STRING APPROACH

In this section we present a modification of the call-string approach developed in Section 7-4, which yields a convergent algorithm for any data flow analysis, even though this algorithm may in general fail to produce precisely the desired (meet-over-all-interprocedurally-valid-paths) solution. However, the output of the algorithm to be presented will always be an underestimated (and hence safe) solution. This compromise, which is useful even when L is finite, can make the call-string approach much more efficient.

Moreover, if L is infinite, or if F is not bounded or does not admit compact representation, then this modified approach is one of the very few ways to perform interprocedural analysis that we know of.

Three things should be kept in mind when evaluating any approximative approach to an interprocedural data flow problem: (1) Even in intraprocedural analysis, a meet-over-all-paths solution is itself an underestimation to the "true" run-time situation, since many of the static execution paths which enter into such an analysis may not be executable; (2) many data flow analyses whose semilattices L are not finite are also not distributive [Kam77, Shar78a], so that even the intraprocedural iterative solution of the data flow equations may underestimate the meet-over-all-paths solution; and (3) in nondistributive cases, the meet-over-all-paths solution may not be calculable (cf. [Hech77] for details).

By analyzing the abstract approach presented in Section 7-4, we can easily see that the convergence (and efficiency) of the call-strings approach depends primarily on Γ. Convergence can be ensured in general only if Γ is finite; and the smaller Γ is, the less complex the algorithm becomes. This observation motivates the approach that we propose in this section, whose general outline is as follows.

Choose some finite (preferably rather small) set $\hat{\Gamma}$ which is closed under a binary operation $*$ and has a left identity with respect to this operation. (We suggest that in practice $*$ be associative and noncommutative, but the general description given below will not assume this.) As in Section 7-4, let Γ denote the set of all call strings. Choose an "encoding" map σ which maps each call block to some element of $\hat{\Gamma}$. Using $*$, we can extend σ to Γ by putting $\sigma(\gamma) = \sigma(c_1) * \sigma(c_2) * \ldots * \sigma(c_j)$ (computed left to right) for each $\gamma = (c_1, c_2, \ldots, c_j) \in \Gamma$. We also define $\sigma(\lambda)$ to be w, the left-identity of $\hat{\Gamma}$.

Let (L, F) be any (not necessarily distributive) data flow framework. We will define a modified data flow framework (L^*, F^*) in essentially the same way as we did in Section 7-4, but with some differences reflecting the nature of the approximative approach, as detailed below.

L^* is defined as $L^{\hat{\Gamma}}$. All the observations made in Section 7-4 concerning L^* still apply, only now L^* is bounded since $\hat{\Gamma}$ has been assumed to be finite.

As before, in order to define F^*, we first define an updating operation between encoded call strings and edges in E^*. This updating operation is now more complex than that defined earlier, and need not be one-to-one and single-valued any more. It is therefore best described by assigning to each edge $(m, n) \in E^*$ a relation $R_{(m,n)}$ in $\hat{\Gamma}$. Essentially, $R_{(m,n)}$ is the identity relation for each $(m, n) \in E^0$ and for each call edge (m, n) and its corresponding return edge (m', n') we have $R_{(m',n')} = R^{-1}_{(m,n)}$. Then a path $(n_1, n_2, \ldots, n_k)$ will be considered to be *acceptable* if and only if $R_{(n_1,n_2)} \circ R_{(n_2,n_3)} \circ \ldots \circ R_{(n_{k-1},n_k)} \neq \varnothing$. To make these relations bear some meaningful relationship to the updating map $\circ$ defined in Section 7-4, we first define for each $(m, n) \in$

E^* a relation $I_{(m,n)}$ in Γ, so that for each $\gamma_1, \gamma_2 \in \Gamma$, $\gamma_1 I_{(m,n)} \gamma_2$ iff $\gamma_2 = \gamma_1 \circ (m, n)$, and then require the relation $\sigma \circ R_{(m,n)} \circ \sigma^{-1}$ to contain the relation $I_{(m,n)}$. This condition will guarantee that every interprocedurally valid path is also acceptable by our encoding scheme, but not necessarily vice versa.

To make the above ideas more precise, we suggest the following construction to obtain such suitable relations:

Definition. For each procedure p in the program being analyzed, define $\mathrm{ECS}(p) = \{\sigma(\mathrm{CM}(q)) \colon q \in \mathrm{IVP}(r_1, r_p)\}$. This is the set of all encoded call strings which result from interprocedurally valid paths reaching the entry of p.

These sets can be calculated by a rather simple preliminary analysis based upon the following set of equations (where *main* denotes the main program, which is assumed to be nonrecursive):

$$\begin{aligned} &\mathrm{ECS}(main) = \{w\} \\ &\mathrm{ECS}(p) = \{\alpha * \sigma(c)\colon c \text{ is a call to } p \text{ from some procedure } p' \\ &\qquad\qquad \text{and } \alpha \in \mathrm{ECS}(p')\} \qquad \text{for } p \neq main \end{aligned} \tag{7-14}$$

After initializing each $\mathrm{ECS}(p)$ to $\varnothing$, for all $p \neq main$, these equations can be solved iteratively in a fairly standard way. (The iterative solution will converge because $\hat{\Gamma}$ is finite.) It is a simple matter to prove that the iterative solution yields the sets $\mathrm{ECS}(p)$ defined above.

Using the sets ECS, we now define the following objects: for each $n \in N^*$, a set of *interprocedurally acceptable* paths leading from the main entry to n, denoted by $\mathrm{IAP}(r_1, n)$; a modified set-valued map $\widehat{\mathrm{CM}}$ from $\bigcup_{n \in N^*} \mathrm{IAP}(r_1, n)$ to $2^{\hat{\Gamma}}$; and a modified relation-valued map $R\colon E^* \to 2^{\hat{\Gamma} \times \hat{\Gamma}}$. For each $(m, n) \in E^*$, $R_{(m,n)}$ is a relation in $\hat{\Gamma}$, so that for each $\alpha, \beta \in \hat{\Gamma}$ we have

$$\alpha R_{(m,n)} \beta \text{ iff } \begin{cases} \alpha = \beta \in \mathrm{ECS}(p) & \text{if } (m, n) \in E^0_p \text{ for some procedure } p \\ \alpha \in \mathrm{ECS}(p) \text{ and } \beta = \alpha * \sigma(m) & \text{if } (m, n) \text{ is a call edge from procedure } p \\ \beta \in \mathrm{ECS}(p) \text{ and } \alpha = \beta * \sigma(c) & \text{if } (m, n) \text{ is a return edge corresponding to a call edge from a call block } c \text{ in procedure } p \end{cases}$$

Using these relations, we define the map $\widehat{\mathrm{CM}}$, so that for each $n \in N^*$ and each path $q \in \mathrm{path}_{G^*}(r_1, n)$ of the form $(r_1, s_2, s_3, \ldots, s_{k-1}, n)$ we have

$$\widehat{\mathrm{CM}}(q) = R_{(r_1,s_2)} \circ R_{(s_2,s_3)} \circ \ldots \circ R_{(s_{k-1},n)}\{w\}$$

Finally, for each $n \in N^*$ we define

$$\mathrm{IAP}(r_1, n) = \{q \in \mathrm{path}_{G^*}(r_1, n) : \widehat{\mathrm{CM}}(q) \neq \varnothing\}$$

The intuitive meaning of these concepts can be explained as follows: Since we have decided to record the actual call string by a homomorphism CM of paths into a finite set $\hat{\Gamma}$, it is inevitable that we will also admit paths which are not in $\mathrm{IVP}(r_1, n)$. Thus $\mathrm{IAP}(r_1, n) \supseteq \mathrm{IVP}(r_1, n)$ and will also contain paths which the encoding CM cannot distinguish from valid IVP paths. In particular, some returns to other than their originating calls will have to be admitted.

Having defined IAP, $\widehat{\mathrm{CM}}$, and R, we next define F^* in essentially the same manner as in Section 7-4. Specifically, for each $(m, n) \in E^*$ we define $f^*_{(m,n)}: L^* \to L^*$ as follows: For each $\xi \in L^*$, $\alpha \in \hat{\Gamma}$

$$f^*_{(m,n)}(\xi)(\alpha) = \bigwedge \{f_{(m,n)}(\xi(\alpha_1)) : \alpha_1 R_{(m,n)} \alpha\}$$

where, by definition, an empty meet yields Ω.

F^* is now constructed from the functions $f^*_{(m,n)}$ exactly as before. The heuristic significance of this definition is the same as in Section 7-4, only now the "tag" updating which occurs when propagation takes place along an interprocedural edge involves less extensive and less precise information. The modified updating operation that has just been defined can be both one-to-many and many-to-one, possibilities which are both reflected in the above formula. It is easy to verify that both monotonicity and distributivity are preserved as we pass from (L, F) to (L^*, F^*).

Next we associate with (L^*, F^*) the data flow problem of determining the maximal fixed-point solution of the equations

$$\begin{aligned} x^*_{r_1} &= \{(w, 0)\} \\ x^*_n &= \bigwedge_{(m,n) \in E^*} f^*_{(m,n)}(x^*_m) \qquad n \in N^* - \{r_1\} \end{aligned} \tag{7-15}$$

As previously, a solution of these equations can be obtained by standard iterative techniques. Once this solution has been obtained, we make the following final calculation:

$$\hat{x}_n = \bigwedge_{\alpha \in \hat{\Gamma}} x^*_n(\alpha) \tag{7-16}$$

The techniques of Section 7-4 can now be applied to analyze the procedure just described. Theorem 7-4.3 retains its validity, if restated as follows:

Theorem 7-6.2.

1. If (L, F) is distributive, then, for each $n \in N^*$, $x^*_n = y^*_n \equiv \bigwedge \{f^*_p(x^*_{r_1}) : p \in \mathrm{path}_{G^*}(r_1, n)\}$.
2. If (L, F) is monotone, then, for each $n \in N^*$, $x^*_n \leq y^*_n$.

Instead of Lemma 7-4.5, the following variant applies:

Lemma 7-6.3. Let $n \in N^*$, $p \in \mathrm{path}_{G^*}(r_1, n)$ and $\alpha \in \hat{\Gamma}$. Then $f^*_p(x^*_{r_1})(\alpha)$ is defined iff $\alpha \in \widehat{\mathrm{CM}}(p)$, in which case $f^*_p(x^*_{r_1})(\alpha) = f_p(0)$.

Proof. By induction on the length of p. The assertion is obvious if p is the null execution path. Suppose that it is true for all paths with length $<k$ and let $p = (r_1, s_2, \ldots, s_k, n) \in \mathrm{path}_{G^*}(r_1, n)$ be a path of length k. Let $p_1 = (r_1, s_2, \ldots, s_k)$. Then for each $\alpha \in \hat{\Gamma}$ we have

$$\begin{aligned} f^*_p(x^*_{r_1})(\alpha) &= f^*_{(s_k,n)}[f^*_{p_1}(x^*_{r_1})](\alpha) \\ &= \bigwedge \{f_{(s_k,n)}[f^*_{p_1}(x^*_{r_1})(\alpha_1)] : \alpha_1 R_{(s_k,n)} \alpha\} \end{aligned}$$

Thus $f^*_p(x^*_{r_1})(\alpha)$ is defined iff there exists $\alpha_1 \in \hat{\Gamma}$ such that $\alpha_1 R_{(s_k,n)} \alpha$ and $f^*_{p_1}(x_{r_1})(\alpha_1)$ is defined. By inductive hypothesis, this is true iff there exists $\alpha_1 \in \widehat{\mathrm{CM}}(p_1)$ and $\alpha_1 R_{(s_k,n)} \alpha$, and, by the definition of $R_{(s_k,n)}$ and $\widehat{\mathrm{CM}}$, this last assertion is true iff $\alpha \in \widehat{\mathrm{CM}}(p)$. Hence, applying the inductive hypothesis again, $f^*_{p_1}(x^*_{r_1})(\alpha_1) = f_{p_1}(0)$, for all α_1 appearing in the above meet, so that this meet equals $f_{(s_k,n)}[f_{p_1}(0)] = f_p(0)$. ■

Remark: As previously noted, and can be seen, e.g., from the proof of the last lemma, use of an encoding scheme creates chances for propagation through paths which are not interprocedurally valid. However, our lemma shows that even if an execution path q is encoded by more than one element of Γ, all these "tags" are associated with the same information, namely $f_q(0)$. Thus information is propagated correctly along each path, only more paths are now acceptable for that propagation. These observations will be made more precise in what follows.

Lemma 7-6.4. For each $n \in N^*$, $\mathrm{IVP}(r_1, n) \subseteq \mathrm{IAP}(r_1, n)$.

Proof. Let $q \in \mathrm{IVP}(r_1, n)$ for some $n \in N^*$. We will show, by induction on the length of q, that $\sigma(\mathrm{CM}(q)) \in \widehat{\mathrm{CM}}(q)$, so that, by Lemma 7-6.1, $q \in \mathrm{IAP}(r_1, n)$.

Our assertion is obvious if q is the null execution path. Suppose it is true for all paths whose length is less than some $k \geq 0$, and let $n \in N^*$, $q \in \mathrm{IVP}(r_1, n)$ whose length is k. Write $q = q_1 \| (m, n)$. By inductive hypothesis, $\sigma(\mathrm{CM}(q_1)) \in \widehat{\mathrm{CM}}(q_1)$. Now, three cases are possible:

1. $(m, n) \in E^0$. In this case $\widehat{\mathrm{CM}}(q) = \widehat{\mathrm{CM}}(q_1)$ and $\mathrm{CM}(q) = \mathrm{CM}(q_1)$ so that $\sigma(\mathrm{CM}(q)) \in \widehat{\mathrm{CM}}(q)$.
2. (m, n) is a call edge. Then, by definition, $\widehat{\mathrm{CM}}(q)$ contains $\sigma(\mathrm{CM}(q_1)) * \sigma(m) = \sigma(\mathrm{CM}(q))$.

3. (m, n) is a return edge. Let (c', r_p) denote the corresponding call edge. Since $q \in \text{IVP}(r_1, n)$, q can be decomposed as $q' \| (c', r_p) \| q'' \| (m, n)$, where $q' \in \text{IVP}(r_1, c')$ and $q'' \in \text{IVP}_0(r_p, m)$. It is evident from the definitions of the quantities involved that that $\text{CM}(q) = \text{CM}(q')$ and that $\text{CM}(q_1) = \text{CM}(q') \| (c')$. Hence $\sigma(\text{CM}(q_1)) = \sigma(\text{CM}(q)) * \sigma(c')$. It thus follows that $\sigma(\text{CM}(q))$ is a member of the set $\{\beta \in \text{ECS}(p) \mid \beta * \sigma(c') = \sigma(\text{CM}(q_1))\}$ which, by definition, is a subset of $\widehat{\text{CM}}(q)$. ■

We can now state an analog of Theorem 7-4.6:

Theorem 7-6.5.

1. If (L, F) is a distributive data flow framework, then, for each $n \in N^*$
$$\hat{x}_n = \bigwedge \{f_p(0) : p \in \text{IAP}(r_1, n)\} \leq y_n$$
2. If (L, F) is only monotone, then, for each $n \in N^*$
$$\hat{x}_n \leq \bigwedge \{f_p(0) : p \in \text{IAP}(r_1, n)\} \leq y_n$$

Proof.

1. Let $\alpha \in \hat{\Gamma}$. By Theorem 7-6.2 and Lemmas 7-6.1 and 7-6.3, we have
$$x_n^*(\alpha) = \bigwedge \{f_p^*(x_{r_1}^*)(\alpha) : p \in \text{path}_{G^*}(r_1, n)\}$$
$$= \bigwedge \{f_p(0) : p \in \text{IAP}(r_1, n), \alpha \in \widehat{\text{CM}}(p)\}$$
Thus, by Eq. (7-16)
$$\hat{x}_n = \bigwedge_{\alpha \in \hat{\Gamma}} x_n^*(\alpha) = \bigwedge \{f_p(0) : p \in \text{IAP}(r_1, n)\}$$
By Lemma 7-6.4, this is
$$\leq \bigwedge \{f_p(0) : p \in \text{IVP}(r_1, n)\} = y_n$$
proving (1).
2. Can be proved in a manner completely analogous to the proof of (1), using part (2) of Theorem 7-6.2. ■

Thus $\{\hat{x}_n\}_{n \in N^*}$ is an underestimation of the meet-over-all-paths solution $\{y_n\}_{n \in N^*}$. The degree of underestimation depends on the deviation of $\text{IAP}(r_1, n)$ from $\text{IVP}(r_1, n)$, and this deviation is in turn determined by the choice of $\hat{\Gamma}$, $*$, and σ. The most extreme underestimation results if we let $\text{IAP}(r_1, n) = \text{path}_{G^*}(r_1, n)$ for all $n \in N^*$, i.e., define $\hat{\Gamma} = \{w\}$, $w * w = w$, and let σ map all calls to w. If we do this, then the resulting problem is essentially equivalent to a purely intraprocedural analysis, in which procedure calls and returns are interpreted as mere branch instructions.

Another more interesting encoding scheme is as follows. Choose some integer $k > 1$, and let $\hat{\Gamma}$ be the ring of residue classes modulo k. Let $m > 1$ be another integer. For each $\alpha_1, \alpha_2 \in \hat{\Gamma}$, define $\alpha_1 * \alpha_2 = m \cdot \alpha_1 + \alpha_2 (\text{mod } k)$. Let σ be any map which maps call blocks to values between 0 and $m - 1$ (preferably in a one-to-one way). In this scheme, call strings are mapped into a base m representation modulo k of some encoding of their call blocks. Note that if $k = \infty$, i.e., if we operate with integers rather than in modular arithmetic, then $\hat{\Gamma}$ and Γ are isomorphic, with $\mathfrak{b}$ corresponding to concatenation. If $k = m^j$, for some $j \geq 1$, and σ is one-to-one and does not map any call block to 0, then the encoding scheme just proposed can roughly be described as follows: Keep only the last j calls within each call string. As long as the length of a call string is less than j, update it as in Section 7-4, However, if q is a call string of length j, then, when appending to it a call edge, discard the first component of q and add the new call block to its end. When appending a return edge, check if it matches the last call in q, and, if it does, delete this call from q and add to its start all possible call blocks which call the procedure containing the first call in q. This approximation may be termed a *call-string suffix approximation.*

At present we do not have available a comprehensive theory of the proper choice of an encoding scheme. Appropriate choice of such a scheme may depend on the program being analyzed, and reflects the trade-off between tolerable complexity of the interprocedural analysis and some desired level of accuracy.

7-7. CONCLUSION

In this chapter we have studied in some detail two basic approaches to interprocedural analysis of rather general data flow problems. We have seen that by requiring the associated semilattice L to be finite, both approaches yield convergent algorithms which produce the "sharpest" interprocedural information, in a natural sense.

The main concern has been to introduce a comprehensive theory of interprocedural data flow analysis for general frameworks. Subsequent research in this area should address itself to more pragmatic issues that arise when trying to implement our approaches. Some of these issues are:

(a) *Pragmatic implementation of the functional approach for bit-vector problems.* These data flow frameworks are amenable to elimination techniques, which are more efficient than iterative techniques. However, our basic way of solving Eqs. (7-4) is iterative in nature and hence is not optimal for these problems. One would mainly like to come up with an algorithm which incorporates standard intraprocedural elimination techniques, such as interval analysis, in a modular manner, which will enable us to implement the functional approach as an extension of already existing intraprocedural algorithms rather than as a completely different algorithm.

In addition, one might wish to study the efficiency of such an implementation, bearing in mind that recursion is a somewhat rare phenomenon in actual programs, and that co-recursion is much rarer. This issue is closely related to Allen's approach of processing procedures in "inverse invocation order" (see also [Rose79] for a similar observation). However, careful refinement of this method is required to handle recursion. Additional gain might be achieved by processing "offline" parts of the flow graph which are call-free, so that one does not have to repeat all the intraprocedural processing whenever an interprocedural effect is propagated. One possible approach to this problem, which, however, is probably not the best possible one for implementation, is indicated in [Rose79, Section 8].

(b) *Pragmatic implementation of more complex interprocedural data flow problems.* If the relevant framework is not amenable to elimination, then the functional approach may be inadequate for such a problem. Moreover, some commonly occurring complex data flow problems, such as constant propagation [Hech77], type analysis [Tene74b, Jone76], value flow [Schw75b] or range analysis [Harr77a], are usually solved by algorithms which make use of the *use-definition map* [Alle69] in a way which propagates information only to points where it is actually needed. As indicated in [Shar77], interprocedural extension of such algorithms calls for some proper interprocedural extension of the use-definition map itself. It seems that such an extension can be based on the call-strings approach (or the approximative call-strings approach), but exact details have yet to be worked out.

(c) *Extending our approaches to handle reference parameters.* Here the problem of "aliasing" (i.e., temporary equivalence of two program variables during a procedure call) arises, which complicates matters considerably if "sharpest" information is still to be obtained. Major work in this area has been done by Rosen [Rose79].

(d) *Extending the ideas of the call-strings approach methods which take into account more semantic restrictions on the execution flow.* Thus only flow paths which satisfy such restrictions would be traced during analysis. The call-return pattern of interprocedural flow is but one such possible restriction (though a very important one). For example, one might also keep track of the values of boolean flags which control intraprocedural branches. Current research in such directions by Holley and Rosen at IBM seems quite promising. (We are indebted to Barry K. Rosen for some stimulating discussion concerning the above-mentioned research.)

The present chapter has been motivated by the research on the design and implementation of an optimizing compiler for the SETL programming language at Courant Institute, New York University. SETL is a very-high-level language [Schw75d] which fits into our interprocedural model; i.e., parameters are called by value and no procedure variables are allowed. Active research is now under way to implement the approaches suggested in this chapter in the optimizer of our system, as discussed in (a) and (b) above.