J. L. Becerril
J. Bondia
R. Casajuana
F. Valer

Grammar Characterization of Flowgraphs

An extension of the scheme grammar concept given by Urschler is formalized. It is also shown that, in the usual hierarchy of the theory of formal languages, the language generated by the scheme grammar is regular (type 3). The last section gives the description of a system for the automatic structuring of programs, which applies these concepts to the Mills algorithm with some modifications.

Introduction

Several attempts to handle program schemes as mathematical entities have been made in recent years, in an effort to develop algebraic techniques covering program optimization, debugging, verification, and classification [1–3]. The studies in this area tend to follow two different approaches, according to the two fundamental aspects of a program, *i.e.*, its control flow [4] and its semantic contents [2].

In this paper, which corresponds to the first approach, we deal with an extension and formalization of the scheme grammar concept given by Urschler [5], obtaining a series of results applicable to program schemes. In particular, common modules across the scheme are detected, and detailed information is obtained about the scheme structure, the action range of condition nodes, etc. It is also shown that, in the usual hierarchy of the theory of formal languages, the language generated by the scheme grammar is regular.

This paper is divided into four sections. The first one defines some fundamental concepts, as well as a few mathematical techniques needed in the rest of the work. A similar approach can be found in [6]. More concretely, the postdominance relation is defined, which permits a program scheme to be given the poset (partially ordered set) structure, showing the existence and uniqueness of a minimal element in the set of postdominators called immediate postdominator (ipd).

The second section introduces the two g-chains associated with a condition node (the ordered chains of nodes of the graph whose first element is one of the two successors of n and the rest of which are obtained by iterative application of an ipd operation until a node is reached whose ipd equals the ipd of n) and the main g-chain (similarly defined for the starting node). Some basic properties of g-chains are shown.

In the third section, after defining the scheme grammar based on the g-chains, its consequences are studied; it is shown that the language generated is regular and coincides with the set of possible paths through the program scheme.

In order to develop possible applications based on the concepts of the g-chain, scheme grammar, and language, some implementation has been carried out for their evaluation. Along these lines a system has been designed and implemented for the automatic structuring of programs, which, besides these concepts, applies, with some modifications, the Mills algorithm [7]. A general description of this system is given in the last section.

Flowgraphs

Definition 1

We define a flowgraph as the 2-tuple $E = (N, F)$ with the following properties:

Copyright 1980 by International Business Machines Corporation. Copying is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract may be used without further permission in computer-based and other information-service systems. Permission to *republish* other excerpts should be obtained from the Editor.

- $N = N' \cup \{\triangle\}$, $\triangledown \in N'$, $\triangle \notin N'$, where $\triangledown$ and $\triangle$ are, respectively, the entry and exit nodes.
- $F: N' \to P(N - \{\triangledown\})$, *i.e.*, F maps N' into the set of all subsets of $N - \{\triangledown\}$, where the cardinality of the set $F(n)$, denoted by $\#F(n)$, is 1 or 2. If $\#F(n) = 1$, we say n is an action node. The set of action nodes is denoted by N_1. If $\#F(n) = 2$, $F(n)$ is reducible to two injective mappings $F(+, n)$, $F(-, n)$; in this case we say n is a condition node. The set of condition nodes is denoted by N_2.

By convention, in this paper $F(., n)$ denotes either $F(+, n)$ or $F(-, n)$. The rth power of F is

$$F^r(x) = \bigcup_{y \in F^{r-1}(x)} F(y),$$

with $F^0(x) = \{x\}$. The transitive closure of F in x, defined by

$$\bigcup_{r=0}^{\infty} F^r(x),$$

is denoted by $\hat{F}(x)$. The definition of $\hat{F}^{-1}(x)$ is analogous.

Definition 2
A flowgraph is said to be proper if and only if for every $n \in N' - \{\triangledown\}$ both $\triangledown \in \hat{F}^{-1}(n)$ and $\triangle \in \hat{F}(n)$. In this paper we consider only proper flowgraphs.

Definition 3
A flowgraph path is said to be elementary if and only if it does not contain repeated nodes.

Definition 4
We define a program scheme $H = (E, S)$ as the pair consisting of the flowgraph E and the function S. The function S maps $N' - \{\triangledown\}$ into the set $An \cup Cn$ with the rules $S(n) \in An \Leftrightarrow n \in N_1$, $S(n) \in Cn \Leftrightarrow n \in N_2$. Both An and Cn are finite and mutually exclusive sets, called, respectively, action and condition sets.

Since all the work is done only on the control structure, the definition of a program scheme has very low semantic contents, in order not to complicate unnecessarily the process.

- *Postdominance in flowgraphs*

Definition 5
It is said that m postdominates n ($n \leq m$) if for every elementary path $c = (n, \triangle)$ we have $m \in c$, $n, m \in N$.

Lemma 6
The pair $(E, \leq)$ is a poset (partially ordered set).

The binary relation $\leq$ defines a partial ordering on E, since it is

- Reflexive. For any $n \in N$ and any $c = (n, \triangle)$, we have $n \in c$.
- Antisymmetric. Assume that $n \neq m$; then, if $n \leq m$, we have, for every elementary path $c = (n, \triangle)$, $m \in c$, *i.e.*, $c = (n, \cdots, m, \cdots, \triangle)$. If c is elementary, the path $c' = (m, \triangle)$ is elementary with $n \in c'$ ($m \leq n$), which implies that c is not an elementary path, a contradiction. Hence $m = n$.
- Transitive. If $n \leq m$ and $m \leq p$, then $n \leq p$. Consider an elementary path $c = (n, \triangle)$; if $n \leq m$, we have $m \in c$, *i.e.*, $c = (n, \cdots, m, \cdots, \triangle)$. If c is elementary, the path $c' = (m, \triangle)$ is elementary. By hypothesis $m \leq p$, which implies that $p \in c'$. Hence $p \in c$ for every $c = (n, \triangle)$, *i.e.*, $n \leq p$.

Definition 7
A node $n' \neq n$ is an immediate postdominator (ipd) of the node n if and only if $n \leq n'$ and, if $n \leq n''$, then $n' \leq n''$. In other words, the ipd of a node is the first junction point of all the elementary paths going from it to the exit node.

The existence and uniqueness of an ipd for every $n \in N'$ are shown by applying Zorn's lemma [8] to the finite set $P(n) = \{m;\ m \neq n,\ n \leq m\}$, the ipd being the minimal element in $P(n)$. From the definition we have that the ipd of an action node is always its successor.

G-chains

Definition 8
We define the main g-chain as the chain $C(\triangledown) = (p_i)$ with the properties $p_1 = F(\triangledown)$ and $p_i = \text{ipd}(p_{i-1})$. Obviously the last element is $\triangle$.

Definition 9
Let us consider a condition node n and the . direction; we define the g-chain with head n, $C(n, .) = (n_i)$, by means of the following rules:

- If $F(., n) = \text{ipd}(n)$, then $C(n, .)$ is the empty chain.
- Otherwise, $n_1 = F(., n)$, and $n_i = \text{ipd}(n_{i-1})$.
- The last element and n have the same ipd.

The consistency of these definitions is guaranteed by the existence and uniqueness of the ipd for every $n \in N'$.

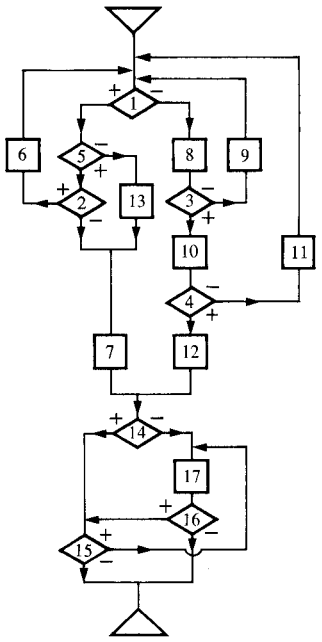
- *Example*

The g-chains of the flowgraph of Fig. 1 are listed beside it.

Lemma 10
Let $C(n, .)$ be a g-chain with head n. Then $C(n, .)$ has the following properties:

1. If $m \in C(n, .)$, then $m \in \hat{F}(n)$.

Effectively, since $\leq$ is transitive, $n_1 \leq m$, $m \in C(n, .)$. Furthermore, $n_1 = F(., n)$; hence, $m \in \hat{F}(n)$.



$C(\triangledown)$ = 1 14 $\triangle$
$C(1, +)$ = 5
$C(1, -)$ = 8 3
$C(2, +)$ = 6 1
$C(2, -)$ = 7
$C(3, +)$ = 10 4
$C(3, -)$ = 9 1
$C(4, +)$ = 12
$C(4, -)$ = 11 1
$C(5, +)$ = 2
$C(5, -)$ = 13 7
$C(14, +)$ = 15
$C(14, -)$ = 17 16
$C(15, +)$ = 17 16
$C(15, -) = \varnothing$
$C(16, +)$ = 15
$C(16, -) = \varnothing$

Figure 1 Flowgraph and its g-chains.

2. If $m \in C(n, .)$, then $m \leq \text{ipd}(n)$.

Let $m \in C(n, .)$. If the last element of $C(n, .)$ is p, since $\leq$ is transitive, every m verifies $m \leq p$. Since $p \leq \text{ipd}(p) = \text{ipd}(n)$, then $m \leq \text{ipd}(n)$.

3. If $m, p \in C(n, .)$, then $m \neq p$.

If the ith element were m and the jth one ($j > i$) were also m, then, applying the ipd operation $j - i$ times to m, we would obtain m again, contradicting the ipd definition.

4. If $n \in C(n, .)$, then $C(n, .) = A\ n$.

This stems directly from the g-chain definition.

Scheme grammar and language

A first application of g-chains is the scheme grammar, a formalized extension of the concept given by Urschler in [5]. Before the definition, let us introduce a set $\underline{Cn} = \{\underline{b}; b \in \underline{Cn}\}$, and a mapping $\underline{S}$, which associates the symbol $\underline{b}$ with the condition node n, where $b = S(n)$.

We denote by $[n]$, $S(n)$ if n is an action node or n if it is a condition node. By convention, $[\triangle] = \triangle$. This notation is trivially extended to a g-chain $C(n, .) = n_1 \cdots n_r$ applying the rule $[C(n, .)] = [n_1] \cdots [n_r]$. In this section, we denote by P either S or $\underline{S}$.

Definition 11
The grammar associated with a program scheme H is the 4-tuple $G(H) = (Vn, Vt, R, \triangledown)$, where

- The set of nonterminal symbols Vn is $\{\triangledown\} \cup N_2$.
- The set of terminal symbols Vt is $An \cup Cn \cup \underline{Cn} \cup \{\triangle\}$.
- R is the set of production rules:
- $\triangledown \rightarrow [C(\triangledown)]$
- $n \in N_2$ $\quad n \rightarrow S(n)[C(n, +)]$ and $n \rightarrow \underline{S}(n)[C(n, -)]$.

This grammar is context-free in the usual hierarchy of formal language theory.

Lemma 12
If there exists a derivation in $G(H)$ of the form $\triangledown \overset{*}{\Rightarrow} A\ n\ m\ X$, then $m = \text{ipd}(n)$.

This can be shown by induction on the number of steps of the derivation:

- For $k = 1$, it is obvious from the main g-chain definition.
- Let it be true for an integer k, *i.e.*, $\triangledown \overset{k}{\Rightarrow} A\ n_i\ n_{i+1}\ X$, $n_{i+1} = \text{ipd}(n_i)$. To obtain a derivation of the desired form in the step $k + 1$, we must apply a substitution like $n_i \rightarrow C\ n_k$. The corresponding derivation would be $\triangledown \overset{k+1}{\Rightarrow} A\ C\ n_k\ n_{i+1}\ X$. From the g-chain construction rules we have ipd (n_i) = ipd (n_k). From this and the induction hypothesis $n_{i+1} = \text{ipd}(n_k)$.

Lemma 13
If there exists a derivation in $G(H)$ of the form $\triangledown \overset{*}{\Rightarrow} A\ P(p)\ P(q)\ Y$, then q is a successor of p.

Such a derivation can only be obtained in two cases:

$\triangledown \overset{*}{\Rightarrow} A\ n\ X \Rightarrow A\ B\ P(p)\ P(q)\ Y\ X$
$\triangledown \overset{*}{\Rightarrow} A\ n\ m\ X \Rightarrow A\ B\ P(p)\ P(q)\ Y\ X.$

In the first one, the rule used is $n \rightarrow B\ P(p)\ P(q)\ Y$, and we can consider two possibilities:

- B is the empty string. By definition of the grammar $n = p$ and $P(q)$ is the P image of the successor of p in the considered direction.
- B is not empty. Then, due to the same definition, $q = \text{ipd}(p)$ and p is an action node; hence q is a successor of p.

In the second case, the rules must be $n \rightarrow B\ P(p)$ and $m \rightarrow P(q)\ Y$. From Lemma 12, $m = \text{ipd}(n)$, and from the grammar definition ipd (n) = ipd (p) and $q = m$. Hence, $q = \text{ipd}(p)$. Since p is an action node, we deduce that q is a successor of p.

Corollary
If $P(n_1) \cdots P(n_k)\ \triangle$ is a word of the language associated with $G(H)$, then $\triangledown\ n_1 \cdots n_k\ \triangle$ is a path of the considered flowchart.

This is self-evident if the previous lemma is applied iteratively.

Lemma 14
Let us assume $n \overset{*}{\Rightarrow} A$ is a derivation in $G(H)$. For every condition node m in A, ipd (n) is a postdominator of m $(m \leq \text{ipd } (n))$.

This can be shown by induction on the number of steps of the derivation:

- For $k = 1$, it is obvious from the grammar definition.
- Let it be true for an integer k, *i.e.*, $n \overset{k}{\Rightarrow} C\ p\ X$, $p \leq$ ipd (n). In the step $k + 1$ we have $n \overset{k+1}{\Rightarrow} C\ D\ q\ Y\ X$ with ipd $(p) \leq$ ipd (n), and from the g-chain definition $q \leq$ ipd (p). Since the postdominance relation is transitive, we have $q \leq$ ipd (n).

Corollary
If $m \in A$, then ipd $(m) \leq$ ipd (n). If ipd $(m) \neq$ ipd (n), it follows just by considering, instead of m, the ipd (m) that belongs to A.

Lemma 15
If there exists a derivation $n \overset{*}{\Rightarrow} A\ m\ X$ in which ipd $(n) =$ ipd (m), then X is the empty string.

It can be shown by induction on the number of steps of the derivation:

- For $k = 1$ it follows from the grammar definition.
- Let it be true for an integer k, *i.e.*, if $n \overset{k}{\Rightarrow} C\ q\ Y$ with ipd $(n) =$ ipd (q), then Y is empty. Let $q \rightarrow D\ m\ Z$ be the rule applied in the $(k + 1)$th step. Then $n \overset{k+1}{\Rightarrow} C\ D\ m\ Z\ Y$. If ipd $(m) =$ ipd (n), applying the induction hypothesis, we have ipd $(m) =$ ipd (q). Then, from the grammar definition, Z is the empty string.

Lemma 16
For every program scheme H, $G(H)$ is not self-embedding [9].

Since $Vn = \{\triangledown\} \cup N_2$ and we know that a derivation of the form $\triangledown \overset{*}{\Rightarrow} A\ \triangledown\ X$ is not possible, it is sufficient to show that, for every condition node n, there cannot exist derivations $n \overset{*}{\Rightarrow} A\ n\ X$ such that A and X are both different from the empty string.

The only cases leading to such a situation are:

- There exists a production rule $n \rightarrow A\ n\ X$, with A and X different from the empty string. This is not possible from Lemma 10.
- There exists a production rule $n \rightarrow A\ \triangledown\ X$, with A and X different from the empty string. This is not possible, since, by definition, there cannot exist paths going from a node to the entry node.
- There exists a derivation of the type $n \overset{*}{\Rightarrow} A\ m\ X \overset{*}{\Rightarrow} A\ B\ n\ Y\ X$. In this case, Y and X must be empty. Effectively, from the corollary of Lemma 14 ipd $(m) \leq$ ipd (n). If $m \overset{*}{\Rightarrow} B\ n\ Y$, we have, by Lemma 14, that $n \leq$ ipd (m), which, together with the ipd definition, leads us to ipd $(n) \leq$ ipd (m). Since the postdominance relation is an antisymmetric one, ipd $(n) =$ ipd (m). Applying Lemma 15, we deduce that X must be the empty word. Having in mind the derivation $m \overset{*}{\Rightarrow} B\ n\ Y$, by analogous reasoning, we conclude that Y must also be the empty word.

Theorem 17
For every program scheme H, the language generated by $G(H)$ is regular.

It is immediate from Lemma 16 and a classical result from the theory of formal languages which states that every nonself-embedding, context-free grammar generates a regular language [9].

Structuring schemes using g-chains

Another possible application of the g-chain concept is the automatic structuring of flowcharts. In this section we describe the general guidelines of this application, emphasizing the simplicity of implementation. Consult [10] for more detailed information.

The action node A0 (A1) denotes a node in which the value 0 (1) is assigned to the variable A. Similarly, TA denotes the condition node in which the test A = 0 is performed.

Definition 18
Consider the flowgraph $E = (N, F)$; the module associated with a condition node n is defined as the subgraph of E, $Mn = (Nn, Fn)$, having the following properties:

- Fn is the restriction of F to the set Nn.
- Nn is the closure of the set succession $\{Un(k)\}$ defined by

$$Un(0) = \{n\},$$

$$Un(1) = C(n, +) \cup C(n, -),$$

$$Un(k) = \bigcup_{p \in N_2 \cap Un(k-1)} Up(1) \qquad k > 1.$$

The existence and uniqueness of this set is guaranteed by the fact that the flowgraph is finite and by the g-chain definition.

In the flowgraph of Fig. 1, there are two modules, $M1 = (N1, F1)$ and $M14 = (N14, F14)$, where $N1 = \{1, 2, \cdots, 13\}$ and $N14 = \{14, 15, 16, 17\}$.

Definition 19
A submodule is a subgraph of a module which is also a module. A module is said to be minimal if no other module is included in it.

In the flowgraph of Fig. 1, the minimal modules are $M1$ and $M14$.

Definition 20
Consider a module $Mn = (Nn, Fn)$. The minimal element of Nn with respect to the postdominance relation is called the head of the module.

In the flowgraph of Fig. 1, the head of $M1$ is the node 1, and the head of $M14$ is the node 14.

Definition 21
Let us consider a module with head n; we define the branch $B(n, .)$ of the module as the closure of the set succession $\{U\dot{n}(k)\}$ defined by

$$U\dot{n}(0) = C(n, .),\ U\dot{n}(k) = \Big\{ m \in \bigcup_{p \in N_2 \cap U\dot{n}(k-1)} Up(1);$$

$$\exists r > 0\ (F^{-1})^r(m) = F(., n)\ k = 1, 2, \cdots \Big\}.$$

The consistency of this definition is self-evident; besides, $B(n, .) \subset Nn$.

In the flowgraph of Fig. 1, the two branches of module $M1$ are $B(1, +) = \{2, 5, 6, 7, 13\}$ and $B(1, -) = \{3, 4, 8, 9, 10, 11, 12\}$. In the module $M14$, the branches are $B(14, +) = \{15\}$ and $B(14, -) = \{16, 17\}$.

Definition 22
A node m is said to be the entry node of a module Mn if and only if $m \notin Nn$ and $F(m) \cap Nn \neq \emptyset$.

The entry node of module $M1$ in Fig. 1 is $\triangledown$.

Definition 23
A skeleton S is an ordered set of nodes with their directions $n_1^{\cdot}\ A\ n_2^{\cdot} \cdots K\ n_k$ such that

- $n_k = n_1$.
- There exists a collection of g-chains $C(n_1, .) = A\ n_2, \cdots, C(n_{k-1}, .) = K\ n_k$.

Given a skeleton S, WS is the set of nodes belonging to S.

In Fig. 1, the skeletons are

$S1 = 1^+\ 5^+\ 2^+\ 6\ 1$
$S2 = 1^-\ 8\ 3^-\ 9\ 1$
$S3 = 1^-\ 8\ 3^+\ 10\ 4^-\ 11\ 1$
$S4 = 16^+\ 15^+\ 17\ 16$

Definition 24
Let us consider a set of skeletons Si $(i = 0, \cdots, k)$, such that

$$\bigcap_{i=0}^{k} WSi \neq \emptyset.$$

A condition node m is said to be the head of Si if it verifies

- $m \in \bigcap_{i=0}^{k} WSi$,
- m, with respect to the postdominance relation, is the minimal element of

$$\bigcap_{i=0}^{k} WSi.$$

Since the flowgraph is proper, there will be at least a condition node in every skeleton.

The existence and unicity of the skeleton head is guaranteed by Zorn's lemma.

In the flowgraph of Fig. 1, the skeleton heads are $H1 = H2 = H3 = 1$, $H4 = 15$.

Definition 25
An element n^+ $[n^-]$ is said to be an exit of a skeleton S if and only if $n \in WS$ and n^- $[n^+] \notin S$.

In Fig. 1, the skeleton exits are $E1 = \{1^-, 5^-, 2^-\}$, $E2 = \{1^+, 3^+\}$, $E3 = \{1^+, 3^-, 4^+\}$, $E4 = \{16^-, 15^-\}$.

Definition 26
An exit $n^{\cdot}$ is said to be real if and only if it does not appear in any other skeleton having the same head.

In Fig. 1, $RE1 = \{5^-, 2^-\}$, $RE2 = \emptyset$, $RE3 = \{4^+\}$, $RE4 = \{16^-, 15^-\}$.

Definition 27
A module Mn (with head n) is said to have a double crossing among branches if and only if there exists a skeleton S such that WS verifies $N_2 \cap WS \cap B(n, +) \neq \emptyset$ and $N_2 \cap WS \cap B(n, -) \neq \emptyset$. In Fig. 1, an example of a double crossing exists in module $M14$, since, considering skeleton $S4$, $N_2 \cap WS4 \cap B(14, +) = \{15\}$ and $N_2 \cap WS4 \cap B(14, -) = \{16\}$. A similar procedure would confirm that there are no double crossings in $M1$.

Definition 28
We call a crossing point in the branch $B(n, .)$ the minimal element of the set $N_2 \cap WS \cap B(n, .)$.

In the example considered, node 15 is the crossing point in the branch $B(14, +)$ and node 16 in $B(14, -)$.

Definition 29

Let us consider the skeleton S with head n_1: $n_1^{\cdot} A\ n_2^{\cdot} \cdots n_k$. We call the additional entry e the condition node which verifies $e \neq n_i$, $i = 1, \cdots, k$, and $C(e, .) \cap WS \neq \varnothing$. Additional entries are detected by checking for the appearance of any element belonging to the set WS in any g-chain not used in the calculation of the skeleton.

In the example of Fig. 1, this case does not occur; it will appear when we resolve the double crossing.

Lemma 30

If a skeleton S has only one exit node n, there exists a pair of g-chains $C(n, +) = A\ n$, $C(n, -) = \varnothing$ [or $C(n, +) = \varnothing$, $C(n, -) = B\ n$].

This point is clear since, if there exists only one exit node, only one g-chain can exist. Then, this chain will be of the type $C(n, +) = A\ n$ [or $C(n, -) = B\ n$]. In this situation, the successor of n in the $-$ [or $+$] direction will coincide with its ipd; then, $C(n, -) = \varnothing$ [or $C(n, +) = \varnothing$].

Lemma 31

Let us consider a skeleton $n_1^{\cdot} A\ n_2^{\cdot} \cdots n_k$ with head n_1 having $r + 1$ real exits, namely $n_i^{\cdot}, \cdots, n_{i+r}^{\cdot}$. Let us call GC the set of g-chains associated with the nodes $n_1, n_2, \cdots, n_k$. There exists a set GC' associated with the nodes TA, $n_1, n_2, \cdots, n_k$ which contains paths equivalent to the ones in GC if and only if before entering GC' there is a node A0. (These paths are obtained by derivation from the scheme grammar production rules associated with these g-chains).

Let us consider the set GC', which differs from GC in the g-chains:

$$C'(\text{TA}, +) = n_1\text{TA},\ C'(\text{TA}, -) = \varnothing,$$

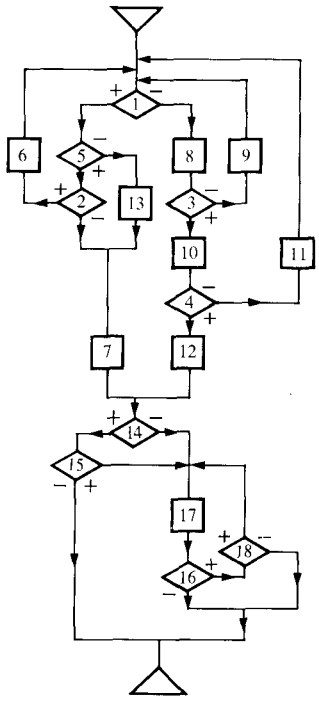
$$C'(n_{i+j}, .) = \text{A1}\ C(n_{i+j}, .) \qquad \text{with } j = 0, 1, \cdots, r,$$

$$C(n_k, .) = C'(n_k, .)\ n_1.$$

We prove the lemma by considering the differences between GC and GC'. With the pair associated with TA, we have introduced a DO-WHILE; once we have entered its iterative branch, the paths are identical to the ones in GC, except for the appearance of nodes A1 in the real exits of the skeleton. In this way, when performing the main test of the DO-WHILE, only if we come from those real exits will it be possible to get out of the loop. The suppression of node n_1 in the last g-chain is due to the fact that this node has been introduced in $C(\text{TA}, +)$, which is the first g-chain following the postdominance ordering.

Algorithm description

The first step of the algorithm is the detection of g-chains, minimal modules, module heads, entry nodes, branches,



$C(\triangledown) = 1\ 14\ \triangle$
$C(1, +) = 5$
$C(1, -) = 8\ 3$
$C(2, +) = 6\ 1$
$C(2, -) = 7$
$C(3, +) = 10\ 4$
$C(3, -) = 9\ 1$
$C(4, +) = 12$
$C(4, -) = 11\ 1$
$C(5, +) = 2$
$C(5, -) = 13\ 7$
$C(14, +) = 15$
$C(14, -) = 17\ 16$
$C(15, +) = 17\ 16$
$C(15, -) = \varnothing$
$C(16, +) = 18$
$C(16, -) = \varnothing$
$C(18, +) = 17\ 16$
$C(18, -) = \varnothing$

Figure 2 Flowgraph and its g-chains after splitting.

skeletons, skeleton heads, real exits, and possible double crossings among branches. The double crossings among branches are immediately eliminated using splitting techniques in the g-chains. In our example, there is a case of double crossing in the module $M14$. The only two possible splitting transformations are repeating nodes 16 and 17 in the branch $B(14, +)$ or repeating node 15 in the branch $B(14, -)$. The second alternative is chosen because fewer nodes are repeated. If we assign number 18 to the node split from node 15, this transformation is performed on the g-chains by changing the g-chain $C(16, +) = 15$ to $C(16, +) = 18$, and by introducing a new pair of g-chains, associated with node 18, identical to the old ones of node 15. Note that if we consider a program scheme instead of a flowgraph, we would have $P(15) = P(18)$.

The transformed flowgraph and its corresponding g-chains are shown in Fig. 2.

The next point is the classification of the minimal modules of the flowgraph according to the number of entry points (one or N) and the number of exits of its corresponding skeleton (one or M).

The case of N entry points and any number of skeletons with one single exit is immediately solved by calculating

$C(\triangledown)$ = A0 TA 14 $\triangle$
$C(TA, +)$ = 1 TA
$C(TA, -) = \varnothing$
= 17 B0 TB
$C(1, +)$ = 5
$C(1, -)$ = 8 3
$C(2, +)$ = 6
$C(2, -)$ = A1 7
$C(3, +)$ = 10 4
$C(3, -)$ = 9
$C(4, +)$ = A1 12
$C(4, -)$ = 11
$C(5, +)$ = 2
$C(5, -)$ = A1 13 7
$C(14, +)$ = 15
$C(14, -)$
$C(TB, +)$ = 16 TB
$C(TB, -) = \varnothing$
$C(15, +)$ = 17
$C(15, -) = \varnothing$
$C(16, +)$ = 18
$C(16, -)$ = B1
$C(18, +)$ = 17
$C(18, -)$ = B1

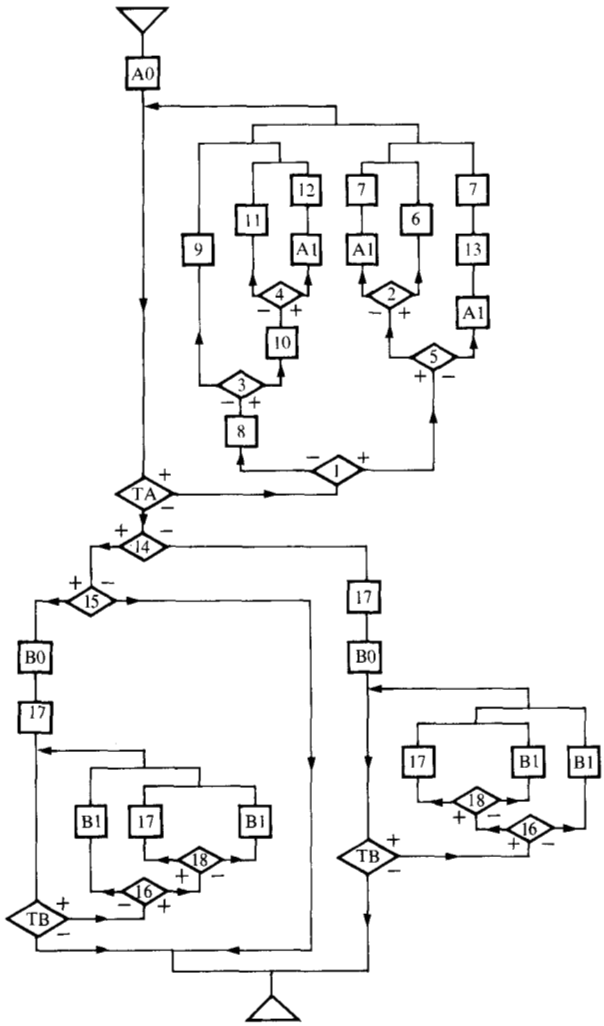


Figure 3 Structured flowgraph.

the g-chains, once the ones associated with those skeletons must be of the type $C(n, +) = A\ n$, $C(n, -) = \varnothing$ [or $C(n, +) = \varnothing$, $C(n, -) = A\ n$] (by Lemma 30), where the elements of A are BLOCK, IF-THEN-ELSE, or DO-WHILE structures. This calculation implies the needed splitting for the module to be structured. The equivalence is guaranteed by the corollary of Lemma 13, since they have the same g-chains.

In the case of modules with one entry and any number of skeletons having M exits, a set of basic transforms is applied, suggested by Lemma 31, namely:

- Replace, in the first g-chain (according to post-dominance) in which it appears, the skeleton head n, with the pair of nodes A0 TA. (In A0 the variable A is set to zero before GC' is entered, and TA introduces the skeleton associated with n).
- Introduce a new pair of g-chains $C(TA, +) = n$ TA, $C(TA, -) = \varnothing$.
- Replace, in the other g-chains, the symbol n with the empty string if it is at the end of the chain and with the string A0 TA otherwise.
- Introduce, in the g-chains associated with exits not being entries to skeletons having the same head, a new first node, A1, in which the control variable A is assigned the value 1.

The only case that remains to be solved using the described steps is the one with more than one entry point and more than one skeleton exit, when any of the entry points possesses a successor in any of the skeletons of the module. In this case, assume that e is the condition node which, in the direction ., is an entry point of a skeleton through node p. The needed replacements in the g-chain $C(e, .)$ are p by A0 p, where A is the control variable associated with the considered skeleton; h by TA h, where h is the first node in the g-chain that does not belong to the skeleton. The module will be completely structured when this process is applied to every "e-type" node.

Up to this point we have not considered the additional entries. Intuitively, this fact implies the need to perform two actions in the g-chain associated with the entry node, namely, to set the entry condition to the skeleton (A0), and to add its associated paths (TA module).

- *Example*

Let us apply this idea to the flowchart in Fig. 2.

We are going to consider the differences from Fig. 1, all of them in module $M14$. The skeleton is

$S4 = 16^{+}\ 18^{+}\ 17\ 16.$

The exits (which coincide with the real ones) are

$E4 = \{16^{-}, 18^{-}\}.$

The entry points for $S4$ have $e = 15$ in the + direction, $p = 17$, $h = \varnothing$.

After applying the set of basic transforms, the new g-chains would be as shown in Fig. 3. Considering the entry point to the skeleton $S4$, the g-chain $C(15, +)$ becomes $C(15, +)$ = B0 17 TB. The flowgraph which corresponds to these g-chains is also shown in Fig. 3. It is left to the reader to verify that it is both structured and equivalent to the flowgraph of Fig. 1.

Summary

We have introduced the concept of the g-chain, a formalization of the grammar scheme given by Urschler, and have shown that the scheme language is regular and its words correspond to the possible paths through the flowgraph.

This regular set (language) can also be obtained by means of an automaton, a graph of the scheme associated with each program [11].

The difference between the two approaches lies in the amount of information contained in the different structures. The regular grammar that one can relate to the automaton, although equivalent, is much less condensed than the context-free grammar we obtain, since the scheme grammar, in the axiom rule, gives the image of the whole structure, while the rules associated with the condition nodes contain information about the range of action of the nodes.

Another application of storing the flowchart structure by means of a grammar is the described structuring algorithm, whose relative simplicity relies on the versatility of the g-chain concept.

References

1. Y. I. Ianov, "The Logical Schemes of Algorithms," *Problems of Cybernetics*, Vol. 1, Pergamon Press, Inc., Elmsford, NY, 1960, pp. 82-140.
2. S. J. Garland and D. C. Luckham, "Program Schemes, Recursive Schemes and Formal Languages," *J. Computer Syst. Sci.* **7**, 119-160 (1973).
3. B. K. Rosen, "Program Equivalence and Context-Free Grammars," *Proceedings of the IEEE 13th Annual Symposium on Switching and Automation Theory*, 1972, pp. 7-18.
4. K. Indermark, "On A Class of Schematic Languages," *Proceedings of Seminar UAM-IBM*, Madrid Scientific Center, North-Holland Pub. Co., 1976, pp. 1-13.
5. G. Urschler, "Automatic Structuring of Programs," *IBM J. Res. Develop.* **19**, 181-194 (1975).
6. M. Schaeffer, *A Mathematical Theory of Global Program Optimization*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1973.
7. H. Mills, "The New Math of Computer Programming," *Commun. ACM* **18**, 43 (1975).
8. K. Kuratowski and A. Mostowski, *Set Theory*, North-Holland Publishing Co., Amsterdam, 1968, pp. 263-264.
9. A. Salomaa, *Formal Languages*, Academic Press, Inc., New York, 1973.
10. J. L. Becerril, J. Bondia, R. Casajuana, and F. Valer, "Estructuracion Automatica de Programas," *Technical Report PLC 03.79*, Centro de Investigacion UAM-IBM, Madrid, 1979.
11. R. Kaplan, "Regular Expressions and the Equivalence of Programs," *J. Computer Syst. Sci.* **3**, 361-386 (1969).

Received January 29, 1980; revised June 18, 1980

The authors are located at the IBM Madrid Scientific Center, Paseo de la Castellana 4, Madrid 1, Spain.