CONTROL STRUCTURES AND MONADIC LANGUAGES *)

Klaus Indermark **)

GMD - Universität Bonn

1. Introduction The theory of program schemes allows to investigate the impact of control mechanisms on the computation power of programs. Those properties which do not depend on the state structure and which hold for arbitrary base functions and data structures can be analysed by means of monadic schemes with one variable. Two such schemes are equivalent iff the sets of computation sequences of their standard forms coincide. This shows how control constructs can be treated syntactically, namely by comparing classes of languages |2|.
The purpose of this paper is to study the particular structure of these languages, called monadic languages (with conditions). A complete version with proofs can be found in |3|.

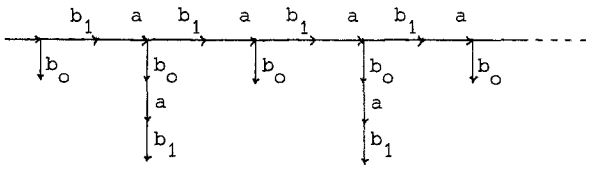
2. Monadic languages

Let A and B be disjoint sets with $|A| \geq 1$ and $|B| \geq 2$. $\Sigma := A \cup B$. The elements of A are called actions, those of B conditions, and those of $C(A,B) := (BA)^*B$ computations over (A,B).
A set L of computations over (A,B) is called a monadic language over (A,B) iff it satisfies (M):

(M) $\forall (w,v,v' \in \Sigma^*, a \in A)\ \exists (v'' \in \Sigma^*)\ wav, wv' \in L \Rightarrow v' = av''$

This property means that an action is uniquely determined by the preceding condition. In other words: a monadic language can be represented by a tree (finite or infinite) which branches out via B.

Example $A = \{a,\ldots\}$, $B = \{b_o, b_1, \ldots\}$



represents the same monadic language L_o as the regular expression
$(b_1\ a\ b_1\ a)^*\ (b_o \vee b_1\ a\ b_o\ a\ b_1)$.

---

*) This research has been supported by the Gesellschaft für Mathematik und Datenverarbeitung mbH., Bonn (GMD)
**) Address: Institut für Informatik, 53 Bonn, Wegelerstr. 6, W-Germany

By $\mathfrak{M}(A,B)$ we denote the class of monadic languages over (A,B). They coincide with Engelfriet's deterministic standard L-schemes |2|. In fact, the sets of computations of Ianov schemes and de Bakker/Scott schemes in standard form are languages of this type. Moreover, if $\underset{\sim}{D} = (D; \phi, \pi)$ is an interpretation of (A,B), i.e. (i) D is a nonvoid set, (ii) $\phi : A \to D^D$, (iii) $\pi : D \to B$, then $(L,\underset{\sim}{D})$ with $L \in \mathfrak{M}(A,B)$ represents a partial function $f_{(L,\underset{\sim}{D})} : D \to D$, and this representation is unique: $f_{(L_1,\underset{\sim}{D})} = f_{(L_2,\underset{\sim}{D})}$ for all interpretations $\underset{\sim}{D}$ of $(A,B) \iff L_1 = L_2$.

3. Operations on $\mathfrak{M}(A,B)$

We shall define certain operations on $\mathfrak{M}(A,B)$ in order to characterize subclasses of languages that correspond to classes of program schemes. Therefore, we simulate syntactically operations on programs such as composition, branching and iteration. However, one cannot use the algebra of regular sets as in |5| because of the particular nature of $\mathfrak{M}(A,B)$.

Let $L, L_1, L_2 \subset C(A,B)$ and $\beta \subset B$. We call

(i) $L_1 \circ L_2 := \{w\, b\, v \mid w\, b \in L_1,\ b\, v \in L_2\}$

the conditional product of L_1 and L_2 and

(ii) $L_1 \,(\beta)\, L_2 := (\beta \circ L_1) \cup ((B \setminus \beta) \circ L_2)$

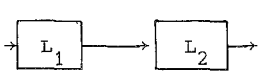
the conditional union of L_1 and L_2 w.r.t. β.

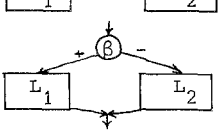
(iii) With $L^{(0)} := B$ and $L^{(n+1)} := L \circ L^{(n)}$ $(n \in \mathbb{N})$ we call $L^{(*)} := \bigcup_{n=0}^{\infty} L^{(n)}$

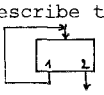
the conditional iteration of L.

Lemma $\mathfrak{M}(A,B)$ is closed under conditional product and conditional unions. This does not hold for conditional iteration.

Conditional product and union correspond to program composition and branching, respectively:

$L_1 \circ L_2$ is the language of 

$L_1 \,(\beta)\, L_2$ is the language of 

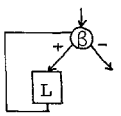
Finally, the conditional iteration can be used to describe the language of the following program iteration 

Namely, if L_i is the set of computations leading to exit i in the original program $(i = 1,2)$ then $L_1^{\circledast} \circ L_2$ describes the computations of this program iteration.

Lemma $\mathfrak{M}(A,B)$ is closed under program iteration, i.e.,

$$L, L_1, L_2 \in \mathfrak{M}(A,B) \quad , \quad L = L_1 \cup L_2 \Rightarrow L_1^{\circledast} \circ L_2 \in \mathfrak{M}(A,B)$$

A special case of program iteration is the so-called while iteration:

$(\beta \circ L)^{\circledast} \circ (B \setminus \beta)$ is the language of 

.

4. Goto and while languages

By means of these operations it is possible to characterize the languages of while and goto schemes. While schemes are Ianov schemes with while loops only whereas goto schemes are arbitrary Ianov schemes.

The class $\mathfrak{L}(A,B)$ of base languages over (A,B) is defined by $\mathfrak{L}(A,B) := \{\beta \mid \beta \subset B\} \cup \{B\, a\, B \mid a \in A\}$. Clearly, $\mathfrak{L}(A,B) \subset \mathfrak{M}(A,B)$.

The class $\mathfrak{G}(A,B)$ of goto languages over (A,B) is defined as the smallest subclass of $\mathfrak{M}(A,B)$ that contains $\mathfrak{L}(A,B)$ and that is closed under conditional product, conditional unions and program iteration.
If we replace in the definition of $\mathfrak{G}(A,B)$ program iteration by while iteration we get the class $\mathfrak{W}(A,B)$ of while languages over (A,B).

Theorem $\mathfrak{G}(A,B) = \mathfrak{M}(A,B) \cap \mathrm{Reg}(A \cup B)$

This theorem shows that the goto languages are just the regular monadic languages, hence, the computation sets of Ianov schemes in standard form.

From the definition of while schemes and while languages it follows that while languages are the computation sets of while schemes. Moreover, we can prove that $\mathfrak{W}(A,B) \subsetneq \mathfrak{G}(A,B)$ using a generalization of Brzozowski's derivations |1|.

Theorem $L_o \in \mathfrak{G}(A,B) \setminus \mathfrak{W}(A,B)$

This proves syntactically that, in general, goto's cannot be eliminated by while statements unless one allows boolean variables, e.g.

5. Conclusion We defined an algebraic structure on $\mathfrak{M}(A,B)$ in order to give inductive descriptions of computation sets of goto and while schemes. However, the characterization of $\mathfrak{Q}(A,B)$ is unsatisfactory insofar as the program iteration is a partial operation: it can be applied only to certain pairs of monadic languages. But, what are the permissible pairs?

In a forthcoming paper |4| , we remove this draw-back by means of so-called vector languages. Moreover, they can be used to characterize computation sets of repeat exit schemes.

References

|1| J.A. Brzozowski: Derivatives of Regular Expressions.
Journal ACM 11 (1964), 481 - 494

|2| J.Engelfriet: Simple Program Schemes and Formal Languages.
Springer Lecture Notes in Computer Science 2o (1974)

|3| K. Indermark: On a Class of Schematic Languages.
GMD-ITAS-Seminarbericht 82 (1974); to appear in:
Proc. International Seminar on Languages and Programming Theory, Madrid (1975),
North Holland P.C.

|4| K. Indermark: The Continuous Algebra of Monadic languages.
Proc. Mathematical Foundations of Computer Science, Mariánské Lázne,
Czechoslovakia (1975); to appear in:
Springer Lecture Notes in Computer Science

|5| D.E. Knuth and R.W. Floyd: Notes on Avoiding goto-Statements.
Information Processing Letters 1 (1971), 23 - 31