CONSTRUCTION OF STANDARD PROGRAM DESIGNS IN THE MUL'TIPROTSESSIST SYSTEM

A. V. Galushka

UDC 519.71

The use of hyperschemas for the construction of standard program designs by the MSPD method is considered. A relationship of hyperschemas with language processors, and in particular mixed computation, is established.

Software development largely relies on various program design support systems [1]. One such system is MUL'TIPROTSESSIST [2], which is based on the method of multilevel structured program design (MSPD) [2]. A specific feature of this method and its support tools is the orientation toward the development of classes of algorithms and programs expressed by schemas in Glushkov's systems of algorithmic algebras (SAA) [3, 4] and toward software product standardization. While program standardization tools are fairly developed in programming systems using assembly languages (macro tools, conditional compiling) and high-level languages (preprocessors, generic procedures), this is not so for design languages. The absence of standardization tools on the level of program design languages is an obstacle to the implementation of the full technological development cycle for standard program modules. The development of such tools in the MSPD framework has led to the creation of the apparatus of structured design grammars (SDG) [2] and the apparatus of hyperschemas [5] (in addition to the concept of schemas, which define classes of algorithms).

In this paper, we consider some applied aspects of the apparatus of hyperschemas, including particular examples.

As we have noted above, an SAA schema is a description of a class of programs with a fixed control structure that are defined by different implementations of elementary operators and conditions (the basis). The program design represented by the schema is thus adapted to the application environment by specifying the corresponding implementations. Basis implementations can be constructed using parametrization tools on the object language level (e.g., macro generation), which allows parametric tuning of the program design. The control structure of the program modules remains unchanged, however, which essentially restricts the flexibility of adaptation.

Adaptation on the control-structure level can be achieved by means of hyperschemas, which introduce the combined philosophy of mixed computation and macro generation into the MSPD method. Hyperschemas are defined on the information set \( P = M \times L \), where \( M \) is the set of all possible states of the parameters controlling the adaptation of the program design and \( L \) is the set of all possible states of the field \( L \) where the adapted program designs are formed in terms of regular SAA schemas. The operations in the algebra of hyperschemas (AHS) are similar to the operations in SAA. By analogy with modified SAA, modified AHS were proposed in [4]. The conditions in AHS, in addition to the traditional logical values \( \varnothing \) ("false"), \( 1 \) ("true"), and \( \mu \) ("undefined"), may also take the value \( \eta \) (not computed). The logical operations \( \wedge \) (conjunction), \( \vee \) (disjunction), and \( \neg \) (negation) are defined by the tables in Fig. 1.

Each operator (hyperschema) $A$ acting on the current state $p = (p^e, p^l)$ of the information set takes $p$ to the state $(A^e(p), A^l(p))$. The state $A^l(p)$ is obtained from the state $p^l$ by writing into the field $L$ (to the right of the accumulated text) a fragment $F(A, p^e)$ of the schema being constructed. The schema is defined on the information set $M$, which is associated with the parameters processed by the program being designed. If $A$ is an elementary operator $A$, $F(A, p^e)$ is also an elementary operator, whose implementation is the result of processing the implementation code of $A$ by the object language preprocessor (in particular, a macro generator) in the context of the state $p^e$. For compound operators, the function $F$ is defined as follows. For the composition of operators $A * B$,

$$F(A * B, p^e) = F(A, p^e) * F(B, A^e(p));$$

$$F($$IF $\alpha$ THEN $A$ ELSE $V$ ENDIF, $p^e$) =
\begin{align*}
F(A, p^e) & \text{ if } \alpha(p) = 1; \\
F(B, p^e) & \text{ if } \alpha(p) = 0; \\
N & \text{ if } \alpha(p) = \mu \text{ (undefined operator)}; \\
& \text{IF } F(\alpha, p^e) \text{ THEN } F(A, p^e) \text{ ELSE } F(B, A^e(p)) \text{ ENDIF if } \alpha(p) = \eta;
\end{align*}$$

$$F($$WHILE NOT $\alpha$ DO $A$ ENDDO, $p^e$) =
\begin{align*}
E & \text{ if } \alpha(p) = 1 \text{ (identity operator)}; \\
N & \text{ if } \alpha(p) = \mu; \\
F(A, p^e) & * F($$WHILE NOT $\alpha$ DO $A$ ENDDO, $A^e(p)$) \text{ if } \alpha(p) = 0; \\
& \text{WHILE NOT } F(\alpha, p^e) \text{ DO } F(A, p^e) \text{ ENDDO if } \alpha(p) = \eta.
\end{align*}$$

We similarly define $F$ for other operators in the signature of AHS and their modifications. For $C_1 = A * B$; $C_2 = $IF $\alpha$ THEN $A$ ELSE $B$ ENDIF; $C_3 = $WHILE NOT $\alpha$ DO $A$ ENDDO$ and $\forall p \in P$, we have

$$C_1^g(p) = B^g(A(p));$$

$$C_2^g(p) = \begin{cases} A^g(p), & \text{if } \alpha(p) = 1; \\
B^g(p), & \text{if } \alpha(p) = 0; \\
B^g(A(p)), & \text{if } \alpha(p) = \eta; \\
N^g(p), & \text{if } \alpha(p) = \mu; \\
E, & \text{if } \alpha(p) = 1 \text{ if } \alpha(p) = \mu; \\
C_3^g(p), & \text{if } \alpha(p) = \eta; \\
A^g(p) * C_3^g(A(p)), & \text{if } \alpha(p) = 0.
\end{cases}$$

The values of the elementary conditions during hyperschema execution are determined by processing the implementation codes of the given conditions in the context of the current state $p \in P$ by the object language preprocessor. If the code is transformed to a logical constant $0$, $1$, or $\mu$, then this constant is the value of the condition. Else $\alpha(p) = \eta$ and the resulting