Abstract: - We consider models of sequential programs (recursive program schemes) and analyze their extension with parallel functions. For this purpose, we introduce a special class of parallel functions (called invariant functions) that don’t depend on interpretation of domain on which they are defined. Expressive power of extended classes of recursive schemes is analyzed in terms of sequential reducibility between the used parallel functions. It is shown that the obtained hierarchy of schemes is infinite but not dense.

Key-Words: - Program models, semantics, parallel functions, expressive power.

1 Introduction
Program scheme is an abstract model that represents a set of concrete programs having similar structure. As opposed to a program where every used function (operation) has well defined meaning (or, in other words, interpretation), in a program scheme some function symbols are left non-interpreted. Every interpretation of such symbols converts a scheme into a program that implements some specific function. Dealing with models such as program schemes, allows for concentration on essential semantic properties of programs, while ignoring irrelevant details. This approach is in the basis of formal methods for program analysis, testing, transformations, etc.

Various classes of program schemes have been extensively studied in the literature. The research concentrated in several major directions: study of various types of equivalence relations and problems of their decidability; analysis and equivalent transformations; comparative study of expressive power for different classes of schemes (e.g. see [1]); etc. This paper belongs to the latter direction. We analyze the hierarchy of languages obtained by addition of various parallel functions to sequential recursive program schemes. Parallel functions were first considered in [1] and continued to attract attention of researches, especially in the context of semantics of algorithmic languages. [2, 5-8].

2 Basic Concepts
Consider an example of recursive scheme:
\[ F(x) = \text{if } p(x) \text{ then } a(x) \text{ else } F(b(x)) \]
here if \((\alpha, x, y)\) is the conditional function, and \(p, a, b\) are uninterpreted function symbols. Note that if \((\alpha, x, y)\) may get well defined value even when only some of its arguments are defined. Thus, when \(p(x)\) is true, \(F(x)\) equals \(a(x)\) regardless of whether \(F(b(x))\) is defined or not (the latter may happen when computation of \(F(b(x))\) leads to endless chain of recursive calls).

Formalization of functions such as if \((\alpha, x, y)\) is based on the use of undefined value \(\omega\) that is viewed as a “result” of non-terminating computation process. Two types of values are considered: Boolean and domain values (the only assumption regarding the latter is that they belong to a constructive set \(D\)). Each of the two types is augmented by \(\omega\), and a partial order \(\subseteq\) is defined: non-\(\omega\) elements are uncomparable, and \(\omega\) is less than any other element; we write \(x \equiv y\) when \(x \subseteq y\) and \(y \subseteq x\). Finally, only monotonic functions (with respect to \(\subseteq\)) are considered [2-4]:
\[ \forall \{x_i \subseteq y_i\} \rightarrow f(x_1,\ldots,x_n) \subseteq f(y_1,\ldots,y_n). \]

Clearly, if \((\alpha, x, y)\) is monotonic; some other important examples:
- function IF\((\alpha, x, y)\); as opposed to if \((\alpha, x, y)\) it produces value \(x\) when \(x = y\), even if \(\alpha \equiv \omega\)
- equality \(x = y\); gets value \(\omega\) when \(x \equiv \omega\) or \(y \equiv \omega\)
- disjunction \(x \text{ or } y\); equals true when \(x = \text{true}\) (even if \(y \equiv \omega\))
- another disjunction \(x \text{ OR } y\); equals true when at least one of \(x, y\) is true (even if another one is \(\omega\))
- voting function \(V^n_m(x_1,\ldots,x_n)\) ( \([n/2] < m < n\) ) : if at least \(m\) arguments get the same non-\(\omega\) value then \(V^n_m\) also gets this value; otherwise \(V^n_m(\overline{x}) \equiv \omega\)
Monotonic functions are classified as \textit{sequential} and \textit{parallel}. Examples of sequential functions are $f(\alpha, x, y), x = y, x \ or \ y$. On the other side, IF$(\alpha, x, y), x \ OR \ y$ and $V_n^m(x_1, ..., x_n)$ are parallel.

A common property of the above functions is that in a sense they don’t depend on the nature of domain $D$; this is formalized in the following definition.

**Definition 1.** A monotonic function $f$ is called \textit{invariant} if for any $1$-$1$ mapping $h : D \rightarrow D$ such that $h(\omega) \equiv \omega$, the following holds:

1. if $f$ gets boolean values then
   
   \[ f(\alpha, h(x_1), ..., h(y_n)) \equiv f(\alpha, x) \]

2. if $f$ gets values in domain $D$ then
   
   \[ f(\alpha, h(x_1), ..., h(y_n)) \equiv h(f(\alpha, x)) \]

Note that none of the constant functions (except true, false and $\omega$) is invariant.

To get a better sense of what are invariant functions, consider the following definition:

**Definition 2.** Two sets $(\alpha, x)$ and $(\alpha, y)$ of values for arguments of function $f$ are called \textit{similar} if:

\[ \forall i[x_i \equiv \omega \leftrightarrow y_i \equiv \omega] \]
\[ \forall i, j[x_i \equiv x_j \leftrightarrow y_i \equiv y_j] \]

**Statement 1.** Let $f$ be an invariant function, and $(\alpha, x)$ and $(\alpha, y)$ are similar.

1. If $f$ gets Boolean values then
   
   \[ f(\alpha, x) \equiv f(\alpha, y) \]

2. If $f$ gets values in domain $D$ then one of the following holds:
   
   \[ f(\alpha, x) \equiv f(\alpha, y) \equiv \omega \]
   
   or
   
   \[ \exists i[f(\alpha, x) \equiv x_i, \ & f(\alpha, y) \equiv y_i] \]

**Example:** functions $\text{if, IF}, = \text{and } V_n^m$ are invariant. Furthermore, non-boolean invariant function selects a value of one of its domain arguments; selection depends on function’s input values.

**Statement 2.** Every invariant function is computable

**Statement 3.** Let $f$ be an invariant sequential function that gets values in $D$. Then it can be expressed as composition of function $\text{if, equality predicate =, and constant } \omega$.

### 3 Main Problems and Results

Invariant (independent of domain $D$) functions are natural candidates to be used for augmentation of program schemes where $D$ is uninterpreted. In this paper, we consider augmentation of regular (sequential) recursive schemes $R$ with invariant parallel functions.

A \textit{recursive scheme} in $R$ (see [3-4]) is a set of equations of the form:

\[ F_i(x_1, ..., x_n) \equiv t_i \]

where term $t_i$ is a composition of function symbols $F_1, ..., F_n$ (functions defined by the scheme), symbols of interpreted functions, interpreted function $f(\alpha, x, y), x = y$, variables $x_1, ..., x_n$ and constants true, false, $\omega$.

For example:

\[ F(x) \equiv (\text{if } (x = d(x)) \text{ then } a(x) \text{ else } F(G(x, b(x))) \]
\[ G(x, y) \equiv (\text{if } q(y) \text{ then } x \text{ else } G(y, c(x)) \]

Note that according to Statement 3, language $R$ allows, in fact, the use of any sequential invariant functions that gets values in $D$.

Let $R(g_1, ..., g_n)$ be an extension of $R$ allowing the use of parallel invariant functions $g_1, ..., g_n$. Furthermore, we write $R(g_1, ..., g_n) \prec R(h_1, ..., h_m)$ when for every scheme in $R(g_1, ..., g_n)$ there exists an equivalent scheme in $R(h_1, ..., h_m)$. For example, in [1] extensions $R(\text{IF})$ and $R(\text{OR})$ were considered, and it was shown that $R \prec R(\text{IF})$ and $R \prec R(\text{OR})$. In this paper we investigate the following:

**Problem:** Characterize expressive power of the various parallel functions through analysis of the hierarchy of classes obtained from $R$ by addition of such functions.

The main results are summarized in Theorems 1-4 below.

**Theorem 1.** For every invariant functions $g$ and $h$, $R(g) \prec R(h)$ holds iff $g$ can be expressed as composition of $h$ and sequential invariant functions.

This theorem reveals a relation between functions added to $R$ that is important for analysis of their impact on expressive power of the extended language. This relation is formalized below:

**Definition 3.** Let $g$ and $h$ be invariant functions. We say that $g$ is reducible to $h$ ($g \leq_{sqi} h$) if $g$ can be expressed as composition of $h$ and sequential invariant functions.

Clearly, relation $\leq_{sqi}$ is reflexive and transitive; it introduces so called \textit{degrees of parallelism} in the class of invariant functions. Theorem 1 stimulates analysis of structure of such degrees. This structure is characterized by the following theorems.

**Theorem 2.** There are infinitely many degrees of parallelism; namely:

\[ V_3^{sqi} > V_4^{sqi} > ... > V_n^{sqi} > ... \]

**Theorem 3.** For every invariant function $f$:

1. If $f$ gets boolean values then $f \leq_{sqi} OR$

2. If $f$ gets values in $D$ then $f \leq_{sqi} V_3^2$
(3) Functions $V^2_3$ and $OR$ are not comparable with respect to $\leq_{sqi}$.

**Theorem 4.** The hierarchy of degrees of parallelism is not dense: there exists $h$, $h <_{sqi} V^2_3$, such that there are no other degrees between $h$ and $V^2_3$; in other words, there is no $g$ such that $h <_{sqi} g <_{sqi} V^2_3$.

Theorem 3 shows that there is no maximal degree of parallelism, because functions $V^2_3$ and $OR$ are not reducible to each other. However, if invariant functions $f$ and $g$ both get Boolean values, or both get values in domain $D$, then there exists a degree that is the exact upper bound of degrees $f$ and $g$: we will denote it as $f+g$.

The rest of the paper concentrates on the proofs of theorems 2 and 4. As for theorems 1 and 3:
- Theorem 1 was proved by us in [7]
- Proof of Theorem 1 is based on a rather simple fact that any set of invariant functions that is closed under composition, is also closed under recursion; we omit the details.

4 Auxiliary Lemmas

The proofs of theorems 2 and 4 are based on three auxiliary lemmas. But first we give some definitions (note that in the sequel, “function” means “invariant function”).

**Definition 4.** Let $f(x_1,...,x_p)$ be an arbitrary (possibly parallel) function. With every non-empty subset of arguments of $f$ we associate a (strict) subfunction of $f$. For example, subfunction $\tilde{f}(x,y)$ of function $f(x,y,z)$ is defined as follows: if $x \equiv \omega$ or $y \equiv \omega$ then $\tilde{f}(x,y) \equiv \omega$; otherwise $\tilde{f}(x,y) \equiv f(x,y,\omega)$. In a similar way, all other subfunctions are defined.

Clearly, any two subfunctions that differ from $\Omega$ (function that is always undefined) have a common argument.

**Definition 5.** Subfunction $\tilde{f}(x_{i_1},...,x_{i_s})$ is called minimal if it differs from $\Omega$ but for every $r$, $1 \leq r \leq s$, subfunction $\tilde{f}(x_{i_1},...,x_{i_{r-1}},x_{i_{r+1}},...,x_{i_s})$ is identical to $\Omega$.

Note that every function $f$ can be fully characterized in terms of its minimal subfunctions, in the following sense. For given values of arguments $x_1,...,x_p$, function $f$ gets value $t$ if there exists its minimal subfunction that also equals $t$ for these input values. For this reason, in the sequel we will consider only minimal subfunctions, and will refer them just as subfunctions.

**Definition 6.** Let $f(x_1,...,x_p) : D^p \rightarrow D$ be a function, and $t$ a value that differs from $\omega$.

Subfunction $\tilde{f}$ that satisfies the condition:

$$\tilde{f}(x_{i_1},...,x_{i_s}) = t \iff x_{i_1},...,x_{i_s} = t$$

(*)

will be called subfunction of type (*).

**Lemma 1.**

$V^2_{sqi} \geq V^3_{sqi} \geq \ldots \geq V^{n-1}_{sqi} \geq \ldots$

**Lemma 2.** Let $f : D^p \rightarrow D$ be a function. (1) If $f$ has $m \geq 3$ subfunctions of type (*) without a common argument, then:

$$\forall n \geq m[V^1_{sqi} \leq_{sqi} IF + f]$$

(2) If $m = 3$, then:

$$\forall n \geq 3[V^1_{sqi} \leq_{sqi} f]$$

**Lemma 3.** Let $f : D^p \rightarrow D$ be a function. If any $m$ subfunctions of type (*) have a common argument. Then:

$$\forall n \leq m[V^{n-1}_{sqi} is not reducible to \ IF + f]$$

Lemmas 2, 3 provide a criterion for reducibility of $V^{n-1}_{sqi}$ to $IF + f$. According to this criterion, the allowed “structure” of $f$ is rather restrictive. Namely, $V^1_{sqi} \leq_{sqi} IF + f$ iff $f$ is, in a sense, an extension of $V^1_{sqi}$.

5 Proof of Theorems

**Proof of Theorem 2.** It is already known (see lemma 1) that:

$$V^2_{sqi} \geq V^3_{sqi} \geq \ldots \geq V^{n-1}_{sqi} \geq \ldots$$

Let us show now that for any $n \geq 3$, $V^{n-1}_{sqi}$ is not reducible to $V^{n+1}_{sqi}$. Note that $V^{n+1}_{sqi}(y_1,...,y_{n+1})$ has $n+1$ subfunctions, each of them depending on $n$ arguments; all these subfunctions are of type (*). We will show that every $n$ of these subfunctions have a common argument. Then, according to Lemma 3, $V^{n-1}_{sqi}$ is not reducible to $IF + V^{n+1}_{sqi}$, and moreover, $V^{n-1}_{sqi}$ is not reducible to $V^{n+1}_{sqi}$.

Suppose that the assumption is not correct. Hence each variable $y_i (1 \leq i \leq n+1)$ appears in argument...
sets of not more than \(n - 1\) subfunctions; a total of such appearances is estimated as:

\[
\leq (n - 1)(n + 1) = n^2 - 1
\]

On the other side, each of the \(n\) subfunctions depends on \(n\) arguments, and hence there should be \(n^2\) of such appearances. This contradiction completes the proof of the theorem.

**Proof of Theorem 4.** Let us define an invariant function \(f(x_1, \ldots, x_n)\) as follows:

\[
f(x_1, \ldots, x_n) = \begin{cases} 
  x_1, & \text{if } x_1 = x_2 = x_3 \text{ OR } x_1 = x_2 = x_4 \text{ OR } x_1 = x_3 = x_4 \\
  x_2, & \text{if } x_2 = x_3 = x_4 \\
  x_3, & \text{if } x_1 = x_2 = x_5 \text{ OR } x_1 = x_3 \neq x_6 \\
  \alpha, & \text{otherwise}
\end{cases}
\]

Let us define \(h \equiv IF + f\); we will show that \(h\) fulfills the theorem’s requirements.

First, note that \(f(x)\) has four subfunctions of type (\(*\)), namely: \(f_1(x_1, x_2, x_3), f_2(x_1, x_2, x_4), f_3(x_1, x_2, x_3),\) and \(f_4(x_1, x_2, x_4).\) Any three of them have a common argument and hence according to Lemma 3, \(V_3^2\) is not reducible to \(IF + f\). On the other side, according to Theorem 3, \(IF + f \leq s_{IF} V_3^2\)

and hence \(IF + f < s_{IF} V_3^2\).

If now \(g\) is an invariant function that gets boolean values, then \(IF + f < s_{IF} g\) is impossible because otherwise we would get \(IF < s_{IF} g\) and then, according to Theorem 3, \(IF < s_{IF} OR\). But as was proved in [7], the latter is wrong.

As for functions that get values in domain \(D\), we will prove even a stronger fact than the one claimed in the theorem, namely: if \(g < s_{IF} V_3^2\) then \(g \leq s_{IF} + f\). Note that it is enough to check this for functions that have only non-boolean arguments. Indeed, let \(g(a_1, \ldots, a_k, x_1, \ldots, x_n)\) be a function with Boolean arguments \(a_i\) and domain arguments \(x_j\), and let \(g_1, \ldots, g_{sk}\) be all functions obtained from \(g\) by assigning all possible combinations of values to the Boolean arguments. It is shown in [8] that \(g \leq s_{IF} + g_1 + \ldots + g_{sk}\). If now \(g < s_{IF} V_3^2\) then necessarily \(g_i < s_{IF} V_3^2\) holds for every \(i\) (otherwise we would have \(V_3^2 < s_{IF} g_i < s_{IF} g\) and hence \(V_3^2 < s_{IF} g\)). But \(g_i\) has only domain arguments; hence if we will know how to prove that in such case \(g_i \leq s_{IF} + f\), then we will immediately get:

\[
g \leq s_{IF} + g_1 + \ldots + g_{sk} \leq s_{IF} + f
\]

Hence, the sequel only functions with all domain arguments are considered.

Let \(g_1, \ldots, g_n\) be all subfunctions of function \(g\). It is easy to see that for each pair \(g_i\) and \(g_j\) there exists their union (or, in other words, the exact upper bound with respect to relation \(\subseteq\)) that we denote by \(g_i \cup g_j\), and furthermore — that \(g = g_1 \cup \ldots \cup g_n\). Suppose now that for every three subfunctions \(g_i, g_j, g_k\) we can prove (based on \(g < s_{IF} V_3^2\)) that:

\[
g_i \cup g_j \cup g_k \leq s_{IF} + f
\]

Then based on the fact that for \(k \geq 3\):

\[
g_i \cup \ldots \cup g_{k+1} \equiv V_4^3 (g_i \cup \ldots \cup g_{k+1})
\]

we will easily get:

\[
g_1 \cup \ldots \cup g_n \leq s_{IF} + f + V_4^3
\]

and hence \(g \leq s_{IF} + f + V_4^3\). Since the four subfunctions of \(f\) don’t have a common argument, we conclude (according to Lemma 2) that \(V_4^3 \leq s_{IF} + f\) and hence \(g \leq s_{IF} + f\) as required.

So, we only have to check that our assumption is correct. We consider four cases, taking into account that for every \(g\) and for every pair of its subfunctions \(g_i\) and \(g_j\) the following reducibility holds (see [8]):

\[
g_i \cup g_j \leq s_{IF}.
\]

Without loss of generality, we consider subfunctions \(g_1\), \(g_2\) and \(g_3\).

**Case 1.** Let \(g_1\), \(g_2\) and \(g_3\) be subfunctions of type (\(*\)). Then they must have a common argument, say \(y_i\). Indeed, otherwise (according to Lemma 2) we would have \(V_3^2 \leq s_{IF} g\) which is incorrect. It is clear that if the value of \(g_k\) \((k = 1, 2, 3)\) differs from \(\omega\) then it is equal to the value of \(y_i\). Hence clearly:

\[
g_1 \cup g_2 \cup g_3 = f(g_1 \cup g_2 \cup g_3) = f(g_1, g_2, g_3, y_i, \omega, \omega, \omega)
\]

**Case 2.** Suppose that \(g_1\) is not of type (\(*\)), while \(g_2\) and \(g_3\) are of type (\(*\)). Let \(y_i\) be an argument common for \(g_2\) and \(g_3\). Note that if \(g_1\) differs from
then there exists its argument \( y_1 \) whose value is also different from \( \omega \) and from that of \( g_1 \). Then:

\[
g_1 \cup g_2 \cup g_3 = \quad f(g_1 \cup g_2, g_1 \cup g_3, y_1, y_2, \omega, \omega)
\]

**Case 3.** If \( g_3 \) is of type (*) but \( g_1 \) and \( g_2 \) are not, then by selecting \( y_1, y_2 \) and \( y_3 \) as above we get:

\[
g_1 \cup g_2 \cup g_3 = \quad f(g_1 \cup g_2, g_1 \cup g_3, g_2 \cup g_3, y_1, y_2, y_3, \omega)
\]

**Case 4.** Finally, if all three subfunctions \( g_1, g_2 \) and \( g_3 \) are not of type (*) then similar to the above we get:

\[
g_1 \cup g_2 \cup g_3 = \quad f(g_1 \cup \bar{g_2}, \bar{g_1} \cup \bar{g_3}, \bar{g_2} \cup \bar{g_3}, y_1, y_2, y_3)
\]

This completes the proof of the theorem.

### 6 Conclusion

The paper provides a comparative study of expressive power for invariant parallel functions. Use of such functions in sequential recursive schemes leads to a rich hierarchy of extended schemes. Classes of schemes in this hierarchy are compared in terms of sequential reducibility. It is shown that the hierarchy is infinite but not dense (there is no way to “get close” to the function that possesses the maximal expressive power). A further research is required to get a more complete characterization of the expressive power for parallel functions.

### References:


