*Annales Univ. Sci. Budapest., Sect. Comp. 17 (1998) 173-191*

# THE FORMAL SPECIFICATION OF A PROBLEM SOLVED BY A PARALLEL PROGRAM – A RELATIONAL MODEL

**Z. Horváth** (Budapest, Hungary)

**Abstract.** We introduce the basic concepts of a relational model of parallel programming. We define the concepts of a problem, an abstract program and a solution. Our approach is functional, the primary concepts of the model are problem and refinement of a problem. Problems are given an own semantical meaning. The refinement of problems may be interpreted as a relation over the domain of problems. The abstract program is regarded as a relation generated by a set of nondeterministic conditional assignments similar to the concept of abstract program in UNITY. We introduce the behaviour relation of a parallel program which is easy to compare to the relation which is the interpretation of a problem. The abstraction level defined by the semantical mapping from abstract programs to their behaviour relations is identical with the abstraction level defined by the semantics of abstract problems.

## 1. Introduction

We introduce the basic concepts of a relational model of parallelism [15, 16, 18]. Our model is an extension of a powerful and well-developed relational model of programming which formalizes the notion of state space, problem, sequential program, solution, weakest precondition, specification, programming theorem, type, program transformation etc. [9, 10, 14, 29].

We take the specification of the *problem* as the starting point for program design. We develop a model of programming which supports the top-down

Supported by the Hungarian National Science Research Grant (OTKA), Grant Nr. 2045.

refinement of specifications. The proof of the correctness of the solution is developed parallel to the refinement of the problem. We formalize the main concepts of UNITY [3] in an alternative way. We use a relatively simple mathematical machinery [29]. The result is an expressive model, which is related to branching time temporal logics [6].

The preliminary form of the programming model elaborated in this paper was used in [16, 11]. The paper [12 (*in this volume*) is based on the introduced model, too.

### 1.1. Preliminary notions

In the following we use the terminology used also in [9, 10, 14, 29, 16].

Denote $A^*$ the set of all finite and $A^\infty$ the set of all infinite sequences of elements of an finite or numerable set $A$. $A^{**} ::= A^\infty \cup A^*$.

The *concatenation* of sequences $\alpha \in A^*$, $\beta \in A^{**}$ is $con(\alpha, \beta) ::= < \alpha_1, \ldots, \alpha_{|\alpha|}, \beta_1 \ldots >$.

The *reduced sequence* $\mathrm{red}(\alpha)$ corresponding to $\alpha \in A^{**}$, is obtained by replacing each finite stationary subsequence within $\alpha$ by one of its elements.

Let $\tau : A^* \longmapsto A$ be a function which associates with each finite sequence its last element, $\tau(\alpha) ::= \alpha_{|\alpha|}$.

An arbitrary subset of a direct product of sets is called a *relation*. Further we deal with relations which are subsets of the direct product of two sets (binary relations).

Let $R \subset A \times B$, where $A$ and $B$ are arbitrary sets. The *domain of the relation* $R$ is defined by $\mathcal{D}_R ::= \{a \in A \mid \exists b \in B : (a, b) \in R\}$.

The *image of* $a \in A$ *with respect to the relation* $R$ is $R(a) ::= \{b \in B \mid (a, b) \in R\}$.

A relation $R$ is called a *partial function*, if for all $a \in A$ the set $R(a)$ has at most one element. If $\forall a \in A : |R(a)| = 1$ then $R$ is a *function*.

Let $R \subseteq A \times B$, $H \subseteq B$. The set $R^{-1}(H)$ is called the *pre-image of* $H$ *with respect to relation* $R$, if $R^{-1}(H) = \{a \in A \mid a \in \mathcal{D}_R \wedge R(a) \subseteq H\}$.

We can define the *composition* of two relations as follows: $P \subseteq A \times B$, $Q \subseteq B \times C$, $Q \circ P ::= \{(a, c) \in A \times C \mid \exists b \in B : (a, b) \in P \wedge (b, c) \in Q\}$.

Let $R \subseteq A \times \mathrm{L}$ be a relation, where $A$ is an arbritrary set and L is the set of the logical values. Then $R$ is called a *logical relation*.

The *truth set of the relation* $R$ is $\lceil R \rceil ::= R^{-1}(\{ \mathit{true} \})$. The logical functions $\uparrow, \downarrow : A \longmapsto \mathrm{L}$ are defined by their truth sets. $\lceil \uparrow \rceil = A, \lceil \downarrow \rceil = \emptyset$.

The set $(\mathcal{P}(A))$ denotes the powerset of the set $A$.

We use the words predicate and condition as synonyms for logical function. $[f]$ abbreviates the theorem $(\lceil f \rceil = A)$ [5] The operations $\cup, \cap, A\backslash$ correspond

to the function $\wedge, \vee, \neg$. $\Rightarrow$ corresponds to $\subseteq$, $P \rightarrow Q$ is an abbreviation of $\neg P \vee Q$.

Let $A_1, \ldots, A_n$ be arbitrary finite or numerable sets. The set $A ::= A_1 \times \times A_2 \times \cdots \times A_n$ is called the *state space*.

The projections (functions) $v_i : A \rightarrow A_i$ (for all $i \in \{1, \ldots, n\}$) of the state space $A_1 \times A_2 \times \cdots \times A_n$ are called *variables*.

## 2. The specification of a problem

The notion of the state space makes it possible to define the semantical meaning of a problem independently of any program.

We generalize the specification method which specifies the problem by the help of pre- and postconditions which are formulated as a composition of logical functions or relations. The generalized concept of a problem is applicable for cases in which termination is not required, but the behaviour of the specified (closed) system is restricted by safety and progress properties. The solution of a problem may be a sequential program, a parallel one, or even *a program built up from both sequential and parallel components*. The primary goal of the generalization of the problem is to extend the relational model for the case of nontermination [2].

Our purpose is to set up a model in which the problem can be formulated independently of its solution, it can be compared with other problems solved by different programs. Any program running over the state space of the problem can be checked whether it is a solution of the problem or not[3].

### 2.1. Specification properties

The problem is defined as a set of specification relations. Every specification relation is defined over the powerset of the state space. Let $P, Q, R, U : A \longmapsto \mathcal{L}$ be logical functions. We define $\triangleright, \mapsto, \hookrightarrow \in \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A))$, and $\mathrm{FP}, \mathrm{INIT}, \mathrm{inv}, \mathrm{TERM} \subseteq \mathcal{P}(A)$.

We introduce the following infix notations:

[2] We show how to generalize the concept of a problem for open systems in section 2.3.

[3] This view is basically different from the point of view of temporal logic where the interpretation of the specification is connected to a model of which time structure is defined by a program.

$P \triangleright Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \triangleright$, $P \mapsto Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \mapsto$, $P \hookrightarrow Q ::= (\lceil P \rceil, \lceil Q \rceil) \in \hookrightarrow$, $\mathrm{FP} \Rightarrow R ::= \lceil R \rceil \in \mathrm{FP}$, $Q \hookrightarrow \mathrm{FP} ::= \lceil Q \rceil \in \mathrm{TERM}$, $Q \in \mathrm{INIT} ::= \lceil Q \rceil \in \mathrm{INIT}$, $\mathrm{inv}P ::= \lceil P \rceil \in \mathrm{inv}$.

The $P \triangleright Q$, $P \mapsto Q$, etc. formulas are called specification properties or short properties. The $\triangleright, \mapsto, \hookrightarrow, \mathrm{inv}, \mathrm{TERM}$ relations define transition properties, the $\mathrm{FP}, \mathrm{INIT}$ relations define boundary properties. The transition relations $\triangleright$ and inv express so called safety properties, while the relations $\mapsto, \hookrightarrow, \mathrm{TERM}$ express progress properties.

The meaning of specification properties are given by the definition of a solution (def. 4.1). For the sake of better understanding we give an informal explanation in advance.

A program satisfies the safety property $P \triangleright Q$, if there is no direct transition from $P \wedge \neg Q$ to $\neg P \wedge \neg Q$ only through $Q$ if any. $P$ is said to be *stable* if $P \triangleright \downarrow$. A program satisfies the progress properties $P \mapsto Q$ or $P \hookrightarrow Q$ if the program starting from $P$ inevitably reaches a state, in which $Q$ holds. $P \mapsto Q$ defines further restriction for the direction of the progress. The fixed point property $\mathrm{FP} \Rightarrow R$ defines a necessary condition for the case when the program is in one of its fixed points. It is sufficient if the program satisfies these requirements over the reachable states only [27].

A $Q \in \mathrm{INIT}$ property defines a sufficient condition for the initial states of the program. $Q \hookrightarrow \mathrm{FP}$ expresses that the program starting from $Q$ inevitably reaches one of its fixed points. Problems are easy to decompose to subproblems by the appropriate choice of the INIT and FP relations [11, 12].

If $P$ holds "initially" and $S$ preserves the truth of $P$, then $P$ is an **invariant**, denoted by $\mathrm{inv}P$.

## 2.2. A specification of closed systems

**Definition 2.1.** *Let $A$ be a state space and let $B$ be a finite or countable set. Two relations expressing boundary properties and five relations expressing transition properties are associated to every point of set $B$. The relation $F \subseteq$ $\subseteq B \times (\overset{\times}{}_{i \in [1..3]} \mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A)) \, \overset{\times}{}_{i \in [1..4]} \mathcal{P}(\mathcal{P}(A)))$ is called a problem defined over the state space $A$. $B$ is called the parameter space of the problem.*

Let $b \in B$ denote an arbitrary element of the domain of the problem. Let $h$ denote an element of $F(b)$. The components of $h$ are denoted by $\triangleright_h, \mapsto_h, \hookrightarrow_h$ and by $\mathrm{INIT}_h, \mathrm{FP}_h, \mathrm{inv}_h, \mathrm{TERM}_h$ respectively. If $|F(b)| = 1$ then we use $b$ instead of $h$ in the indices for the sake of simplicity.

A parameter space [10] is similar to a state space, it is a direct product of type value sets. Usually the state space and the parameter space have common components. Although there is no transition caused by the execution of the

program in the parameter space, we call the projections of the parameter space to its components variables as like as in the case of the projections of the state space [4]. To distinguish between the two kinds of variables we extend the variables of the parameter space by a', for example: $v'$.

The introduction of parameter space makes it possible to formalize problems allowing alternative behaviours as their solutions [5].

On the other hand the appropriate choice of the parameter space reduces the number of verification steps. The relations $\triangleright_h$, $\hookrightarrow_h$, $\mathrm{FP}_h$, etc. are usually finite sets (often with one single element). If the parameter space $B$ is infinite, then we define infinite number of relations. These relations are different from one another in the value of their parameter only. This means it is sufficient to perform the verification calculus in few parameterized cases.

**Remark 2.1.** *The 2.1 definition of a problem is generalization of the concept of a problem given in [7, 10, 29]. If* $\forall b \in B : |F(b) = 1|$ *and* $\{Q_b\} = \mathrm{TERM}_b$, $\{Q_b\} = \mathrm{INIT}_b$ *and* $\{R_b\} = \mathrm{FP}_b$, *then the problem specified according to the definition 2.1 can be rewritten in the form required by the theorem of specification [10] and vice versa.*

**Remark 2.2.** *The problem as relation is not depending on the syntactical form of the specification properties. Even different set of properties may express the same abstract problem (Def. 2.4).*

### 2.3. A specification of open systems

In this section we show a way how to extend the model for open systems.

**Definition 2.2.** *Let $A$ be a state space and let $B$ be a finite or countable set. Three relations expressing boundary properties and ten relations expressing transition properties are associated to every point of set $B$. The relation* $F \subseteq$ $\subseteq B \times (\overset{\times}{}_{i\in[1..6]}\mathcal{P}(\mathcal{P}(A)\times\mathcal{P}(A))_{i\in[1..7]}\overset{\times}{}\mathcal{P}(\mathcal{P}(A)))$ *is called a task defined over the state space $A$. $B$ is called the parameter space of the task.*

The components of an arbitrary $h \in F(b)$ are denoted by $\triangleright_h, \mapsto_h$, $\hookrightarrow_h, \triangleright_h^E, \mapsto_h^E, \hookrightarrow_h^E$, $\mathrm{TERM}_h$,$\mathrm{FP}_h$, $\mathrm{inv}_h$, $\mathrm{INIT}_h$,$\mathrm{TERM}_h^E$,$\mathrm{FP}_h^E$, $\mathrm{inv}_h^E$ respectively.

[4] The variables of the state space are called local variables, while the variables corresponding to the variables of the parameter space are called global or rigid variables in first order temporal logic languages.

[5] If the problem is deterministic, then the problem can be formulated without the introduction of a parameter space, i.e. we can substitute the parameter space by a new one having a single element only and form the specification relations as the union of the specification relations associated to the elements of the original parameter space.

The components having an $^{E}$ superscript are assumed properties of the full (closed) system [3] including the specified open process and its environment, i.e. the superscripted specification properties define implicit requirements for the environment [4]. A process (a modul of an abstract program) solves its task if it solves the problem assuming that the behaviour of the full system satisfies the superscripted specification properties. The elaboration of the model based on the definition 2.2 of task is out of the scope of this paper [17].

### 2.4. A refinement of problems

We interpret a refinement of problems as a binary relation defined over the domain of problems, i.e. over the set $B \times (\overset{\times}{_{i\in[1..3]}} \times\mathcal{P}(\mathcal{P}(A)\times \times\mathcal{P}(A))\overset{\times}{_{i\in[1..4]}}\mathcal{P}(\mathcal{P}(A)))$ (2.1 def.).

Our view of stepwise refinement is different from the models of [1, 22, 24]. We will not speak about stepwise refinement of programs, but about of stepwise refinement of problems. We do not use programs as specification tools usually.

Since the refinement relation is connected to a given state space and to a given parameter space [6], the stepwise refinement of problems includes state space and/or parameter space (coordinate system) transformation steps [8, 5]. For example, we introduce new variables often, i.e. we extend the state space by new components. The extension of a problem to the new state space is defined as in [9, 29], i.e. the extended problem does not imply any restrictions for the new variables. However, a refinement of the extended problem may include further or stricter specification properties which properties restrict the behaviour of a possible solution in respect of the new components, too.

The refinement relation with respect to a fixed state and parameter space is induced by the solution relation mapping from the set of abstract parallel programs to the set of problems. An ordered pair of an abstract program and of a problem is an element of the solution relation, if and only if the program solves the problem.

Let us denote by $\mathcal{S}_A$ the set of abstract parallel programs defined over $A$.

**Definition 2.3.** *Let $F_1, F_2$ be problems defined over the state space $A$. If $\forall S \in \mathcal{S}_A$: $S$ solves $F_2 \Rightarrow S$ solves $F_1$, then the problem $F_2$ is a refinement of the problem $F_1$.*

The refinement relation is a preorder, i.e. it is a reflexive, transitive relation and its kernel is an equivalence relation [13].

[6] Deterministic problems defined over a common state space, but over different parameter spaces, are easy to compare from the point of view of the refinement relation (see footnote 5).

**Remark 2.3.** *The concept of a refinement relation is based on the concepts of an abstract program and on the concept of a solution as on parameters only.*[7].

To decide whether a problem is a refinement of an other one is hard in general [6]. On the other hand a formal verification of the refinement steps is possible during stepwise refinement of problems if the refinement steps are chosen carefully.

**Definition 2.4.** *The problem $F_1$ is equivalent with the problem $F_2$, if the problem $F_1$ is a refinement of the problem $F_2$ and the problem $F_2$ is a refinement of the problem $F_1$.*

The equivalence classes generated by the equivalence relation of problems are called abstract problems.

## 3. The definition of a parallel program

The specification of a problem and its solution, the abstract program is independent of architecture, scheduling and programming languages. The abstract program is regarded as a relation generated by a set of nondeterministic (simultaneous) conditional assignments similar to the concept of abstract program in UNITY [3]. The conditions of the assignments encode the necessary synchronization restrictions explicitly. Some assignments are selected nondeterministically and performed in each step of the execution of the abstract program. Every statement is executed infinitely often, i.e. an unconditionally fair scheduling is postulated. If more than one processor selects statements for execution, then the executions of different processors are fairly interleaved. A fixed point is said to be reached in a state, if none of the statements changes that state [3].

[7] For concrete definitions of an abstract parallel program and of a solution see definitions 3.9 and 4.1.

### 3.1 Conditional assignments

We can imagine a statement (a sequential program) as a relation, which associates a sequence of points of the state space to some points of the state space, i.e. a statement is a subset of the direct product $A \times A^{**}$ [29].

A relation $s \subset A \times A^{**}$ is called a *statement* over the state space $A$, if $D_s = A$, $\forall a \in A : \forall \alpha \in R(a) : \alpha = red(\alpha)$, $\forall a \in A : \forall \alpha \in s(a) : \alpha_1 = a$.

The *effect relation* of a statement $s$ is a relation $p(s) \subseteq A \times A$ defined as follows:
$\mathcal{D}_{p(S)} ::= \{a \in A \mid s(a) \subseteq A^*\}$,
$\forall a \in \mathcal{D}_{p(S)} : \; p(s)(a) ::= \{b \in A \mid \exists \alpha \in s(a) : \; \tau(\alpha) = b\}$.

A statement over the state space $A$ is called *empty* and termed $SKIP$, if $\forall a \in A : \; SKIP(a) = \{(a)\}$.

Let $A = A_1 \times ... \times A_n$, $F = (F_1, ..., F_n)$, where $F_i \subseteq A \times A_i$. The statement $s \subseteq A \times A^{**}$ is a *general assignment* defined by $F$, if $s ::= \{(a, red(a,b)) \mid a, b \in \in A \wedge a \in \bigcap_{i \in [1,n]} \mathcal{D}_{F_i} \wedge b \in F(a)\} \cup \{(a, (aaa....)) \mid a \in A \wedge a \notin \bigcap_{i \in [1,n]} \mathcal{D}_{F_i}\}$.

A special kind of statement, the conditional assignment is the basic building element of abstract parallel programs. First we will define the extension of a relation with respect to a condition

**Definition 3.1** *Le $B$ be a subspace of $A$ and let $R \subseteq A \times B$. $\pi : A \longmapsto \mathcal{L}$. $R|_\pi ::= (R \cap (\lceil \pi \rceil \times B)) \cup \{(a, pr_B(a)) | a \in \lceil \pi \rceil \setminus \mathcal{D}_R\}$, where*

*$pr_B(a)$ denotes the projection of $a \in A$ into the subspace $B$;*

*$R|_\pi$ is called the extension of $R$ with respect to $\pi$.*

As an important special case we may extend relations defined over a state space for the truth set of the condition $\uparrow$, i.e. we extend the domain of the relation for the whole state space. $A = A_1 \times \ldots \times A_n$, $F \subseteq A \times A$, $F = (F_1, \ldots, F_n)$, where $F_i \subseteq A \times A_i$. Let $\lceil \pi_i \rceil ::= \mathcal{D}_{F_i}$. The relation $F_i|_\uparrow$ is the extension of $F_i$ for the truth set of condition $\uparrow$, i.e. $F_i|_\uparrow(a) ::= F_i(a)$, if $a \in \lceil \pi_i \rceil$ and $F_i|_\uparrow(a) ::= a_i$, otherwise. $F||_\uparrow ::= (F_1|_\uparrow, \ldots, F_n|_\uparrow)$.

**Definition 3.2.** *Let be given an assignment $s_j$, for which $((\mathcal{D}_{s_j} = A) \wedge \wedge (\forall a \in A : p(s_j)(a) = F||_\uparrow(a)))$. This kind of (simultaneous, nondeterministic) assignment is called conditional if $\forall a \in A : |p(s_j)(a)| < \omega$. We denote the conditional assignment $s_j$ the following way: $(\|_{i \in [1,n]} (v_i :\in F_{j_i}(v_1, .., v_n), \; if \; \pi_{j_i}))$.*

By virtue of its definition the effect relation of a conditional assignment is total, i.e. its domain is equal with the whole state space. This means that a conditional assignment terminates always. If the program is in the state $a$

and $\neg\pi_{j_i}(a)$ holds, then the value of the variable $v_i$ is not changed by the assignment $s_j$.

We define the semantical meaning of a variable to be on the left hand side or on the right hand side of an assignment.

**Definition 3.3.** *The relation $R$ is independent of the variable $v_i : A \longmapsto \longmapsto A_i$, if and only if $\forall a, b \in \mathcal{D}_R : (\forall k \in ([1, i-1] \cup [i+1, n]) : a_k = b_k) \Rightarrow R(a) = R(b)$. If the effect relation of a general assignment is not independent of the variable $v$, then $v$ occurs on the right hand side of the assignment.*

**Definition 3.4** *The variable is on the left hand side of the conditional assignment $s$, if $v \neq v \circ p(s)$, i.e. $\exists a \in \mathcal{D}_{p(s)} : v(a) \neq v(p(s)(a))$.*

We omit the component $(v_i := F_i(v_1, .., v_n)$, if $\pi_i(v_1, ..., v_n))$ from the description of the conditional assignment, if $v_i$ does not occur on the left hand side of it. Similarly, we omit the variable $v_j$ from the notation of the relation $F_{j_i}$, if $F_{j_i}$ is independent of $v_j$.

### 3.2. A semantics of abstract parallel program

Let $S$ be an ordered pair of a conditional assignment and of a nonempty, finite set of conditional assignments, such that $S = (s_0, \{s_j \mid j \in J\})$, where $J = \{1..m\}$, $m \geq 1$.

The program is defined as a binary relation which associates equivalence classes of correctly labelled state transition trees to the points of the state space. The labelled state transition trees are generated by the ordered pair of the effect relation of initial assignment $s_0$ and of the disjoint union of the effect relations of $\{s_1, ..., s_m\}$ elements of the abstract program.

First we define the concept of a labelled transition tree generated by an ordered pair of total relations.

A labelled transition tree is a system (r,N,V,L,S) where $r$ is the root, $N$ is the set of the vertices, $V \subseteq N \times N$ is the set of the edges, $L : N \longmapsto A$ and $S : V \longmapsto J$ are the vertex and edge labelling functions, $\forall x \in N : (x, r) \notin V$, there is a unique path leading from $r$ to any $x \in N, x \neq r$.

Two labelled transition trees are called isomorph if there exists a bijection between their vertices which bijection preserves the root vertex and the labelling of the edges and vertices.

**Definition 3.5.** *The labelled transition trees $G_1 = (r_1, N_1, V_1, L_1, S_1)$ and $G_2 = (r_2, N_2, V_2, L_2, S_2)$ are isomorph, if there exists a bijection $f : N_1 \longmapsto N_2$ for which: $\forall x \in N_1 : f(V_1(x)) = V_2(f(x)) \wedge L_1(x) = L_2(f(x))$, $\forall (x, y) \in V_1 : (f(x), f(y)) \in V_2$ and $S_1(x, y) = S_2(f(x), f(y))$.*

By virtue of its definition the isomorphism defined by 3.5 is an equivalence relation over the labelled transitions trees.

**Definition 3.6.** *Let $R_0, R \subseteq A \times A$ be total relations, i.e, $\mathcal{D}_R = \mathcal{D}_{R_0} = A$. The labelled transition tree $GR(a) = (r, N_a, V_a, L_a, S_a)$ is said to be generated by the ordered pair $(R_0, R)$ of relations at the point $a$, if*

- $L_a(r) = a$,
- $\forall x \in N_a \setminus \{r\} : L_a(V_a(x)) = R(L_a(x))$,
- $L_a(V_a(r)) = R_0(L_a(r)) = R_0(a)$.

Let us choose the state space $A$ as the set of labels for vertices and $J \subset \mathcal{N}_0$ as the set of labels for edges.

**Definition 3.7.** *Let us denote by $A^{***}$ the equivalence classes of labelled transition trees generated over the state space $A$ and index set $J$ by ordered pairs of relations.*

Let us represent the equivalence class by one of its elements.

Let $S = (s_0, \{s_1, ...s_m\})$ denote an ordered pair of a conditional assignment $s_0$ and a finite nonempty set of conditional assignments. Let $J = \{1, .., m\}$. Let us denote by $UP(S)$ the disjoint union of the effect relations $p(s_j)$.

**Definition 3.8.** *The labelling of the tree generated by the ordered pair of relations $(p(s_0), UP(s))$ is correct, if all edges starting from $r$ are labelled by 0 and for all edges having the label $j$ and pointing from a vertex labelled by $b$ to a vertex labelled by $c$ hold that $(b, c) \in p(s_j)$.*

**Definition 3.9.** *The relation $UPG(S) \subseteq A \times A^{***}$ is called an abstract parallel program, if it associates equivalence classes of labelled transition trees to the element $b \in A$, which trees are generated at $b$ by the ordered pairs of relations $(p(s_0), UP(s))$ and have a correct labelling.*

The abstract parallel program $UPG(S)$ generated by $S = (s_0, \{s_1, ...s_m\})$ is abbreviated by $S$ in the following. The conditional assignment $s_0$ is called the initialization in $S$ and $s_j : j \in [1..m]$ is said to be an element of the program $S$.

**Remark 3.1.** *Programs which are equivalent from the point of view of correctness but implementing different algorithms correspond to different abstract parallel programs, i.e. their semantical representation is different.*

**Definition 3.10.** *Any path of a representative of the equivalence class $UPG(a)$ is called an execution path of the abstract parallel program starting in the state $a$.*

The introduced semantics is an interleaving semantics of parallel programs. An implementation which allows true parallel asynchronous execution of conditional assignments is not modelled. The concurrent execution of the

conditional assignments should satisfy the requirement of serializibility [20]. Every execution path of the abstract parallel program represents a possible sequential execution sequence of the assignments. A true parallel semantics would destroy even the restricted compositionality of the programming model [2. 4].

### 3.3. A behaviour relation of abstract parallel programs

The program properties with respect of an abstract parallel program are characterized as relations over the powerset of the state space.

#### 3.3.1. A generalization of the weakest precondition

The program properties are defined in terms of the weakest precondition of the element statements [5, 29] of the abstract program. We use the dual concept of strongest postcondition, too [21].

The logical function $wp(s, R)$ is called the **weakest precondition** of the postcondition $R$ in respect to the statement $s$. We define $\lceil wp(s,R)\rceil ::= \{a \in \in \mathcal{D}_{p(s)} | p(s)(a) \subseteq \lceil R\rceil\}$. The logical function $sp(s, Q)$ is called the strongest postcondition of $Q$ in respect to $s$. $\lceil sp(s,Q)\rceil ::= p(s)(\lceil Q\rceil)$.

Let us denote by $S$ an abstract parallel program over the state space $A$, where $S = (s_0, \{s_1, \ldots s_m\})$. We generalize the concept of weakest precondition for abstract parallel programs [15].

**Definition 3.11.** $wp(S, R) ::= \forall s \in S : wp(s, R)$.

#### 3.3.2. Invariants and reachable states

Let us denote by $\mathrm{inv}_S(Q)$ the set of logical functions whose truth are preserved by the elements of $S$ if the program is started from a state satisfying $Q$.[8]

**Definition 3.12.** $\mathrm{inv}_S \subseteq \mathcal{P}(A) \times \mathcal{P}(\mathcal{P}(A))$. $\mathrm{inv}_S(Q) \subseteq \mathcal{P}(A)$. $\mathrm{inv}_S(Q) ::= \{\lceil P\rceil \mid sp(s_0, Q) \Rightarrow P$ *and* $P \Rightarrow wp(S, P)\}$. *Let us denote by* $\mathrm{INV}_S(Q)$ *the conjunction of the elements of the set* $\mathrm{inv}_S(Q)$[9].

The truth set of $\mathrm{INV}_S(Q)$ is the set of reachable states starting the program from the truth set of $\lceil Q\rceil$ [27].

[8] $\mathrm{inv}_S(Q)$ is the set of strong invariants [27].

[9] $\mathrm{INV}_S(Q)$ is the strongest invariant [21, 27].

### 3.3.3. Safety properties

Let us denote by $\triangleright_S$ the set of ordered pairs $(P, Q)$ of logical functions for which holds that $P$ is stable while $\neg Q$ during the execution of $S$.

**Definition 3.13.** $\triangleright_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.
$\triangleright_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P \wedge \neg Q \Rightarrow wp(S, (P \vee Q)))\}$ [10]

### 3.3.4. Progress properties

Let us denote by $\mapsto_S$ the set of ordered pairs $(P, Q)$ of logical functions for which holds that $P$ is stable while $\neg Q$ during the execution of $S$ and there is a conditional assignment $s_j$ which ensures the transition from $P$ to $Q$.

**Definition 3.14.** $\mapsto_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$.
$\mapsto_S ::= \{(\lceil P \rceil, \lceil Q \rceil) \mid (P, Q) \in \triangleright_S \wedge \exists j \in J : (P \wedge \neg Q \Rightarrow wp(s_j, Q))\}$ [11]

**Definition 3.15** *Let be* $\hookrightarrow_S \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$ *the transitive disjunctive closure of* $\mapsto_S$ *[25], i.e. the smallest binary relation over A satisfying the conditions* [12]

- $\mapsto_S \subseteq \hookrightarrow_S$.
- *if* $(P, Q) \in \hookrightarrow_S$ *and* $(Q, R) \in \hookrightarrow_S$, *then* $(P, R) \in \hookrightarrow_S$.
- *Let W denote an countable set. If* $\forall m : (m \in W :: (P(m), Q) \in \hookrightarrow_S$ , *then* $((\exists m : m \in W :: p(m)), Q) \in \hookrightarrow_S$.

**Remark 3.2.** *The relational extensions of UNITY [25] redefine the concepts of unless, ensures and leads-to in a form which correspond to the program properties and do not correspond to the specification properties.*

**Remark 3.3.** *There are alternative definitions of* $\hookrightarrow_S$ *elaborated in [15, 19] which are based on the fixed point properties of monotone functionals [26].*

**Theorem 3.1.** *If and only if* $(\{b\}, \lceil P \rceil) \in \hookrightarrow_S$ *then on all execution pathes leading from b and satisfying the axiom of the unconditionally fair scheduling there is a node at a finite unbounded distance from b of which label is an element of the truth set of P, i.e., the program inevitable reaches the truth set of P started from b.*

The proof can be constructed based on results of [25]. The size limit given for this paper does not allow us to include the proof here, the proof is given in [17].

---

[10] The definition of $\triangleright_S$ corresponds to the concept of UNITY's unless [3].

[11] The definition of $\mapsto_S$ corresponds to the concept of UNITY's ensures, if the axiom of the unconditionally fair scheduling is postulated [3].

[12] *The definition of* $\hookrightarrow_S$ *corresponds to the concept of UNITY's leads-to [3].*

We can prove the following theorems corresponding to the properties used in the definition of leads-to in UNITY [3]. The proof of progress properties is supported by the introduction of so called variant functions [7, 3].

**Theorem 3.2.** *Let be* $P, Q : A \longmapsto \mathcal{L}$, $t : A \longmapsto \mathcal{Z}$ *such that* $(P \wedge \neg Q) \Rightarrow t > 0$. *If* $\forall m \in \mathcal{N} :: (P \wedge \neg Q \wedge t = m) \hookrightarrow_S ((P \wedge t < m) \vee Q)$, *then* $P \hookrightarrow_S Q$.

The proof is based on structural induction and given in [17].

### 3.3.5. Fixed point properties

A fixed point is said to be reached in a state of the state space $A$, if none of the statements changes the state.

Let us denote by $\pi_{j_{id}}$ the logical function, which characterizes the set of states over which the relation $F_{j_i}$ is deterministic, i.e. $\pi_{j_{id}}(a) \Leftrightarrow (|F_{j_i}(a)| = 1)$.

**Definition 3.16.**

$$fixpoint_S ::= (\bigwedge_{j \in J, i \in [1..n]} (\neg \pi_{j_i} \vee (\pi_{j_{id}} \wedge v_i = F_{j_i}(v_1, \ldots, v_n))))$$

See paper [12] for examples calculating $fixpoint_S$. The definition of $fixpoint_S$ generalizes the fixpoint calculation method introduced in [3] for the case of nondeterministic assignments.

**Lemma 3.1.** $fixpoint_S$ *is the set of fixed points of the program* $S$ *over the state space* $A$.

The lemma follows directly from the definition 3.16.

**Definition 3.17.** *Let us denote by* $\mathrm{FP}_S$ *the set* $\{\lceil R \rceil | fixpont_S \Rightarrow R\}$.

### 3.3.6. Termination properties

**Definition 3.18** *Let us denote by* $\mathrm{TERM}_S$ *the set* $\{\lceil Q \rceil | (Q, fixpoint_S) \in \in \hookrightarrow_S\}$.

### 3.3.7. An other semantics of abstract parallel programs

We may define the semantical meaning of a parallel program by the system of the set of its strongest invariants, its safety, progress, fixed point and termination properties. Unfortunately this kind of semantics is not generally compositional for the program construction rule union [3, 17], compositionality is violated in the case of the program property $\hookrightarrow_S$ and $\mathrm{TERM}_S$.

**Definition 3.19** *Let* $S$ *be a program over the state space* $A$. *The system of relations* $(\triangleright_S, \mapsto_S, \hookrightarrow_S, \mathrm{FP}_S, \mathrm{inv}_S, \mathrm{TERM}_S)$ *is called the behaviour relation of the parallel program* $S$.

## 4. The formal definition of a solution

We introduce a relation called solution. The domain of solution is the set of problems, the range of it is the set of programs. If $(F, S)$ is an element of the solution relation, then $F$ is said to be solved by program $S$. Since the structure of the behaviour relation of abstract parallel program is similar to the problem, the concept of solution is expressed in term of the behaviour relation (3.19 def.).

This way we get a simple UNITY like verification calculus ultimately based on the concept of the weakest precondition. The calculus proves the correctness of an abstract parallel program with respect to the problem in $O(m)$ steps, where $m$ is the number of conditional assignments of $S$. Finally, we state some useful lemmas to simplify the calculus in practice and to justify the correctness of the definition of solution.

**Definition 4.1.** *The abstract parallel program* $S \subseteq A \times A^{***}$ *is a solution of the problem* $F \subseteq B \times (\overset{\times}{}_{i\in[1..3]}\mathcal{P}(\mathcal{P}(A) \times \mathcal{P}(A))_{i\in[1..4]}\overset{\times}{}\mathcal{P}(\mathcal{P}(A)))$, *if* $\forall b \in B$ : $\exists h \in F(b)$, *such that the program* $S$ *satisfies all the specification properties given in the* $\mathrm{inv}_h$, $\triangleright_h$, $\mapsto_h$, $\hookrightarrow_h$, $\mathrm{FP}_h$, $\mathrm{TERM}_h$ *components of* $h$ *assuming that the program starts from a state satisfying all the elements of* $\mathrm{INIT}_h$.

Let us define what to mean about the program satisfying a specification property with respect to $init_h$ initial properties.

**Definition 4.2.** *The program* $S$ *satisfies the specification property* $(\mathrm{inv}_h P)$, *if and only if there exists an invariant property* $K$ *such that the program statisfies* $(\mathrm{inv}_h P)$ *with respect to* $K$, *i.e.* $K \in \mathrm{inv}_S(\overset{\wedge}{}_{Q\in\mathrm{INIT}_h} Q)$ *and* $P \wedge K \in \mathrm{inv}_S(\overset{\wedge}{}_{Q\in\mathrm{INIT}_h} Q)$.

**Remark 4.1.** *As we mentioned in Section 3.3.2 the points of the state space, not in the truth set of an invariant property, are unreachable states, i.e. the truth set of an invariant property may be regarded as a characterization of a subset of unreachable states. It is sufficient for us, if the program satisfies all properties over the truth set of an invariant property [27]. This means that a program is said to satisfy a specification property, even if the program fails to satisfy it over a subset of the unreachable states [3, 23, 27, 17].*

**Definition 4.3.** *The program* $S$ *satisfies the specification property* $P \triangleright_h Q$, *if and only if there exists an invariant property* $K$ *such that the program statisfies* $P \triangleright_h Q$ *with respect to* $K$, *i.e.* $K \in \mathrm{inv}_S(\overset{\wedge}{}_{Q\in\mathrm{INIT}_h} Q)$ *and* $(P \wedge K, Q \wedge K) \in \triangleright_S$.

**Definition 4.4.** *The program $S$ satisfies the specification property $P \mapsto_h Q$, if and only if there exists an invariant $K$ such that the program statisfies $P \mapsto_h Q$ with respect to $K$, i.e. $K \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$ and $(P \wedge K, Q \wedge K) \in \mapsto_S$.*

**Remark 4.2.** *The validity of the Definition 4.4 is depending on the truth of the axiom of unconditionally fair scheduling [3].*

**Definition 4.5.** *The program $S$ satisfies the specification property $P \hookrightarrow_h Q$, if and only if there exists an invariant $K$ such that $K \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$ and $(P \wedge K, Q) \in \hookrightarrow_S$.*

**Definition 4.6.** *The program $S$ satisfies the specification property $P \hookrightarrow \mathrm{FP}_h$, if and only if there exists an invariant property $K$ such that $K \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$ and $(sp(s_0, P) \wedge K) \in \mathrm{TERM}_S$.*

**Definition 4.7.** *The program $S$ satisfies the specification property $(\mathrm{FP}_h \Rightarrow R)$, if there exists an invariant $K$ such that $K \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$ and $fixpont_S \wedge K \Rightarrow R$.*

### 4.1. Lemmas to simplify the verification calculus

**Lemma 4.1.** *$S$ satisfies $(\mathrm{inv}_h P)$, if and only if there exists an invariant property $K \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$, such that*
*$sp(s_0, (\bigwedge_{Q \in \mathrm{INIT}_h} Q)) \Rightarrow P \wedge K$ and $P \wedge K \Rightarrow wp(S, P \wedge K)$.*

The lemma follows directly from the definitions 3.12 and 4.2.

**Lemma 4.2.** *$S$ satisfies $P \triangleright_h Q$ if and only if there exists an invariant property $K \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$, such that $(P \wedge \neg Q \wedge K \Rightarrow wp(S, (P \vee Q) \wedge K))$.*

The lemma follows directly from the definitions 3.13 and 4.3.

**Lemma 4.3.** *$S$ satisfies $(Q \mapsto_h P)$ if and only if there exists an invariant property $K \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$, such that $S$ satisfies $Q \triangleright_h P$ with respect to $K$ and $\exists j \in J : (P \wedge \neg Q \wedge K \Rightarrow wp(s_j, Q \wedge K))$.*

The lemma follows directly from the definitions 3.14 and 4.4.

**Lemma 4.4.** *$S$ satisfies $P \hookrightarrow_h Q$ , if it can be deduced in finite steps by the application of the following reasoning rules [3]:*

- *If $S$ satisfies $(P \mapsto_h Q)$, then $S$ satisfies $(P \hookrightarrow_h Q)$.*
- *Transitivity: if $S$ satisfies $(P \hookrightarrow_h Q)$ and $S$ satisfies $(Q \hookrightarrow_h R)$, then $S$ satisfies $(P \hookrightarrow_h R)$.*
- *Disjunctivity: Let $W$ denote a countable set. If $\forall m : S$ satisfies $(m \in W :: P(m) \hookrightarrow_h Q)$, then $S$ satisfies $((\exists m : m \in W :: p(m)) \hookrightarrow_h Q)$.*

The proof of the lemma is given in [17].

**Lemma 4.5.** *The program $S$ satisfies $P \hookrightarrow \mathrm{FP}_h$ ($P \in \mathrm{TERM}_h$), if $S$ satisfies $sp(s_0, P) \hookrightarrow_h fixpont_S$.*

The lemma follows directly from the definitions 3.18, 4.6 and 3.16.

**Lemma 4.6.** *Let $t : A \longmapsto \mathcal{Z}$ be a variant function and $P, Q$ logical functions such that $P \wedge \neg Q \Rightarrow (t > 0)$. If $\forall m \in \mathcal{N}$ the program $S$ satisfies the property $(P \wedge \neg Q \wedge (t = m)) \hookrightarrow_h ((P \wedge (t < m)) \vee Q)$ then $S$ satisfies the property $P \hookrightarrow_h Q$.*

The lemma is a consequence of the Lemma 4.5 and the Theorem 3.2. The proof is given in [17].

**Consequence 4.1.** *Let be $K \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$, $t : A \longmapsto \mathcal{Z}$ such that $(K \wedge \neg\, fixpoint_S) \Rightarrow t > 0$. If $\forall t' \in \mathcal{N} : (K \wedge \neg\, fixpont_S \wedge t = t') \hookrightarrow_S (K \wedge t < t') \vee fixpont_S$, then $S$ satisfies $Q \in \mathrm{TERM}_h$ for any $Q$ logical function.*

**Lemma 4.7.** *$S$ satisfies $(\mathrm{FP}_h \Rightarrow R)$ with respect to $K$, if $K \in \mathrm{inv}_S(\bigwedge_{Q \in \mathrm{INIT}_h} Q)$ and $fixpont_S \wedge K \Rightarrow R$.*

The lemma follows directly from the definitions 4.7 and 3.16.

## 5. Discussion

The introduced model is a coherent extension of the relational model of programming used in education at the Eötvös University. Problems are interpreted as relations providing an expressive tool to formalize alternative requirements for the behaviour of programs. Programs are relations capturing the operational aspects of the solution. Behavioural relation corresponds to the effect relation of sequential programs capturing the main functional properties of a program. Special program constructions for example as union, superposition [3] and sequence of programs are easy to adopt [17]. The generalization of the type concept [29], the extension of the model for open systems and answering the question of compositionality of progress properties are subject for further research.

## References


[1] **Back R.J.R. and Sere K.,** Stepwise Refinement of Parallel Algorithms, *Science of Computer Programming*, **13** (1989/90), 133-180.

[2] **Chandy K.M.,** Reasoning about continuous systems, *Science of Computer Programming*, **14** (1990), 117-132.

[3] **Chandy K.M. and Misra J.,** *Parallel program design: a foundation*, Addison-Wesley, 1989.

[4] **Collette P.,** Composition of assumption-commitment specifications in a UNITY style, *Science of Computer Programming*, **23** (1994), 107-125.

[5] **Dijkstra E.W. and Scholten C.S.,** *Predicate Calculus and Program Semantics*, Springer, 1989.

[6] **Emerson E.A. and Srinivasan J.,** Branching Time Temporal Logic, *Linear Time, Branching Time and Partial Order in Logics an Models for Concurrency, LNCS* **354**, Springer, 1989, 123-172.

[7] **Fóthi Á.,** *Bevezetés a programozáshoz (Introduction into Programming)*, egyetemi jegyzet, ELTE TTK, Budapest, 1983.

[8] **Fóthi Á.,** Lectures in "Introduction into Programming" (1986-1987), verbal communications.

[9] **Fóthi Á.,** A mathematical approach to programming, *Annales Univ. Sci. Bud. Sect. Comp.*, **9** (1988), 105-114.

[10] **Fóthi Á. and Horváth Z.:** The Weakest Precondition and the Theorem of the Specification, *Proceedings of the Second Symposium on Programming Languages and Software Tools, eds. K.Kosmikies and K.Räihä, Pirkkala, Finland, August 21-23, 1991*, Report A-1991-5, University of Tampere, Department of Computer Science, 1991, 39-47.

[11] **Fóthi Á. and Horváth Z.,** A Parallel Elementwise Processing, *Proceedings of the 2nd Austrian-Hungarian Workshop on Transputer Applications, September 29-October 1, 1994, Budapest, Hungary,* eds. Sz.Ferenczi and P.Kacsuk, KFKI-1995-2/M,N Report, 1995, 273-282.

[12] **Fóthi Á., Horváth Z. and Kozsik T.,** Parallel elementwise processing - a Novel version, *Proceedings of the Fourth Symposium on Programming Languages and Software Tools, Visegrád, Hungary, June 8-14, 1995*, ed. L.Varga, ELTE, 1995, 180-194.

[13] **Hennessy M.,** *Algebraic Theory of Processes*, The MIT Press, 1988.

[14] **Horváth Z.,** Fundamental relation operations in the mathematical models of programming, *Annales Uni. Sci. Bud. Sect. Comp.*, **10** (1990), 277-298.

[15] **Horváth Z.,** The Weakest Precondition and the the Specification of Parallel Programs, *Proceedings of the Third Symposium on Programming Languages and Software Tools, Käariku, Estonia, August 21-23, 1993,* ed. M.Tombak, 24-33.

[16] **Horváth Z.,** The asynchronous computation of the values of an associative function, *Acta Cybernetica,* **12** (1) (1995), 83-94.

[17] **Horváth Z.,** *A relational programming model of parallel programs,* PhD thesis, ELTE, Budapest, 1996.

[18] **Horváth Z. and Kozma L.,** Parallel Programming Methodology, *Workshop on Parallel Processing. Technology and Applications, Budapest, Hungary, 10-11 February, 1994,* eds. J.Bogdany and G.Vesztergombi, KFKI-94-09/M,N Report, 1994, 57-65.

[19] **Jutla C.S., Knapp E. and Rao J.R.,** A Predicate Transformer Approach to Semantics of Parallel Programs, *Proc. 8th Ann. ACM SIGACT/SIGOPS Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989,* 249-263.

[20] **Lamport L. and Lynch N.,** Distributed Computing Models and Methods, *Handbook of Computer Science, vol. B,* ed. van Leeuwen, Elsevier, Amsterdam, 1990, 1157-1199.

[21] **Lamport L.,** win and sin: Predicate Transformers for Concurrency, *ACM Transactions on Programming Languages and Systems,* **12** (3) (1990), 396-428.

[22] **Lamport L.,** *The Temporal Logic of Actions,* Technical Report SRC Research Number TR79, Digital Equipment Corporation, Systems Research Center, Palo Alto, CA, ftp: gatekeeper.dec.com: pub/DEC/SRC/research-reports, 1991.

[23] **Misra J. et al.,** Notes on UNITY 1988-1993., University of Texas, Austin, ftp: ftp.cs.utexas.edu.

[24] **Morris J.M.,** A Theoretical Basis for Stepwise Refinement and the Programming Calculus, *Science of Computer Programming,,* **9** (1987), 287-306.

[25] **Pachl J.,** A simple proof of a completeness result for leads-to in the UNITY logic, *Information Processing Letters,* **41** (1992), 35-38.

[26] **Park D.,** On the semantics of fair parallelism, *Lect.Notes on Comp.Sci.* **86**, Springer, 1980, 504-526.

[27] **Prasetya I.S.W.B.,** Error in the UNITY Substitution Rule for Subscribed Operators, *Formal Aspects of Computing,* **6** (1994), 466-470.

[28] **Varga L.,** *Programok analízise és szintézise,* Akadémiai Kiadó, Budapest, 1981.

[29] **Workgroup on Relational Models of Programming,** Some concepts of a Relational Model of Programming, *Proceedings of the Fourth Symposium on Programming Languages and Software Tools, Visegrád, Hungary, June 8-14, 1995,* ed. L.Varga, 434-446.

**Z. Horváth**
Department of General Computer Science
Eötvös Loránd University
VIII. Múzeum krt. 6-8.
H-1088 Budapest, Hungary
hz@lngsc2.elte.hu