

FUNCTIONAL DEPENDENCIES OF VARIABLES IN WAIT-FREE PROGRAMS (Extended Abstract)

Evangelos Kranakis

Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
(eva@cwi.nl)

ABSTRACT

Suppose that we are given a wait-free protocol for the asynchronous, concurrent processes P_1, P_2, \dots, P_r , Q_1, Q_2, \dots, Q_s , with $r \geq 2$, $s \geq 0$. For any run (or interleaving) ρ of the protocol and any initialization $init$ of all the protocol variables let $X[\rho, init]$ be the value of the variable X at the end of the run ρ . The variables X_1, X_2, \dots, X_r "belonging" to the processors P_1, P_2, \dots, P_r , respectively, are called functionally dependent for the initialization $init$, if for any runs ρ, σ of the protocol,

$$(\forall i, j)(X_i[\rho, init] = X_i[\sigma, init] \Leftrightarrow X_j[\rho, init] = X_j[\sigma, init]).$$

For any run ρ and any initialization $init$ of the protocol define the evaluation mapping $eval_{X_1, X_2, \dots, X_r}(\rho, init) = (X_1[\rho, init], X_2[\rho, init], \dots, X_r[\rho, init])$. We show that for any protocol as above, the variables X_1, X_2, \dots, X_r are functionally dependent for the initialization $init$ if and only if the quantity $eval_{X_1, X_2, \dots, X_r}(\rho, init)$ is independent of ρ .

1. Introduction

There has been a lot of interest in the current literature on Distributed Computing for a more thorough examination of the computational possibilities offered by wait-free protocols. In particular, this has led to a re-examination of the necessity of using control primitives in the design of Concurrent Reader, Concurrent Writer protocols. The results obtained so far have been particularly interesting. Several researchers have been able to implement: (i) atomic, 1-reader, 1-writer registers from safe, 1-reader, 1-writer registers ([L], [K], [T]), and (ii) atomic, multireader, multi-writer registers from atomic, 1-reader, 1-writer registers, by using only wait-free protocols ([A2], [B], [K], [L1], [L2], [N], [P], [V]). Wait-free protocols are of particular interest not only because they are free from the usual control primitives (like, Mutual Exclusion, Test and Set, etc.), but also because they make possible a rather quantitative appraisal of the complexity of various algorithms, e.g. determining the wait-free protocol with the best running time ([P], [L2]).

There are certain instances of programming methodology which have "inherent" waiting requirements (e.g. whenever it is necessary to allocate a critical resource among many users). In such instances, the mechanism of waiting has been extensively used ever since its introduction by Dijkstra [D]. As it seems natural these considerations have led [H] to implement a hierarchy of objects such that objects at a certain level of the hierarchy are "stronger" (with respect to waiting mechanisms) than objects lying below this object in this same hierarchy. The purpose of the present paper is twofold: (1) to reappraise the computational aspects and limitations of wait-free programs, and (2) to "draw the line" between what is possible (e.g. solving the Concurrent Readers/Writers problem) and what is impossible (e.g. Mutual Exclusion problem) by using only

wait-free programs. Before giving an outline of the main results it will be necessary to introduce some useful concepts.

1.1. Preliminaries

In order to motivate our results and facilitate the discussion we will first consider wait-free protocols consisting of processors each executing a sequence of assignment statements. In section 5 we will indicate all the modifications necessary to cover the most general wait-free protocols (such protocols in addition to assignment statements will include: **if ... then ... else ... fi**; and **for $i = 1 \dots n$ do ... od** statements). Let Σ be a language consisting of the assignment symbol $:=$, the function symbols F, G, \dots (with subscripts and/or superscripts) each associated with a specified arity ≥ 0 , (if the arity of F is 0 then F is also called a constant) and the variables $v_0, v_1, \dots, v_n, \dots$

An assignment statement of Σ is a formula of the form $x := F(x_1, \dots, x_n)$, where x, x_1, \dots, x_n are variables and $F \in \Sigma$ is an n -ary function symbol. Suppose that P and Q are asynchronous, concurrent processors each executing a finite sequence p_1, \dots, p_m and q_1, \dots, q_n , respectively, of assignment statements. These assignment statements form a program or protocol $P \parallel Q$. Call the variables occurring in all these assignment statements of $P \parallel Q$, protocol variables. If $x := F(x_1, \dots, x_r)$, is an assignment statement of P (respectively, Q) then the variable x is said to belong to P (respectively, Q). The protocol variables are supposed to satisfy certain atomicity conditions (see section 3, for formal definitions). The program $P \parallel Q$ can have numerous possible executions. We illustrate this in the example below.

Suppose that processor P (respectively Q) intends to execute the assignment statement p : " $x := F(x')$ " (respectively q : " $y := G(y', y'')$ "). In general, a possible execution might be given by the following sequence of statements: Q reads y' ; P reads x' ; Q reads y' ; P writes $x = F(x')$; Q writes $y = G(y', y'')$. However, for the purposes of the investigations of the present paper such "lower level" interleavings will never be considered. In other words, although interleavings among the $p_1, \dots, p_m, q_1, \dots, q_n$ are possible, the actions p_i and q_j will be considered atomic, i.e. either all subactions of p_i precede all subactions of q_j or else all subactions of q_j precede all subactions of p_i . For this reason we will also call such assignment statements atomic.

Let x_1, \dots, x_k be a list of all the protocol variables. An interpretation or model of the protocol $P \parallel Q$ is a structure $\mathcal{M} = (M, F^{\mathcal{M}}, G^{\mathcal{M}}, \dots)$ together with a k -tuple $init \in M^k$, where

- if $F \in \Sigma$ is an n -ary function symbol then $F^{\mathcal{M}} : M^n \rightarrow M$, and
- $init = (c_1, \dots, c_k)$ are the initial interpretations of the protocol variables x_1, \dots, x_k , respectively, where $c_1, \dots, c_k \in M$; the k -tuple $init$ is also called initialization of the protocol variables.

For any sequence $\sigma = (r_1, \dots, r_m)$ of atomic assignment statements r_1, \dots, r_m of the protocol $P \parallel Q$, and any protocol variable X define the value $X[\sigma, init]$ of the variable X in the model $\mathcal{M} = (M, F^{\mathcal{M}}, G^{\mathcal{M}}, \dots)$ with respect to the sequence σ by induction on the length of σ . (†) Suppose that X is the variable x_j in the list x_1, \dots, x_k of all the protocol variables. If $\sigma = \emptyset$, i.e. σ is the empty sequence, then $X[\sigma, init] = c_j$. Let σ be the sequence (r_1, \dots, r_{s+1}) and let ρ be the sequence (r_1, \dots, r_s) . Suppose that r_{s+1} is the assignment statement $y := F(y_1, \dots, y_n)$. Then define $X[\sigma, init] = F^{\mathcal{M}}(y_1[\rho, init], \dots, y_n[\rho, init])$ if $X = y$, and $X[\sigma, init] = X[\rho, init]$ if $X \neq y$.

We are interested in program executions (interleavings or runs) $\rho = (\alpha, <_{\rho})$, of the

(†) The value of the variable X in a model \mathcal{M} for the protocol $P \parallel Q$ at the end of the execution of the run ρ depends on $P \parallel Q, \mathcal{M}, init, \rho$, where $init$ is a given initialization in \mathcal{M} . Therefore a more correct notation is $X^{\mathcal{M}}[P \parallel Q, \rho, init]$. Instead we use $X[\rho, init]$ by abuse of notation, because the model \mathcal{M} and protocol $P \parallel Q$ will always be easily understood. Moreover, we will normally be referring to an initialization $init$ of the protocol variables without explicitly mentioning the model \mathcal{M} .

assignment statements $p_1, \dots, p_m, q_1, \dots, q_n$, with $\mathcal{O} = \{r_1, \dots, r_{m+n}\} = \{p_1, \dots, p_m\} \cup \{q_1, \dots, q_n\}$. The order of actions in a run ρ is determined by the relation $<_\rho$, i.e. $p <_\rho p'$ if and only if p immediately precedes p' in the run ρ , where $p, p' \in \mathcal{O}$. We assume that the transitive closure \rightarrow_ρ of $<_\rho$ is a partial ordering on the set \mathcal{O} such that the natural ordering of the execution of the program $P \parallel Q$ is preserved, i.e. for any run ρ of the program $P \parallel Q$ the following order among the atomic assignment statements in \mathcal{O} must be preserved:

$$p_1 \rightarrow_\rho p_2 \rightarrow_\rho \dots \rightarrow_\rho p_m, q_1 \rightarrow_\rho q_2 \rightarrow_\rho \dots \rightarrow_\rho q_n.$$

In general, actions p_i, q_j may be concurrent in a run ρ ; in our framework this can be expressed by simply stating that p_i, q_j are \rightarrow_ρ -incomparable. However, as explained before, here we are only interested in a specific type of runs (or "higher level" interleavings) for which \rightarrow_ρ is a linear order. Let $RUN(P \parallel Q)$, or simply RUN , be the set of all these runs, where $P \parallel Q$ is a certain program as above.

Let \mathcal{M} be a given model of the protocol and suppose that X, Y are variables belonging to P, Q , respectively. The evaluation mapping of the protocol $P \parallel Q$ (with respect to the model \mathcal{M}) is a function $eval_{X,Y} : RUN(P \parallel Q) \times M^k \rightarrow M^2$, defined by the formula

$$eval_{X,Y}(\rho, init) = (X[\rho, init], Y[\rho, init]).(*)$$

1.2. Results of the Paper

In [A1], Anderson and Gouda proved that it is impossible to construct protocols of the form $P \parallel Q$ defined above, which also satisfy the following conditions for any initialization $init$:

the variables X, Y can only assume the values 0, 1,

$eval_{X,Y}(\rho, init) \in \{(0,1), (1,0)\}$, for all runs ρ ,

$eval_{X,Y}(p_1, \dots, p_m, q_1, \dots, q_n, init) = (0,1)$,

$eval_{X,Y}(q_1, \dots, q_n, p_1, \dots, p_m, init) = (1,0)$

(they call such protocols, binary disagreement protocols).

The present paper investigates even further the limitations of wait-free protocols, by analyzing and studying one of their main structural deficiencies, namely "their inability to make a processor wait". As a first step it was observed that the result mentioned above could be generalized to show that there exist no protocol $P \parallel Q$ such that the following conditions are met for any initialization $init$:

the variables X, Y can only assume the values 0, 1,

$eval_{X,Y}(\rho, init) \in \{(0,1), (1,0)\}$, for all runs ρ ,

$(\exists \rho, \sigma \in RUN)[eval_{X,Y}(\rho, init) \neq eval_{X,Y}(\sigma, init)]$.

Motivated from this, we define a new notion of functional dependency among protocol variables. Namely, we call the variables X, Y "belonging" to processors P, Q , respectively, functionally dependent for the initialization $init$, if for any runs ρ, σ of the protocol $P \parallel Q$,

$$X[\rho, init] = X[\sigma, init] \Leftrightarrow Y[\rho, init] = Y[\sigma, init].$$

Using this notion it is possible to provide characterizations of those programs for which the variables X, Y are functionally dependent in terms of the evaluation function $eval_{X,Y}$ of the program (see section 3). In fact we show that for any model \mathcal{M} of the given protocol and any possible

(*) The same remark as in the previous footnote applies to the notation used for the evaluation function $eval$.

initialization *init* of the protocol variables the following statements are equivalent:

- the variables X, Y are functionally dependent for the initialization *init*,
- the quantity $eval_{X,Y}(\rho, init)$ is independent of the run $\rho \in RUN$.

Intuitively, if both the binary relation $\{(X[\rho, init], Y[\rho, init]) : \rho \in RUN\}$ and its inverse are functions then the relation must be a singleton. This makes it possible to give very natural and elegant generalizations of the result of [A1] not only to multivalued variables (as opposed to boolean valued variables considered before), but also to multiprocessor protocols (see section 4). As in [A1] this implies the impossibility of constructing wait-free protocols for Mutual Exclusion. The main combinatorial lemma needed for our analysis is presented in section 2. Extensions to more general protocols are given in section 5.

2. A Combinatorial Principle

At the heart of the proof of the result on functional dependencies in wait-free programs lies a rather simple combinatorial principle. Before stating and proving this principle some definitions will be necessary. Let $A = \{a_1, a_2, \dots, a_m\}, B = \{b_1, b_2, \dots, b_n\}$ be two disjoint sets such that $|A| = m \geq 1, |B| = n \geq 1$. Let $[A, B]$ be the set of sequences $x = (x_1, \dots, x_{m+n})$ of elements of $A \cup B$ such that

$$\{x_1, \dots, x_{m+n}\} = A \cup B,$$

and if $a_i = x_{k(i)}, b_j = x_{l(j)}$, then both sequences $\langle k(i) : i = 1, \dots, m \rangle, \langle l(j) : j = 1, \dots, n \rangle$ are monotone increasing. For each i, j let $a^i = a_{i+1}, \dots, a_m$ be the "final" segment of the sequence (a_1, \dots, a_m) starting from a_{i+1} , and similarly $b^j = b_{j+1}, \dots, b_n$. For $i, j \geq 1$, call a sequence $x \in [A, B]$, $\{a_i, b_j\}$ -separated if x is of one of the following four forms

$$(s, a_i, b_j, a^i, b^j), (s, a_i, b_j, b^j, a^i), (s, b_j, a_i, a^i, b^j), (s, b_j, a_i, b^j, a^i),$$

where s is an arbitrary finite sequence of elements of $A \cup B$ of the appropriate length ($= i+j-2$). For any $\{a_i, b_j\}$ -separated sequence x let $x(a_i, b_j)$ (respectively, $x(a^i, b^j)$) be the sequence obtained from x by interchanging the position of a_i, b_j (respectively, a^i, b^j) in x . An elementary interchange of the type $x \rightarrow x(a_i, b_j)$ is called one-step interchange. Let $F : [A, B] \rightarrow S$ be a function defined on all the sequences in $[A, B]$ and with range the nonempty set S . Then we can prove the following theorem.

Theorem 1. (Combinatorial Theorem)

Assume that for some $i, j \geq 1$ there is an $\{a_i, b_j\}$ -separated sequence $x \in [A, B]$ such that $F(x) \neq F(x(a_i, b_j))$. If $x \in [A, B]$ is an $\{a_i, b_j\}$ -separated sequence such that $i+j$ is maximal with $F(x) \neq F(x(a_i, b_j))$ then we have that $F(x(a^i, b^j)) \neq F(x(a^i, b^j)(a_i, b_j))$.

Proof. Let $x \in [A, B]$ be an $\{a_i, b_j\}$ -separated sequence such that $i+j$ is maximal with $F(x) \neq F(x(a_i, b_j))$. Clearly, in order to prove the theorem it is enough to show that both equations below

$$F(x) = F(x(a^i, b^j)), \tag{1}$$

and

$$F(x(a_i, b_j)) = F(x(a^i, b^j)(a_i, b_j)) \tag{2}$$

are true. We prove only (1). The proof of (2) is similar. Without loss of generality assume that $x = (s, a_i, b_j, a^i, b^j)$. The idea of the proof is to transform the given sequence x into the sequence $x(a^i, b^j)$ in stages via sufficiently many one-step interchanges.

Stage 1. Interchange the position of a_m and each b_s ($s = j+1, \dots, n$) one at a time and let

$$x_{m,j} = x, x_{m,j+1} = x(a_m, b_{j+1}), x_{m,j+2} = x_{m,j+1}(a_m, b_{j+2}), \dots, x_{m,n} = x_{m,n-1}(a_m, b_n),$$

be the resulting sequences.

Stage 2. Start from the sequence $x_{m,n}$, interchange the position of a_{m-1} and each b_s ($s = j+1, \dots, n$) one at a time, and let

$$x_{m-1,j+1} = x_{m,n}(a_{m-1}, b_{j+1}), x_{m-1,j+2} = x_{m-1,j+1}(a_{m-1}, b_{j+2}), \dots, x_{m-1,n} = x_{m-1,n-1}(a_{m-1}, b_n),$$

be the resulting sequences. Continue in this manner.

Final Stage. Start from the sequence $x_{i+2,n}$, interchange the position of a_{j+1} and each b_s ($s = j+1, \dots, n$) one at a time, and let

$$x_{i+1,j+1} = x_{i+2,n}(a_{i+1}, b_{j+1}), x_{i+1,j+2} = x_{i+1,j+1}(a_{i+1}, b_{j+2}), \dots, x_{i+1,n} = x_{i+1,n-1}(a_{i+1}, b_n),$$

be the resulting sequences. Clearly, $x = x_{m,j}$, $x_{i+1,n} = x(a^i, b^j)$. It follows from the maximality of $i + j$ that the function F assumes the same value on all the above sequences, i.e.

$$\begin{aligned} F(x_{m,j}) &= F(x_{m,j+1}) = \dots = F(x_{m,n}) = \\ F(x_{m-1,j+1}) &= F(x_{m-1,j+2}) = \dots = F(x_{m-1,n}) = \\ &\dots \\ F(x_{i+1,j+1}) &= F(x_{i+1,j+2}) = \dots = F(x_{i+1,n}). \end{aligned}$$

This shows that $F(x) = F(x(a^i, b^j))$ and completes the proof of part (1) of the theorem. The proof of part (2) is similar. •

The combinatorial theorem, as well as its proof will be used frequently in the sequel.

3. Two Processor Programs

In this section we prove the main result on functional dependencies for 2-processor protocols. Suppose that we are given two processes P, Q which are executing concurrently and asynchronously the atomic assignment statements p_1, p_2, \dots, p_m and q_1, q_2, \dots, q_n . We assume that each of the p_i, q_j is an atomic assignment statement of the type $x_i := F_i(w_i)$, $y_j := G_j(z_j)$, respectively, where the variables satisfy the following atomicity conditions ([A1]):

Variable Atomicity Conditions:

- The sets $\{x_1, x_2, \dots, x_m\}$, $\{y_1, y_2, \dots, y_m\}$ of program variables are mutually disjoint.
- The atomic statements p_i satisfy the following conditions:
 - either x_i is a local variable of P , and $F_i \in \Sigma$ is a function symbol and the variables $w_i = w_{i,1}, \dots, w_{i,k_i}$ are either local or read variables of the process P ,
 - or x_i is a write variable of P , and $F_i \in \Sigma$ is a function symbol and the variables $w_i = w_{i,1}, \dots, w_{i,k_i}$ are local variables of the process P .
- The atomic statements q_j satisfy the following conditions:
 - either y_j is a local variable of Q , and $G_j \in \Sigma$ is a function symbol and the variables $z_j = z_{j,1}, \dots, z_{j,l_j}$ are either local or read variables of the process Q ,
 - or y_j is a write variable of Q , and $G_j \in \Sigma$ is a function symbol and the variables $z_j = z_{j,1}, \dots, z_{j,l_j}$ are local variables of the process Q .

Such a program will be denoted by $P \parallel Q$. The next theorem ties the notion of functional dependencies of variables with the evaluation mapping $eval_{X,Y}$ of the program $P \parallel Q$. This generalizes the main result of [A1] to the case of multivalued variables.

Theorem 2. (Two Processor Functional Dependencies)

Let $P \parallel Q$ be any wait-free program with X, Y variables of P, Q respectively. Let RUN be the set of all possible runs of $P \parallel Q$. Then for any initialization $init$ of the protocol variables (in a given model \mathcal{M}) the following statements are equivalent:

- (1) The variables X, Y are functionally dependent for the initialization $init$.
- (2) The quantity $eval_{X,Y}(\rho, init)$ is independent of the run $\rho \in RUN$.

Proof. The implication (2) \Rightarrow (1) is trivial. So we will only concentrate on the proof of (1) \Rightarrow (2). As in theorem 1 we use the notation: $p^i = p_{i+1}, \dots, p_m, q^j = q_{j+1}, \dots, q_n$. Fix any initialization $init$ of all the protocol variables. First of all we prove the following claim.

Claim 1. For all i, j and all $\{p_i, q_j\}$ -separated sequences ρ ,

$$eval_{X,Y}(\rho, init) = eval_{X,Y}(\rho(p_i, q_j), init).$$

Proof of Claim 1. Assume on the contrary that there exist i, j and a $\{p_i, q_j\}$ -separated sequence ρ such that $eval_{X,Y}(\rho, init) \neq eval_{X,Y}(\rho(p_i, q_j), init)$. For the given initialization $init$ let the function F be defined on the set RUN of runs of the protocol by $F(\rho) = eval_{X,Y}(\rho, init)$. Clearly, for any initialization of the variables the set RUN can be identified with the set $[\{p_1, \dots, p_m\}, \{q_1, \dots, q_n\}]$ considered in the previous section. Let ρ be $\{p_i, q_j\}$ -separated, with $i + j$ maximal such that $F(\rho) \neq F(\rho(p_i, q_j))$. Without loss of generality assume that

$$\begin{aligned} \rho &= \dots p_i q_j q_{j+1} \dots q_n p_{i+1} \dots p_m, \\ \rho(p_i, q_j) &= \dots q_j p_i q_{j+1} \dots q_n p_{i+1} \dots p_m, \\ \rho(p^i, q^j) &= \dots p_i q_j p_{i+1} \dots p_m q_{j+1} \dots q_n, \\ \rho(p_i, q_j)(p^i, q^j) &= \dots q_j p_i p_{i+1} \dots p_m q_{j+1} \dots q_n. \end{aligned}$$

The variable dependencies that will be proved below are summarized in table 1. Recall that due to the assumption of the functional dependence of the variables X, Y , if ρ and σ are runs such that either $X[\rho, init] = X[\sigma, init]$ or else $Y[\rho, init] = Y[\sigma, init]$ then it is true that $eval_{X,Y}(\rho, init) = eval_{X,Y}(\sigma, init)$. This simple observation will be used frequently in the sequel.

x_i	y_j	Variable Equalities
local	local	$X[\rho] = X[\rho(p_i, q_j)], Y[\rho] = Y[\rho(p_i, q_j)]$
write	write	$X[\rho] = X[\rho(p_i, q_j)], Y[\rho] = Y[\rho(p_i, q_j)]$
local	write	$Y[\rho] = Y[\rho(p_i, q_j)]$
write	local	$X[\rho(p^i, q^j)] = X[\rho(p^i, q^j)(p_i, q_j)]$

Table 1: Variable Equalities in $P \parallel Q$.

If x_i were a local variable of P and y_j were a write variable of Q then $x_i := F_i(w_i)$, where the w_i are local or read variables of P and $y_j := G_j(z_j)$, where the z_j are local variables of Q . But then in the runs $\rho, \rho(p_i, q_j)$, the actions q_{j+1}, \dots, q_n do not see the value assigned to x_i by p_i . Moreover, since y_j is a write variable of Q its value does not depend on p_i . Hence, $Y[\rho] = Y[\rho(p_i, q_j)]$.

If either both x_i, y_j are local variables of P, Q respectively or else both x_i, y_j are write variables of P, Q respectively then $X[\rho] = X[\rho(p_i, q_j)], Y[\rho] = Y[\rho(p_i, q_j)]$.

Hence, the only case left is if y_j is a local variable of Q and x_i is a write variable of P . In view of theorem 1, $F(\rho(p^i, q^j)) \neq F(\rho(p^i, q^j)(p_i, q_j))$. However, since y_j is local to Q , we must have $X[\rho(p^i, q^j)] = X[\rho(p^i, q^j)(p_i, q_j)]$.

This gives contradictions in all four cases considered and completes the proof of claim 1.

Therefore for all i, j and all $\{p_i, q_j\}$ -separated sequences ρ ,

$$\text{eval}_{X,Y}(\rho, \text{init}) = \text{eval}_{X,Y}(\rho(p_i, q_j), \text{init}). \quad (3)$$

But then it is not hard to show that (3) implies the conclusion of the theorem, i.e. there is a constant c such that for all runs ρ $\text{eval}_{X,Y}(\rho, \text{init}) = c$. More formally, the following claim is needed.

Claim 2. If the run ρ is $\{p_i, q_j\}$ -separated with $i + j > 2$ then there exists a $\{p_{i'}, q_{j'}\}$ -separated run ρ' such that $i' + j' < i + j$, $\text{eval}_{X,Y}(\rho, \text{init}) = \text{eval}_{X,Y}(\rho', \text{init})$.

Proof of claim 2. Suppose that $\rho = (s, p_i, q_j, p^i, q^j)$, where s is a sequence of length $i + j - 2$. By repeatedly applying (3) it can be shown that

$$\text{eval}_{X,Y}(\rho, \text{init}) = \text{eval}_{X,Y}(\rho(p_i, q_j), \text{init}) = \text{eval}_{X,Y}(\rho(p^i, q^j), \text{init}).$$

On the one hand, if the last element of s is p_{i-1} then put $\rho' = \rho(p_i, q_j)$, which is a $\{p_{i-1}, q_j\}$ -separated sequence. On the other hand, if the last element of s is q_{j-1} then put $\rho' = \rho(p^i, q^j)$, which is a $\{p_i, q_{j-1}\}$ -separated sequence. This completes the proof of claim 2.

To finish the proof of the theorem start with an arbitrary run ρ and interchange the position of its atomic assignment statements one by one, by performing one-step interchanges, just like in the proof of theorem 1, until ρ is transformed into the run $(p_1, \dots, p_m, q_1, \dots, q_n)$. That this can be done is guaranteed from the result of claim 2. Hence,

$$\text{eval}_{X,Y}(\rho, \text{init}) = \text{eval}_{X,Y}(p_1, \dots, p_m, q_1, \dots, q_n, \text{init}). \bullet$$

4. Multiprocessor Programs

As a byproduct of our analysis on functional dependencies we can now generalize the previous results to multiprocessor, wait-free programs consisting only of assignment statements. Indeed, let

$$P_1 \parallel P_2 \parallel \dots \parallel P_r \parallel Q_1 \parallel Q_2 \parallel \dots \parallel Q_s,$$

be a wait-free program of $r+s$ processors: P_1, P_2, \dots, P_r are the active processors, and Q_1, Q_2, \dots, Q_s are the dummy processors, with $r \geq 2, s \geq 0$. The definitions and assumptions outlined in the previous sections are still assumed true for the case of multiprocessor protocols. In the sequel, we stress the most important of these aspects. We assume that each processor P_i (respectively, Q_j) executes a sequence of "atomic" assignment statements $p_1^i, p_2^i, \dots, p_{m_i}^i$ (respectively, $q_1^j, q_2^j, \dots, q_{n_j}^j$), where $i = 1, \dots, r$ (respectively, $j = 1, \dots, s$). The p_k^i, q_k^j are atomic assignment statements of the form $x := F(w)$, where $x, w = w_1, \dots, w_t$ are variables of the corresponding process, and F is a function symbol in the language Σ . Let $W(P_i)$ be the set of variables x which are assigned a value by the process P_i , i.e. the set of variables x such that some assignment statement p_k^i of the process P_i is of the form $x := F(w)$.

Variable Atomicity Conditions:

- the sets $W(P_i)$ are pairwise mutually disjoint, for $i = 1, \dots, r$, i.e. $W(P_i) \cap W(P_i) = \emptyset$, for $i \neq j$.
- Suppose that $x := F(w)$ is any assignment statement of processor P_i . Then the variables $x, w = w_1, \dots, w_t$ are supposed to satisfy the following conditions:

either x is a local variable of P_i , and F is a function symbol and the variables w are either local or read variables of the process P_i , or

x is a write variable of P_i , and F is a function symbol and the variables w are local variables of the process P_i .

As before, we are interested in program executions (or runs) $\rho = (\mathcal{A}, <_{\rho})$ of the above program, where \mathcal{A} is the set of atomic assignment statements of the program. The order of actions in a run ρ is determined by the relation $<_{\rho}$, i.e. $p <_{\rho} p'$ if and only if p immediately precedes p' in the run ρ , where $p, p' \in \mathcal{A}$. We assume that the transitive closure \rightarrow_{ρ} of $<_{\rho}$ is a partial ordering on the set \mathcal{A} such that the "natural ordering" of execution among the actions— $\{p_k^i\}$ for P^i , and $\{q_k^j\}$ for Q_j — of the program is preserved,

$$\begin{aligned} p_1^i &\rightarrow_{\rho} \cdots \rightarrow_{\rho} p_{m_i}^i, i = 1, \dots, r, \\ q_1^j &\rightarrow_{\rho} \cdots \rightarrow_{\rho} q_{k_j}^j, j = 1, \dots, s. \end{aligned}$$

For any run ρ of the protocol and any initialization $init$ of the variables let $X[\rho, init]$ be the value of the variable X at the end of the run ρ , when all the variables are initialized by $init$ (the formal definition of this which is given in introduction can be generalized easily). For each $i = 1, \dots, r$ let $X_i \in W(P_i)$ (in this case we say that the variable X_i belongs to the processor P_i). As before, for any run ρ and any initialization $init$ of the program variables, define the evaluation mapping of the program by

$$eval_{X_1, X_2, \dots, X_r}(\rho, init) = (X_1[\rho, init], X_2[\rho, init], \dots, X_r[\rho, init]).$$

Call X_1, X_2, \dots, X_r functionally dependent if the following holds for any initialization $init$, where ρ, σ range over runs of the protocol $P_1 \parallel P_2 \parallel \cdots \parallel P_r \parallel Q_1 \parallel Q_2 \parallel \cdots \parallel Q_s$:

$$\forall \rho, \sigma \forall i, j (X_i[\rho, init] = X_i[\sigma, init] \Leftrightarrow X_j[\rho, init] = X_j[\sigma, init]).$$

In general, the values obtained by the evaluation function $eval_{X_1, X_2, \dots, X_r}(\rho, init)$ depend on the initialization $init$ of the protocol variables as well as on the protocol run ρ . However, as before we can prove a necessary and sufficient condition for the evaluation mapping to be independent of the given run ρ . This is done in the following theorem.

Theorem 3. (Multiprocessor Functional Dependencies)

Let X_1, X_2, \dots, X_r be variables belonging to the active processors P_1, P_2, \dots, P_r , respectively, of the wait-free, multiprocessor program $P_1 \parallel P_2 \parallel \cdots \parallel P_r \parallel Q_1 \parallel Q_2 \parallel \cdots \parallel Q_s$, with $r \geq 2, s \geq 0$. Let RUN be the set of all its possible runs. Then for any possible initialization $init$ of the protocol variables (in a given model \mathcal{M}) the following statements are equivalent:

- (1) The variables X_1, X_2, \dots, X_r are functionally dependent for the initialization $init$.
- (2) The quantity $eval_{X_1, X_2, \dots, X_r}(\rho, init)$ is independent of the run $\rho \in RUN$.

Proof. Clearly, theorem 2 proved in the previous section corresponds to the case $r = 2, s = 0$. The implication (2) \Rightarrow (1) is trivial. Hence it only remains to prove the reverse implication. Assume that (1) is true. We want to show that (2) is true, as well. The proof of the present theorem is via two reductions. First we show that the special case $r = 2$ of the theorem implies the more general case $r \geq 2$. Next we show that theorem 2 implies the present theorem in the case $r = 2, s \geq 0$. Obviously, this is enough in order to give a complete proof of the theorem.

Claim. Without loss of generality we can assume $r = 2$.

Proof of the claim. Indeed, assume that the theorem is true for $r = 2$. It will be shown that the theorem is true for any arbitrary $r \geq 2$. Let $i < j \leq r$ be arbitrary, but fixed. Consider the program

$$P_i \parallel P_j \parallel (P_1 \parallel P_{i-1} \parallel P_{i+1} \parallel \cdots \parallel P_{j-1} \parallel P_{j+1} \parallel \cdots \parallel P_r \parallel Q_1 \parallel Q_2 \parallel \cdots \parallel Q_s) \quad (4)$$

with selected variables X_i, X_j and evaluation mapping $eval_{X_i, X_j}(\rho, init)$. By the assumption that

the theorem is true for $r = 2$, if the variables X_i, X_j are functionally dependent in the program (4) then the quantity determined by its evaluation function $eval_{X_i, X_j}(\rho, init)$ is independent of the run ρ . But by assumption (1) of the theorem the variables X_1, X_2, \dots, X_r are functionally dependent. Hence, for all i, j the quantity determined by the evaluation function $eval_{X_i, X_j}(\rho, init)$ is independent of the run ρ . But then it follows immediately that the quantity $eval_{X_1, X_2, \dots, X_r}(\rho, init)$ is also independent of the run ρ . This completes the proof of the claim.

In view of the claim just proved we can assume without loss of generality that we have the program $P \parallel Q \parallel Q_1 \parallel Q_2 \parallel \dots \parallel Q_s$, with $Q_1 \parallel Q_2 \parallel \dots \parallel Q_s$ the dummy processors, and two variables X, Y belonging to the active processors P, Q , respectively, which are functionally dependent, i.e. for any runs ρ, σ of the protocol $P \parallel Q \parallel Q_1 \parallel Q_2 \parallel \dots \parallel Q_s$,

$$X[\rho, init] = X[\sigma, init] \Leftrightarrow Y[\rho, init] = Y[\sigma, init]. \quad (5)$$

Let R be the set of all possible runs of the program $P \parallel Q_1 \parallel Q_2 \parallel \dots \parallel Q_s$. Any run $\rho \in R$ gives rise to a program $P_\rho \parallel Q$, where the processor P_ρ is executing the sequence of assignment statements determined by the run ρ , while Q is executing the sequence of assignment statements it was executing before in the program $P \parallel Q \parallel Q_1 \parallel Q_2 \parallel \dots \parallel Q_s$. For each run $\rho \in R$ let $eval_{\rho, X, Y}$ be the evaluation mapping of the program $P_\rho \parallel Q$. It is clear that for all runs $\rho \in R$,

$$RUN(P_\rho \parallel Q) \subseteq RUN(P \parallel Q \parallel Q_1 \parallel \dots \parallel Q_s).$$

In view of theorem 2, and equivalence (5) the quantity $eval_{\rho, X, Y}(\sigma, init)$ is independent of the run $\sigma \in RUN(P_\rho \parallel Q)$. Put $c_\rho(init) = eval_{\rho, X, Y}(\sigma, init)$. It remains to show that for all runs $\rho, \rho' \in R$, and all initializations $init$, $c_\rho(init) = c_{\rho'}(init)$. To this effect, let $\rho, \rho' \in R$ be two arbitrary but fixed runs, and consider the following two new runs of the program $P \parallel Q \parallel Q_1 \parallel \dots \parallel Q_s$:

$$\sigma : q_1, q_2, \dots, q_n, \rho \in RUN(P_\rho \parallel Q)$$

$$\sigma' : q_1, q_2, \dots, q_n, \rho' \in RUN(P_{\rho'} \parallel Q),$$

i.e. σ (respectively, σ') is formed by executing the sequence of assignment statements q_1, \dots, q_n followed by the assignment statements occurring in ρ (respectively, in ρ'). Let $init$ be any initialization of the variables. Clearly,

$$eval_{X, Y}(\sigma, init) = eval_{\rho, X, Y}(\sigma, init) = c_\rho(init)$$

$$eval_{X, Y}(\sigma', init) = eval_{\rho', X, Y}(\sigma', init) = c_{\rho'}(init).$$

Moreover, since in the run σ (respectively σ') the subrun ρ (respectively ρ') cannot influence the value attained by the variable Y , it is immediate that $Y[\sigma, init] = Y[\sigma', init]$. Since by the assumption of the theorem the variables X, Y are functionally dependent it follows that also $X[\sigma, init] = X[\sigma', init]$. Hence,

$$c_\rho(init) = eval_{X, Y}(\sigma, init) = eval_{X, Y}(\sigma', init) = c_{\rho'}(init).$$

This completes the proof of the theorem. •

4.1. Examples

Now a few illuminating examples are in order.

Example 1. The reader should pay special attention to the variable conditions mentioned at the beginning of this section; they are quite important for the validity of theorem 2. This is easily seen in the following example. Let P, Q be two processors executing the statements $p : "X := Y + 1"$, and $q : "Y := X + 1"$, respectively (where $+$ denotes modulo 2 addition). Further, suppose that X is a write variable and Y is a read variable of P (respectively, Y is a write variable and X is a read variable of Q). Consider the runs $\rho = (p, q)$ and $\sigma = (q, p)$, and the initialization

$init = (0, 0)$ of the variables X, Y . It is then easy to see that at the end of the execution of the runs ρ, σ , $X[\rho, init] = 1, Y[\rho, init] = 2, X[\sigma, init] = 2, Y[\sigma, init] = 1$. Hence, the variables X, Y are functionally dependent for the initialization $init$, but the values assumed by the evaluation mapping $eval_{X,Y}(\cdot, init)$ are not independent of the run, since

$$eval_{X,Y}(\rho, init) = (1, 2) \neq (2, 1) = eval_{X,Y}(\sigma, init).$$

Example 2. If even one processor is allowed to execute a waiting loop then theorem 2 is false. For such an example the reader is referred to [A1].

Example 3. The following example illustrates theorem 2. Suppose that $P \parallel Q$ is a wait-free program, with distinguished variables X, Y belonging to the processes P, Q , respectively. Let $m = (M, F^m, G^m, \dots)$ be a model of the protocol and suppose that $f : M \rightarrow M$ is a one-to-one function. An immediate consequence of the theorem is that the following claim can be proved. Let $init$ be any initialization of all the protocol variables. If for any run ρ of the protocol $P \parallel Q$, $f(X[\rho, init]) = Y[\rho, init]$ then $eval_{X,Y}(\rho, init) = (X[\rho, init], f(X[\rho, init]))$ is independent of the run ρ .

Example 4. Clearly, theorem 2 (and its extension given in the next section) implies that it is impossible to construct wait-free, binary disagreement protocols. Further, it is shown in [A1] that it is impossible to implement Mutual Exclusion without waiting. According to theorem 3 (and its extension given in section 5) this is also the case even if we assume that a finite number Q_1, \dots, Q_s of "dummy" processors is present.

5. Extensions to More General Protocols

The previous two theorems on functional dependencies can easily be extended to programs, which—in addition to assignment statements—include the following additional types of statement constructions: **if ... then ... else ... fi**; **for $i := 1, \dots, n$ do ... od**. We extend the language Σ by adding relation symbols R, S, \dots , (with subscripts and/or superscripts) each of a certain arity ≥ 1 . Statements of the form $R(v_1, \dots, v_n)$ are called primitive statements. In addition, to assignment statements now we also have boolean statements, i.e. boolean combinations of primitive statements. The class of program statements is the smallest class of statements such that assignment statements are program statements, and is closed under

- (a) **if α then p else p' fi**, where α is a boolean statement,
- (b) **for $i := 1, \dots, n$ do p od**,

where p, p' are sequences of program statements. A processor P will now be executing a finite sequence of program statements each of which must satisfy one of the following conditions [A1]:

- if it is an assignment statement then its variables satisfy the variable atomicity conditions for the processor P (see section 3),
- if it is a program statement of the form (a) above then all the variables occurring in α must be local to P ,
- if it is a program statement of the form (b) above then both variables i, n must be local to P , and p has no assignments to either i or n .

An interpretation or model of the protocol is defined as before. For any run ρ and any initialization $init$ the definition of $X[\rho, init]$ as well as of the evaluation function $eval_{X,Y}$ is similar to the definition given in the introduction (however, the definition of $X[\rho, init]$ is given by induction on the length of ρ and the construction of the protocol formulas). In the sequel we outline a proof of the validity of theorems 2 and 3 in this more general context.

Proof of Theorems 2 and 3 (outline). Suppose we are given a program $P \parallel Q$ performed

by processors P and Q , executing the program statements ϕ_1, \dots, ϕ_m and ψ_1, \dots, ψ_n , respectively, and let X, Y be variables belonging to P, Q , respectively. We add extra local variables to the language according to the rules below in such a way that to each initialization $init$ (in the old language) there corresponds an initialization \overline{init} (in the new language), and vice versa. The main idea is to replace each program statement of type (b) or (c) with an appropriate sequence of assignment statements and form a new protocol $\overline{P} \parallel \overline{Q}$ such that:

- \overline{P} (respectively, \overline{Q}) executes the sequence of assignment statements $\overline{\phi}_1, \dots, \overline{\phi}_m$ (respectively, (respectively, $\overline{\psi}_1, \dots, \overline{\psi}_n$),
- to every run $\rho \in RUN(P \parallel Q)$ there corresponds a run $\overline{\rho} \in RUN(\overline{P} \parallel \overline{Q})$ such that for any initialization $init$

$$X[P \parallel Q, \rho, init] = X[\overline{P} \parallel \overline{Q}, \overline{\rho}, \overline{init}], Y[P \parallel Q, \rho, init] = Y[\overline{P} \parallel \overline{Q}, \overline{\rho}, \overline{init}], \quad (6)$$

and vice versa. The essential details of the construction are as follows (see [A1]). Take the first **for** statement occurring in P , say **for** $i := 1, \dots, n$ **do** p **od, and consider the sequence s of statements preceding it in P . Let N be the maximal value of n over all concurrent executions of s and Q . Now replace the above statement with the following sequence of $2N$ statements: $i := 1$; **if** $i > j$ **then skip else** p **fi**; ..., $i := N$; **if** $i > j$ **then skip else** p **fi**, where **skip** is an assignment statement of the form $x := x$, for some local variable x . We can thus eliminate one by one all **for** statements. Next we eliminate the **if** statements; since such statements can be nested they are eliminated one by one starting with the inner-most one. For every write variable x introduce a new local variable \overline{x} and replace each assignment statement $x := F$ with the two assignment statements $x := F$; $\overline{x} := F$. Replace **if** α **then** p **else** p' **fi** by the sequence $a := \alpha$; $p[a]$; $p'[-a]$, where a is a new local variable and $p[a]$ is the same sequence as p except that each assignment statement $x := F$ in p with x a write variable is replaced in $p[a]$ by $x := F$, if a , and $x := x$ otherwise, and each assignment statement $x := F$ in p with x a local variable is replaced in $p[a]$ by $x := F$, if a , and $x := \overline{x}$ otherwise, and similarly for $p'[-a]$.**

Now suppose that the variables X, Y are functionally dependent in the protocol $P \parallel Q$. Then we must show that the variables X, Y will be functionally dependent in the protocol $\overline{P} \parallel \overline{Q}$, as well. This follows from the fact that every run of $\overline{P} \parallel \overline{Q}$ is "essentially" of the type $\overline{\rho}$, for some run ρ of $P \parallel Q$. It follows from theorem 2 that the quantity $eval_{X,Y}(\overline{P} \parallel \overline{Q}, \overline{\rho}, \overline{init})$ is independent of the run $\overline{\rho}$. Hence, it follows from equality (6) that $eval_{X,Y}(P \parallel Q, \rho, init)$ is also independent of the run ρ , as desired. A similar proof will work for multiprocessor programs. •

Acknowledgements

Discussions with Paul Vitányi and John Tromp are gratefully acknowledged.

REFERENCES

- [A1] J. H. Anderson and M. G. Gouda, *The Virtue of Patience: Concurrent Programming with and without Waiting (Draft)*, University of Texas, Department of Computer Science, 78712-1188, 1987.
- [A2] Anderson, J. H., Gouda, M. G., and Singh, A. K., *The Elusive Atomic Register*, Proceedings of 6th ACM Symposium on Principles of Distributed Computing, Vancouver, Canada, 1987.
- [B] Bloom, B., *Constructing Two-writer Atomic Registers*, Proceedings of 6th ACM Symposium on Principles of Distributed Computing, Vancouver, Canada, 1987.
- [D] Dijkstra, E. W., *A Solution to a Problem in Concurrent Programming Control*, Comm. ACM, Vol. 8, No. 9, p. 569, 1965.

- [H] Herlihy, M. P., *Impossibility and Universality Results for Wait-Free Synchronization*, Proceedings of 7th ACM Symposium on Principles of Distributed Computing, 1988.
- [K] Kirousis, L. M., Kranakis, E., and Vitányi, P. M. B., *Atomic Multireader Register*, 2nd International Workshop on Distributed Algorithms, Amsterdam 1987, Springer Verlag Lecture Notes in Computer Science, 312 (1988), pp. 278-296.
- [L1] Lamport, L., *On Interprocess Communication, Part I: Basic Formalism, Part II: Algorithms*, Distributed Computing, vol. 1, pp. 77-101, 1986.
- [L2] Li, M. and P. Vitányi, *A Very Simple Construction for Atomic Multiwriter Register*, in proceedings of ICALP 1989.
- [M] Misra, J., *Axioms for Memory Access in Asynchronous Hardware Systems*, ACM Transactions on Programming Languages and Systems Vol. 8, No. 1, pp. 142-153, Jan. 1986.
- [N] Newman-Wolfe, R., *A Protocol for Wait-Free, Atomic, Multi-Reader Shared Variables*, Proceedings of 6th ACM Symposium on Principles of Distributed Computing, Vancouver, Canada, 1987.
- [P] Peterson, G.L. and J.E. Burns, *Concurrent Reading While Writing I*, Proceedings of 6th ACM Symposium on Principles of Distributed Computing, Vancouver, Canada, 1987.
- [T] Tromp, J., *How to Construct an Atomic Variable*, in present proceedings.
- [V] Vitányi, P. M. B., and Awerbuch, B., *Atomic Shared Register Access by Asynchronous Hardware*, 27th Annual Symposium on Foundations of Computer Science, Toronto, 1986.