ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2008/02(05)-0487-13
DOI:10.3778/j.issn.1673-9418.2008.05.004

E-mail:fcst@public2.bta.net.cn
http://www.ceaj.org
Tel:+86-10-51616056

Assignment: Operation on a Physical Object*

YUAN Chongyi+
School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China
+ Corresponding author: E-mail: lwyuan@pku.edu.cn

赋值:物理对象上的操作*

袁崇义+
北京大学 信息科学技术学院,北京 100871

摘 要:研究了命令式程序的形式语义。赋值被看成当作物理对象的变量上的操作。变量x一方面是个可以容纳数据值的物理对象,另一方面当它出现在数学表达式中时又代表它所容纳的值。作为物理对象,变量x允许它的值用读/写操作来观察或改变,读操作则是写操作的逆操作。事实上赋值就是对物理对象施加写操作。提出了与单变量赋值、多重赋值、顺序赋值及条件赋值等相对应的操作,提出了这些操作应服从的公理,并给出了用这些公理证明程序性质的实例。

关键词:赋值;命令式程序;物理对象;物理对象上的操作;操作公理;公理语义
文献标识码:A **中图分类号**:TP311

YUAN Chongyi. Assignment: operation on a physical object. Journal of Frontiers of Computer Science and Technology, 2008,2(5):487-499.

Abstract: This paper focuses on formal semantics of imperative programs. Assignments are viewed as operations on variables considered as a physical objects. Variable x is, on the one hand, a physical object which is able to hold a data value while on the other hand, it represents the value it currently holds when it appears in a mathematical expression. As a physical object, x allows its value to be observed and/or changed by read/write operations applied on it. Thus, assignments are in fact write-operations applied on physical objects. A read-operation is the reverse of write-operation. Operations corresponding to assignments on single

* the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312004,2002CB312006 (国家重点基础研究发展规划(973)); the National High-Tech Research and Development Plan of China under Grant No.2006AA04A119, 2006AA01Z160 (国家高技术研究发展计划(863)).
Received 2008-05, Accepted 2008-08.

variable, multi-variable sequential assignments and conditional assignments etc, are proposed. Axioms on these operations are also proposed as formal semantics of assignments, and examples are given to show how to verify program properties with these axioms.

Key words: assignment; imperative program; physical object; operation on physical object; operation axiom; axiom semantics

An imperative program consists of assignments and control mechanisms, including sequential control, choice, loop and possibly some parallel controls. We start this paper from the simplest assignments and axioms on such assignments.

1 Operation on a Physical Object

An assignment, in its simplest manner, consists of a variable, say x, and an expression, say e, with the assign symbol ":=" in between. The semantics (or meaning) of $x:=e$, according to [1], is "the assignment of the value of expression e to the variable x", or formally $x'=e$ where x' is, in the alphabet given in [1], "the final value of variable x" (when the execution of $x:=e$ properly terminates).

The predicate $x'=e$ may serve as a semantic description of $x:=e$, but it is hardly a formal one since it depends on the informal description of x'.

The variable x is a 2-fold object: it is, on the one hand, a mathematical object since whenever x appears in an expression, it represents the value x currently holds, while on the other hand, x is a physical object since it is the "name" of a memory location when x appears on the left of ":=".

People have so far mainly understood x as a mathematical object and as such, x is called a "variable" to emphasize the change of the value held by x. But x is, at the first place, the name of a physical object, i.e. the memory loc ation. This memory location has the capability to hold a value (of some type), and allows a write-into operation and a read-from operation to be applied on it. These invariant attributes of physical object x guarantee that x functions as a mathematical object (to be an operand of mathematical operators) and the value in x may change. The assignment $x:=e$ is nothing but a write-into operation applied on physical object x with the value of expression e as the second operand. We use an over_bar "$\bar{}$" as the write-into operator in $\bar{x}(e)$ where x and e are respectively the first and second operands. The read-from operator is a under_bar "$\underline{}$" in $\underline{x}(e_o)$ where x is the first operand and e_o, an operation expression, is the second operand. e_o serves to define a program for which the read_from operation on x is taken. The definition of these two operators, like the treatment of other mathematical operators, is given by

$$\bar{}: V\times E_m \to V_a$$

$$\underline{}: V_a\times E_o \to E_m$$

where V is the set of variables of the program (operation expression) in question and V_a is the subset of V consisting of assigned variables, E_m is the set of mathematical expressions and E_o is the set of operation expressions (to be defined in this paper). $\underline{x}(\bar{x}(e))$ is also written as $\underline{x}\bar{x}(e)$ as shown in axiom

1 below.

Axiom 1

$\underline{x}\bar{x}(e)=e$

Axiom 1 is, in our alphabet, the formal semantics of $x:=e$ where $x:=e$ is represented by $\bar{x}(e)$, and "the final value of x" (i.e. x') is now given by $\underline{x}\bar{x}(e)$.

A computer system consists of hardware and software. Axiom 1 demands the hardware to guarantee that whatever is read-from x must be the same as whatever was written into x by the write-into operation applied on x prior to the read-from operation. Furthermore, the two appearances of e in axiom 1 have different meanings: the e inside the parenthesis represents the values when e is evaluated by the software while the e on the right of "=" represents the value defined by mathematical operations in e. Axiom 1 requires the software to do computation in accordance with mathematics.

2 Multi-assignment: Synchronous Parallelism

A multi-assignment[2-3] assigns values to more than one variable in a synchronous way. For example $x,y:=e_x,e_y$ requires that e_x and e_y are computed first and then the values of e_x and e_y are assigned to x and y respectively.

Such a multi-assignment is, in our alphabet, given by operation expression $\bar{x}(e_x)\bar{y}(e_y)$, or $\bar{\bar{x}\bar{y}}(e_x,e_y)$. Let $V=\{x,y,\cdots,z\}$ be the set of all free variables of the program in question. The normal form of multi-assignments is $\bar{x}(e_x)\bar{y}(e_y)\cdots\bar{z}(e_z)$, or $\bar{\bar{x}\bar{y}\cdots\bar{z}}(e_x,e_y,\cdots,e_z)$. Generally speaking, a multi-assignment assigns values to some but not all, variables in V. The normal form of such a multi-assignment will appear as $\bar{\bar{x}\bar{y}\cdots\bar{z}}(e_x,e_y,\cdots,e_z)$ for $\bar{\bar{x}\bar{y}}(e_x,e_y)$, where each un-assigned variable, like z in this example, take the variable itself as its corresponding expression, e.g. $e_z=z$ for variable z in the above example.

The semantics of a multi-assignment is, in terms of normal form, formally given by axiom 2.

Axiom 2

$\underline{u}(\bar{x}(e_x)\bar{y}(e_y)\cdots\bar{z}(e_z))=\underline{u}\bar{u}(e_u)=e_u$ or $\underline{u}\bar{\bar{x}\bar{y}\cdots\bar{z}}(e_x,e_y,\cdots,e_z)=e_u$, for every $u,u\in V$

Applying axiom 2 to $\bar{\bar{x}\bar{y}}(e_x,e_y)$, we have:

$\underline{x}(\bar{\bar{x}\bar{y}}(e_x,e_y))=e_x$, $\underline{y}(\bar{\bar{x}\bar{y}}(e_x,e_y))=e_y$, $\underline{z}(\bar{\bar{x}\bar{y}}(e_x,e_y))=z$

It is deducible from these three equations:

$\bar{\bar{x}\bar{y}}(e_x,e_y)=\bar{\bar{y}\bar{x}}(e_y,e_x)$ and $x=y\Rightarrow e_x=e_y$

since a unique value is held by a single variable at any state.

It is assumed, in the normal form of a multi-assignment, that every variable appears in $\bar{V}$ exactly once. But when array elements are assigned by a multi-assignment, we don't always recognize two equal elements before the evaluation of their index expressions. Thus, it is not avoidable to have a single array element appearing in a multi-assignment more than once. The above conclusion $x=y\Rightarrow e_x=e_y$ and the commutative property $\bar{x}(e_x)\bar{y}(e_y)=\bar{x}(e_y)\bar{y}(e_x)$ enable us to relax the constraints on a multi-assignment: a single variable may appear more than once as long as the corresponding expressions are equal[2].

Assignment to a single variable is a special case

of multi-assignments, and as such it has also a normal form. Thus, axiom 1 is a special case of axiom 2. Or in other words, axiom 1 is implied by axiom 2.

Still another special case is $\bar{u}(u)$ where u is an arbitrary variable in V. $\bar{u}(u)$ functions as a skip statement, and its normal form is $\bar{V}(x,y,\cdots,z)$. For simplicity, we write $\bar{V}$ instead of $\bar{V}(x,y,\cdots,z)$, and we have, for every u, $u\in V$, $\underline{u}\bar{V}=u$.

3 Conditional Assignment: Choice

Let P and Q be assignments as given in section 2 in the form of operation expressions. Note that we will, in this paper, always define assignments in terms of operation expressions, and not distinguish an assignment and its corresponding normal form. Let b be a Boolean expression. The normal form of a conditional assignment is the operation expression $P^{b}Q^{\neg b}$.

Axiom 3

$\underline{u}(P^{b}Q^{\neg b})=$

$\underline{u}(P)$ if $b\sim\underline{u}(Q)$ if $\neg b$ for every u, $u\in V$

The expression $\underline{u}(P)$ if $b\sim\underline{u}(Q)$ if $\neg b$ is a conditional expression[2] whose normal form is

e_1 if $b_1\sim e_2$ if $b_2\sim\cdots\sim e_n$ if b_n $(n\geqslant 2)$

Let E be the value of above conditional expression, we have

$$E=\begin{cases}e_1 & \text{if } b_1\\ e_2 & \text{if } b_2\\ \quad\vdots\\ e_n & \text{if } b_n\end{cases}$$

It is required, in order to have the conditional expression well-defined, that $b_1\vee b_2\vee\cdots\vee b_n$ and $b_i\wedge b_j\Rightarrow e_i=e_j$ for $i\neq j$.

A special case of conditional assignments is $Q=\bar{V}$, i.e. Q functions as a skip statement. We write P^{b} instead of $P^{b}\bar{V}^{\neg b}$, and we have, by axiom 3, $\underline{u}(P^{b})=\underline{u}(P)$ if $b\sim u$ if $\neg b$.

4 Sequential Assignment

We adopt the conventional sequential operator ";" to form sequential assignments. Thus $P;Q$ denotes the execution of P followed by the execution of Q. Let $P_1,P_2,\cdots,P_n$ be assignments defined above, $n\geqslant 2$. Sequential assignments are given by operation expressions $P;Q$ and $P_1;\cdots;P_n$.

Axiom 4

$\underline{u}(P;Q)=\underline{u}(Q)(\underline{x}(P)/x,\underline{y}(P)/y,\cdots,\underline{z}(P)/z)$

$\underline{u}(P_1;P_2;\cdots;P_n)=\underline{u}(P_n)(\underline{x}(P_1;P_2;\cdots;P_{n-1})/x,\cdots,\underline{z}(P_1;P_2;\cdots;P_{n-1})/z)$

Where $u\in V$ is any variable in V, and $\underline{u}(Q)(\underline{x}(P)/x,\underline{y}(P)/y,\cdots,\underline{z}(P)/z)$ stands for the substitution in $\underline{u}(Q)$, i.e. x replaced by $\underline{x}(P),\cdots$, and z replaced by $\underline{z}(P)$. For simplicity, we often write $(\underline{x}(P),\cdots,\underline{z}(P))$ instead of $(\underline{x}(P)/x,\cdots,\underline{z}(P)/z)$.

Example 1

Let $R=\bar{x}(x+y);\bar{y}(x-y)$ we have

$\underline{x}(R)=\underline{x}(\bar{y}(x-y))(\underline{x}\bar{x}(x+y),\underline{y}\bar{x}(x+y))=$

$x(x+y,y)=x+y$

$\underline{y}(R)=\underline{y}(\bar{y}(x-y))(\underline{x}\bar{x}(x+y),\underline{y}\bar{x}(x+y))=$

$(x-y)(x+y,y)=x+y-y=x$

Example 2

Let $P=\bar{x}(x+y);\bar{y}(x-y);\bar{x}(x-y)$; i.e. $P=R;\bar{x}(x-y)$ where R is as given in example 1.

We have

$\underline{x}(P)=\underline{x}(\bar{x}(x-y))(\underline{x}(R),\underline{y}(R))=$

$(x-y)(x+y,x)=x+y-x=y$

$\underline{y}(P)=\underline{y}(\bar{x}(x-y))(\underline{x}(R),\underline{y}(R))=y(x+y,x)=x$

Apparently, P swaps the values in x and y.

Example 3

Let $Q=\bar{y}(x-y);\bar{x}(x-y)$, and $P'=\bar{x}(x+y);Q$

$\underline{x}(Q)=\underline{x}\bar{x}(x-y)(\underline{x}\bar{y}(x-y),\underline{y}\bar{y}(x-y))=$

$(x-y)(x,x-y)=x-(x-y)=y$

$\underline{y}(Q)=\underline{y}\bar{x}(x-y)(\underline{x}\bar{y}(x-y),\underline{y}\bar{y}(x-y))=y(x,x-y)=$

$x-y$ and

$\underline{x}(P')=\underline{x}(Q)(\underline{x}\bar{x}(x+y),\underline{y}\bar{x}(x+y))=y(x+y,y)=y$

$\underline{y}(P')=\underline{y}(Q)(\underline{x}\bar{x}(x+y),\underline{y}\bar{x}(x+y))=(x-y)(x+y,y)=$

$x+y-y=x$

Comparing $\underline{x}(P)$ and $\underline{y}(P)$ in example 2 with $\underline{x}(P')$ and $\underline{y}(P')$ in example 3, we have $P=P'$, i.e.

$(\bar{x}(x+y);\bar{y}(x-y));\bar{x}(x-y)=\bar{x}(x+y);(\bar{y}(x-y);\bar{x}(x-y))$

Theorem 1

(1)$(P;Q);R=P;(Q;R)$;

(2)$\underline{u}(P_{i+1};\cdots;P_n)=u\Rightarrow\underline{u}(P_1;P_2;\cdots;P_i;P_{i+1};\cdots;P_n)=\underline{u}(P_1;\cdots;P_i)$, for $1<i<n$, $u\in V$.

Proof

(1)Let $P=\bar{x}\bar{y}\cdots\bar{z}(e_{1x},e_{1y},\cdots,e_{1z})$ $Q=\bar{x}\bar{y}\cdots\bar{z}(e_{2x},e_{2y},\cdots,e_{2z})$, $R=\bar{x}\bar{y}\cdots\bar{z}(e_{3x},e_{3y},\cdots,e_{3z})$, and $u\in V$

$\underline{u}(P;Q)=\underline{u}(Q)(\underline{x}(P),\underline{y}(P),\cdots,\underline{z}(P))=$

$e_{2u}(e_{1x},e_{1y},\cdots,e_{1z})$

$\underline{u}((P;Q);R)=\underline{u}(R)(\underline{x}(P;Q),\underline{y}(P;Q),\cdots,\underline{z}(P;Q))=$

$e_{3u}(e_{2x}(e_{1x},\cdots,e_{1z}),e_{2y}(e_{1x},\cdots,e_{1z}),\cdots,e_{2z}(e_{1x},\cdots,e_{1z}))$

$\underline{u}(Q;R)=\underline{u}(R)(x(Q),\cdots,\underline{z}(Q))=e_{3u}(e_{2x},\cdots,e_{2z})$

$\underline{u}(P;(Q;R))=\underline{u}(Q;R)(x(P),\cdots,z(zP))=$

$e_{3u}(e_{2x},e_{2y},\cdots,e_{2z})(e_{1x},e_{1y},\cdots,e_{1z})=$

$e_{3u}(e_{2x}(e_{1x},\cdots,e_{1z}),\cdots,e_{2z}(e_{1x},\cdots,e_{1z}))$

Thus, $\underline{u}((P;Q);R)=\underline{u}(P;(Q;R))$ for all $u\in V$, i.e., $(P;Q);R=P;(Q;R)$.

(2)$\underline{u}(P_1,\cdots,P_i,P_{i+1},\cdots,P_n)=\underline{u}(P_{i+1};\cdots;P_n)(\underline{x}(P_1;\cdots;P_i),\underline{y}(P_1;\cdots;P_i),\cdots,\underline{z}(P_1;\cdots;P_i))=u(\underline{x}(P_1;\cdots;P_i),\cdots,\underline{u}(P_1;\cdots;P_i),\cdots,\underline{z}(P_1;\cdots;P_i))=\underline{u}(P_1;\cdots;P_i)$ for all u, $u\in V$ and $\underline{u}(P_{i+1},\cdots,P_n)=u$.

In proving theorem 1 above, $\underline{u}(P)$, $\underline{u}(P;Q)$, $\underline{u}(Q;R)$, $\underline{u}((P;Q);R)$ and $\underline{u}(P;(Q;R)$ are "final values of variable u" after respectively programs P, $P;Q$, $Q;R$, $(P;Q);R$ and $P;(Q;R)$. In contrast to x', "the final value of variable x"[1], these "final values of variable u" enable us to prove $(P;Q);R=P;(Q;R)$ while the same equation can only be a "law" as given in [1]. □

5 Power and Recursion

Let P be an assignment. Then P raised to power n, denoted as P^n, has axiom 5 as its formal semantics.

Axiom 5

$P^1=P$, i.e. $\underline{u}(P^1)=\underline{u}(P)$ for all u in V, $\underline{u}(P^n)=\underline{u}(P)(\underline{x}(P^{n-1}),\underline{y}(P^{n-1}),\cdots,\underline{z}(P^{n-1}))$ for $n>1$.

Example 4

$P=\bar{x}(x+1)$, $P^n=\bar{x}^n(x+1)$, $\underline{x}(P^n)=\underline{x}(P)(\underline{x}(P^{n-1}))=(x+1)(\underline{x}(P^{n-1}))=\underline{x}(P^{n-1})+1$

By mathematical induction, $\underline{x}(P^n)=x+n$.

Example 5

Let $Q=\bar{i}(i+1)\bar{f}(f*i)$ and $P=\bar{i}(1)\bar{f}(1);Q^n$ where n is a positive integer, $n\geqslant 1$. We have

$n=1\Rightarrow\underline{i}(P)=2$ and $\underline{f}(P)=1$

$n>1\Rightarrow \underline{i}(P)=n+1$ and $\underline{f}(P)=n*\underline{f}(P')$ where $P'=\bar{i}(1)\bar{f}(1);Q^{n-1}$

By mathematical induction, $\underline{f}(P)=n*(n-1)*\cdots*2*1$.

Thus, P is a program to compute $n!$.

In case P is a loop body, P^n is apparently a loop. When n goes to infinity, we have

$$P^*=\lim_{n\to\infty} P^n$$

and the next axiom.

Axiom 6

$\underline{u}(P^*)=\lim_{n\to\infty}(\underline{u}(P^n))$, for all u, $u\in V$.

Example 6

$P=\bar{x}(x+1)$

We know from example 4 that $\underline{x}(P^n)=x+n$ Thus

$$\underline{x}(P^*)=\lim_{n\to\infty}(\underline{x}(P^n))=\lim_{n\to\infty}(x+n)=\infty$$

In other words, P^* cannot reach a fixed point, or P^* does not have a fixed point.

There does not necessarily exist a limit in computing $\underline{u}(P^*)$. This will lead to a non-termination situation. We will consider termination issues in a separated paper.

Example 7

$P=\bar{x}^{x<7}(x+1)$, $P^n=(\bar{x}^{x<7}(x+1))^n$

$\underline{x}(P^n)=\underline{x}(P)(\underline{x}(P^{n-1}),\cdots,\underline{z}(P^{n-1}))=$

$(x+1$ if $x<7\sim x$ if $x\geqslant 7)(\underline{x}(P^{n-1}))=$

$\underline{x}(P^{n-1})+1$ if $\underline{x}(P^{n-1})<7\sim\underline{x}(P^{n-1})$ if $\underline{x}(P^{n-1})\geqslant 7$

Take the limit at both sides we have

$\underline{x}(P^*)=\underline{x}(P^*)+1$ if $\underline{x}(P^*)<7\sim\underline{x}(P^*)$ if $\underline{x}(P^*)\geqslant 7$

Thus, $\underline{x}(P^*)\geqslant 7$ since $\underline{x}(P^*)<7$ leads to $\underline{x}(P^*)=\underline{x}(P^*)+1$ according to the above equation, but $\underline{x}(P^*)\neq\underline{x}(P^*)+1$.

It is easy to prove that $x<7\Rightarrow\underline{x}(P^n)=7$ and $x\geqslant 7\Rightarrow\underline{x}(P^n)=x$ for $n\geqslant 1$. So, $x<7\Rightarrow\underline{x}(P^*)\leqslant 7$ and $x\geqslant 7\Rightarrow\underline{x}(P^*)=x$.

Example 8

To find the greatest common divisor of integers held by x,y when $x>0\wedge y>0$. Let $\gcd(x,y)$ be the greatest common divisor of x,y, and let $P=\bar{x}^{x>y}(x-y)\bar{y}^{y>x}(y-x)$. We will prove that $\underline{x}(P^*)=\underline{y}(P^*)=\gcd(x,y)$.

As we know from mathematics, $\gcd(L,M)=\gcd(L-M,M)$ if $L>M\sim\gcd(L,M-L)$ if $L<M\sim L$ if $L=M$ when $L>0\wedge M>0$. So we have $\gcd(x,y)=\gcd(\underline{x}(P),\underline{y}(P))$. From $x>0\wedge y>0$, we conclude $\underline{x}(P)>0\wedge\underline{y}(P)>0$. It's easy to extend these conclusions to P^n, for all positive integer n, $n\geqslant 1$.

$\gcd(x,y)=\gcd(\underline{x}(P^n),\underline{y}(P^n))$ and $\underline{x}(P^n)>0\wedge\underline{y}(P^n)>0$.

Thus, $\gcd(x,y)=\gcd(\underline{x}(P^*),\underline{y}(P^*))$and $\underline{x}(P^*)>0\wedge\underline{y}(P^*)>0$.

But $\underline{x}(P^*)=\underline{x}(P^*)-\underline{y}(P^*)$ if $\underline{x}(P^*)>\underline{y}(P^*)\sim\underline{x}(P^*)$ if $\underline{x}(P^*)\leqslant\underline{y}(P^*)$.

$\underline{y}(P^*)=\underline{y}(P^*)-\underline{x}(P^*)$ if $\underline{y}(P^*)>\underline{x}(P^*)\sim\underline{y}(P^*)$ if $\underline{y}(P^*)\leqslant\underline{x}(P^*)$.

since $\underline{x}(P^*)>0\wedge\underline{y}(P^*)>0$, $\underline{x}(P^*)\neq\underline{x}(P^*)-\underline{y}(P^*)\wedge\underline{y}(P^*)\neq\underline{y}(P^*)-\underline{x}(P^*)$, we have $\underline{x}(P^*)\leqslant\underline{y}(P^*)\wedge\underline{y}(P^*)\leqslant\underline{x}(P^*)$, that is $\underline{x}(P^*)=\underline{y}(P^*)$.

So $\gcd(x,y)=\gcd(\underline{x}(P^*),\underline{y}(P^*))=\underline{x}(P^*)=\underline{y}(P^*)$.

Remarks

(1)$\gcd(x,y)=\gcd(\underline{x}(P),\underline{y}(P))$ and $\underline{x}(P)>0\wedge\underline{y}(P)>0$ are invariants (safety properties) of P as defined in [2].

(2)P can be extended to compute the greatest common divisor of integers of any number. For example, to compute $\gcd(x,y,z)$, we need only to add $\bar{z}^{z>x}(z-x)$ to P:

$$R=\bar{x}^{x>y}(x-y)\bar{y}^{y>z}(y-z)\bar{z}^{z>x}(z-x)$$

and from $\underline{x}(R^*)\leqslant\underline{y}(R^*)\leqslant\underline{z}(R^*)\leqslant\underline{x}(R^*)$ we conclude:

$$\underline{x}(R^*)=\underline{y}(R^*)=\underline{z}(R^*)=\gcd(x,y,z)$$

(3)P^* is a loop of "repeat_until" style.

(4)$\underline{x}(P^*)=\underline{y}(P^*)$ defines a so_called "fixed point"[1-2] of P^*. The execution of P^* may terminate when, for some n, $\underline{x}(P^n)=\underline{y}(P^n)$.

6 Assignment to Array and Array Elements

6.1 Assign Array to Array

We may assign an array to another array: $\bar{A}(B)$, $\bar{A}(B+C)$ where A, B and C are arrays of the same size and type. To be more precise, they are indexed with the same index range.

Axiom 7

$$\underline{A}[i](\bar{A}[E])=E[i]$$

Where i is an arbitrary index of A, and E is an array expression like $B+C$.

6.2 Assignment to Array Elements

$\bar{A}[i](e)$ assigns e to given element $A[i]$:

Axiom 8

$$\underline{A}[j](\bar{A}[i](e))=e \text{ if } j=i\sim A[j] \text{ if } j\neq i$$

for all index j of A.

Sometimes assignment is applied to one or more array elements selected by Boolean expression b synchronously:

$$\bar{A}[i:b(i)](e(i))$$

Axiom 9

$$\underline{A}[j]\bar{A}[i:b(i)](e(i))=e(j) \text{ if } b(j)\sim A[j] \text{ if } \neg\, b(j)$$

for all index j of A.

Index values satisfying b are called instances of b. $\bar{A}[i:b(i)](e(i))$ assigns a value to each of the instances in synchronous manner.

Example 9

For all i, $0\leqslant i\leqslant N$ (N represents one natural number), assign $B[i]$ to $A[i]$ if $B[i]>A[i]$.

The instances in this example are determined by $0\leqslant i\leqslant N\wedge B[i]>A[i]$, and conventionally $0\leqslant i\leqslant N$ is given by default:

$$\bar{A}[i:B[i]>A[i]](B[i])$$

We have

$$\underline{A}[j](\bar{A}[i:B[i]>A[i]](B[i]))=B[j]$$
$$\text{if } B[j]>A[j]\sim A[j] \text{ if } B[j]\leqslant A[j]$$

for all j, $0\leqslant j\leqslant N$.

6.3 Loop: Sequential Assignment to Array Elements

We have seen in Section 5 how to loop with variables other than array elements. Often is the case that loops are applied to array elements. One way to do this is by power.

Example 10

To find the maximal element of $A[0...N]$.

What we have to do is to compute <max $i:0\leqslant i\leqslant N::A[i]$>. <max $i:0\leqslant i\leqslant N::A[i]$> is a quantified expression whose normal form is <op $i:b(i)::e(i)$> where op is a binary commutative operator, i is a variable, $b(i)$ is a Boolean expression and $e(i)$ is an expression. Instances of i are determined by $b(i)$

and with these instances, say $i_1, i_2, \cdots, i_n$, we obtain expressions $e(i_1)$, $e(i_2)$, $\cdots$, $e(i_n)$ and <op i: $b(i)$::$e(i)$> is defined as $e(i_1)$ op $e(i_2)\cdots$op $e(i_n)$. i is a bounded variable. A quantified expression may have more than one bounded variable.

Let us back to the example. We need auxiliary variables i and m of proper types. Let $P_0=\bar{i}(1)\bar{m}(A[0])$ and $P=\bar{i}(i+1)\bar{m}^{A[i]>m}(A[i])$. We will prove $\underline{m}(P_0;P^N)=$<max i:$0\leqslant i\leqslant N$::A[i]>. $\underline{i}(P_0)=1$ and $\underline{m}(P_0)=A[0]$ by axiom 2. From axiom 4 we have

$\underline{i}(P_0;P)=\underline{i}(P)(\underline{i}(P_0),\underline{m}(P_0))=(i+1)(1,A[0])=1+1=2$

$\underline{m}(P_0;P)=\underline{i}(P)(\underline{i}(P_0),\underline{m}(P_0))=$

$(A[i]$ if $A[i]>m$~m if $A[i]\leqslant m)(1,A[0])=$

$A[1]$ if $A[1]>A[0]$~$A[0]$ if $A[1]\leqslant A[0]=$

<max i:$0\leqslant i\leqslant 1$::$A[i]$>

Note that $\underline{i}(P)(\underline{i}(P_0),\underline{m}(P_0))$ stands for a substitution in $\underline{i}(P)$:i replaced by $\underline{i}(P_0)$ and m replaced by $\underline{m}(P_0)$.

It's easy to find $\underline{i}(P_0;P^2)=3$ and $\underline{m}(P_0;P^2)=$<max i:$0\leqslant i\leqslant 2$::$A[i]$>.

By mathematical induction, we have

$\underline{i}(P_0;P^N)=N+1$ and $\underline{m}(P_0;P^N)=$<max i:$0\leqslant i\leqslant N$::$A[i]$>

So $P_0;P^N$ is a solution for example 10. Apparently P^N has functioned as a loop of "for i=1 to n" style. The index range of A is $0\cdots N$, and P^N loops over only a portion of the index range. Let d and u be, respectively, the lower bound and the upper bound of the portion over which the program loops, $0\leqslant d<u\leqslant N$. It is assumed here the loop starts from index d and ends at index u. In case the loop index i goes from u down to d, $j=N-i$ will go up from $d'=N-u$ to $u'=N-d$. So, without lose of generality, we assume that the portion in question is always $d\cdots u$, $0\leqslant d<u\leqslant N$.

In the example above, actual assignments occur only when $A[i]>m$, i.e. $A[i]$ is assigned to m only when i is an instance of $A[i]>m$. Such sequential assignments will be denoted by $[A;\ i:\ b(i)](e(i))$. Note that d and u should appear in $b(i)$ and by default $d=0$, $u=N$.

Axiom 10

$(\bar{A}[;i:b(i)])(e(i))=$

$\bar{A}[0]^{b(0)}(e(0));\bar{A}[1]^{b(1)}(e(1));\cdots;\bar{A}[N]^{b(n)}(e(N))$

By axiom 4 and theorem 1, we have:

$\underline{A}[j](\bar{A}[;i:b(i)])(e(i))=A[j]$ if $0\leqslant j<d\vee u<j\leqslant N$~$\underline{A}[d](\bar{A}[d]^{b(d)}(e(d)))$ if $j=d$~$\underline{A}[j](\bar{A}[j]^{b(j)}(e(j)))(\underline{A}[d](P_{j-1})/A[d],\cdots,\underline{A}[j-1](P_{j-1}))/A[j-1])$ if $d<j\leqslant u$

where j is an index of A:$0\leqslant j\leqslant N$, and P_{j-1} stands for $\bar{A}[;i:d\leqslant i<j\wedge b(i)](e(i))$.

Example 11

To assign $A[i-1]+i$ to $A[i]$ for $i=1,2,\cdots,N$. Apparently $d=1$ and $u=N$ here. The solution is

$P=\bar{A}[;i:0<i\leqslant N](A[i-1]+i)$

And by axiom 10 we have

$\underline{A}[0](P)=A[0]$ since $0<d=1$

$\underline{A}[1](P)=\underline{A}[1](\bar{A}[1](A[0]+1))=A[0]+1$

$\underline{A}[2](P)=\underline{A}[2](\bar{A}[2](A[1]+2))(A[0]+1/A[1])=$

$(A[1]+2)(A[0]+1/A[1])=A[0]+1+2=$

$A[0]+3$

By mathematical induction we conclude: $\underline{A}[0](P)=A[0]$ and $\underline{A}[j](P)=A[0]+<+i:1\leqslant i\leqslant j::i>$ for j, $0<j\leqslant N$.

Sequential assignment to array instances provides

another loop style that is different from loop by power. It's easy to check that $\bar{i}(1);(\bar{i}(i+1)\bar{A}[i](A[i-1]+i))^N$ is also a solution to example 11.

6.4 Instance Pair of Array Elements

We know that $\bar{x}(y)\bar{y}(x)$ swaps the values held by x and y. It's sometimes necessary to do something to array elements selected as instance pairs:

$\bar{A}[i,j:b(i,j)](e_1(i,j),e_2(i,j))$

Axiom 11

$\underline{A}[u](\bar{A}[i,j;b(i,j)](e_1(i,j),e_2(i,j))=e_1(u,j)$

if $\exists j:b(u,j)\sim e_2(i,u)$ if $\exists i:b(i,u)\sim A[u]$

if $\forall i,j:\neg\, b(u,j)\wedge\neg\, b(i,u)$

Since a read-from operation on $A[u]$ must yield a unique value, we have, as a constraint on $\bar{A}[i,j;b(i,j)](e_1(i,j),e_2(i,j))$.

Theorem 2

For any u, $0\leqslant u\leqslant N$, $\forall i,j:b(u,j)\wedge b(i,u)\Rightarrow e_1(i,u)=e_2(u,j)\wedge\forall i_1,i_2:b(i_1,u)\wedge b(i_2,u)\Rightarrow e_1(i_1,u)=e_1(i_2,u)\wedge\forall j_1,j_2:b(u,j_1)\wedge b(u,j_2)\Rightarrow e_2(u,j_1)=e_2(u,j_2)$

Example 12

Sorting array $A[0\cdots n]$ to ascending order. Let $P=\bar{A}[i,i+1:i<n\wedge\text{even}(i)\wedge A[i]>A[i+1]](A[i+1],A[i])$

$Q=\bar{A}[i,i+1:i<n\wedge\text{odd}(i)\wedge A[i]>A[i+1]](A[i+1],A[i])$

We are going to prove that $(P;Q)^N$ and $(Q;P)^N$ are both solutions.

Let $A[s]$, $A[g]$ be the smallest and greatest elements of A respectively, where $0\leqslant s,g\leqslant N$. We have

$\text{even}(s)\wedge s>0\Rightarrow\underline{A}[s-1](Q)=A[s]$

In fact $\underline{A}[s-1](Q)=A[s]$ if $A[s-1]>A[s]\sim A[s-1]$ if $A[s-1]\leqslant A[s]$.

Since $A[s]$ is the smallest, $A[s-1]\leqslant A[s]\Rightarrow A[s-1]=A[s]$. Thus $\underline{A}[s-1](Q)=A[s]$ in any case. Similarly we have

$\text{Odd}(s)\wedge s>0\Rightarrow\underline{A}[s-1](P)=A[s]$

So

$\underline{A}[s-1](P;Q)=A[s]\vee\underline{A}[s-2](P;Q)=A[s]$

$\underline{A}[s-1](Q;P)=A[s]\vee\underline{A}[s-2](Q;P)=A[s]$

where $s-1\geqslant 0$, $s-2\geqslant 0$. This is to say, the smallest element will be at least one position nearer to index 0 in $\underline{A}(P;Q)$ and $\underline{A}(Q;P)$. Therefore, $\underline{A}[0]((P;Q)^N)=A[s]$, $\underline{A}[0]((Q;P)^N)=A[s]$. The above analysis applies to $A[g]$ and all other elements: they are at least one position nearer to their proper position in ascending order in $\underline{A}(P;Q)$ and $\underline{A}(Q;P)$, and they will be in the right positions in $\underline{A}((P;Q)^N)$ and $\underline{A}((Q;P)^N)$.

6.5 Instance of Array Elements Assigned Sequentially to a Single Variable

Let $\bar{m}[;i:b(i)](A[i])$ be an abbreviation of $\bar{m}(A[i_1]);\bar{m}(A[i_2]);\cdots;\bar{m}(A[i_n])$ where $i_1,i_2,\cdots,i_n$ are instance of i determined by $b(i)$. We have $P=\bar{m}(A[0]);\bar{m}[;i:0<i\leqslant N\wedge A[i]>m](A[i])$ and $\underline{m}(P)=<\max\ i:0\leqslant i\leqslant N::A[i]>$. To see this, we do the following computing:

$\underline{m}(\bar{m}(A[0]);\bar{m}(A[i_1]))=A[i_1](\underline{m}(\bar{m}(A[0])/m)=A[i_1]$

This is because i_1 is an instance of $A[i_1]>m$ where $m=\underline{m}(\bar{m}(A[0]))$, so $A[i_1]>A[0]$. And for i, $0<i<i_1$, i is not an instance: $A[i]\leqslant m$, i.e., $A[i]\leqslant A[0]$. This is to say

$\underline{m}(\bar{m}(A[0]);(A[i_1]))=<\max\ i:0\leqslant i\leqslant i_1::A[i]>$

By mathematical induction

$$\underline{m}(\bar{m}(A[0]);\bar{m}[i:0\leqslant i\leqslant N\wedge A[i]>m](A[i]))=$$
$$<\max\ i:0\leqslant i\leqslant i_n::A[i]>$$

But $A[i_n]$ is the last instance of $0\leqslant i\leqslant N\wedge A[i]>m$, we have, for all i, $i>i_n$, $A[i]\leqslant m$. So

$$\underline{m}(\bar{m}(A[0]);\ \bar{m}[;i:0\leqslant i\leqslant N\wedge A[i]>m](A[i]))=$$
$$<\max\ i:0\leqslant i\leqslant N::A[i]>$$

$\bar{m}[;i:b(i)](A[i])$ is just an abbreviation of sequential assignments; it does not need an axiom. Besides, $A[i]$ can be any expression as long as a value can be computed for each of the instances of i.

7 Asynchronous Parallel Assignment

The addition operator "+" is used to denote asynchronous parallel assignments. Let $P_1,P_2,\cdots,P_n$ be programs composed from assignments given in previous sections. The normal form of asynchronous parallel assignments is $P_1+P_2+\cdots+P_n$. The simplest case is n=2.

7.1 Simplest Case

For the simplest case, consider $P+Q$ (instead of P_1+P_2). Let V_p, V_q be the sets of free variables in P and Q respectively and $V_c=V_p\cap V_q$.

Axiom 12

$$V_c=\varnothing\Rightarrow\forall x\in V_p:\underline{x}(P+Q)=$$
$$\underline{x}(P)\wedge\ \forall u\in V_q:\underline{u}(P+Q)=\underline{u}(Q)$$

$V_c=\varnothing$ implies the fact that P and Q are independent with each other, and properties of $P+Q$ can be verified in terms of P or Q respectively.

It is reasonable to assume that all shared variables in V_c, when $V_c\neq\varnothing$, are communication channels between P and Q. We will further assume that communications between P and Q are synchronous, since asynchronous communication can be modeled via synchronous communication[2-5].

Let $c\in V_c$ be a communication channel. Both P and Q can initiate a communication via c. The initiator sends a message e out by performing a write-into operation on c, i.e. $\bar{c}(e)$, and at the other end of c, the receiver performs, when it is ready to receive, a read-from operation on c, followed by a write-into operation on a variable (say u) to store the received message: $\bar{u}(\underline{c})$. Thus, the pair $(\bar{c}(e),\bar{u}(\underline{c}))$ describes a complete communication action, and $\bar{c}(e),\bar{u}(\underline{c})$ are communication attempts over c. Note that the second operand is missing in c since this operand is given by the sender as explained below.

Definition 1

$(\bar{c}(e),\bar{u}(\underline{c}))$ is called a matched pair iff both $\bar{c}(e)$ and $\bar{u}(\underline{c})$ are the first communication attempt of respectively the sender and the receiver.

Note that by "the first communication attempt" we mean any such attempt no matter it is over c or over any other channels.

Let $(\bar{c}(e),\bar{u}(\underline{c}))$ be a matched pair, and as such, P and Q are decomposed as (assuming P is the sender):

$$P=P_1;\bar{c}(e);P_2$$

$$Q=Q_1;\bar{u}(\underline{c});Q_2$$

Axiom 13

(1) $P+Q=Q+P$

(2) $P+Q=(P_1;P_2)+(Q_1;\bar{u}(e');Q_2)$

Where $e'=e(\underline{V}_p(P_1)/V_p)$.

Note $e(\underline{V}_p(P_1)/V_p)$ is obtained from expression e by the substitution: every variable x in V_p is replaced by $\underline{x}(P_1)$ if x appears in e. It is clear now that the missing operand in $\underline{c}$ is P_j; $\bar{c}(e)$, that is the operation expression in P up to the sending action $\bar{c}(e)$ and it is given in Q by default.

$P+Q=Q+P$ indicates that P and Q play equal roles in $P+Q$, i.e. they enjoy equal right to be the sender or receiver in a communication attempt. The second pair of axiom 13 allows us to remove a matched pair from $P+Q$. Then we may detect another matched pair in $(P_1;P_2)+(Q_1;\bar{u}(e');Q_2)$ and remove it. In case all communication attempts are removed one after another, axiom 12 is applicable in verifying properties.

Definition 2

$P+Q$ is well matched iff all communication attempts belong to a matched pair detectable in the process of applying axiom 13 to remove them.

Theorem 3

$P+Q$ is deadlock-free if it is well matched.

This theorem is obviously true since all communication attempts can be removed one after another by axiom 13.

As a synchronous communication channel, c is occupied only when the communication is in progress. Besides, P (or Q) is involved in one complete communication action (i.e. a matched pair) at a time, we may well assume that $|V_c|=1$, i.e. there exists exactly one channel between P and Q. But we do not assume so since more channels bring no difference to the axioms.

7.2 Normal form $P_1+P_2+\cdots+P_n$

Let V_{pi} be the set of free variables in P_i, $i=1,\cdots,n$, and for $i\neq j$, $V_{ij}=V_{pi}\cap V_{pj}$. Variables in V_{ij} are communication channels between P_i and P_j. Let $c\in V_{ij}$ be such a channel, and $\bar{c}(e)$ and $\bar{u}(\underline{c})$ are the sending and receiving attempts respectively. Here Definition 1 is also valid in checking whether $(\bar{c}(e),\bar{u}(\underline{c}))$ is a well matched pair. In case this pair is well matched, P_i and P_j can be decomposed as (assuming P_i is the sender):

$P_i=P_{i1};\bar{c}(e);P_{i2}$

$P_j=P_{j1};\bar{u}(\underline{c});P_{j2}$

Axiom 14

(1)"+" is commutative in $P_1+P_2+\cdots+P_n$

(2)$P_1+\cdots+P_i+\cdots+P_j+\cdots+P_n=$

$P_1+\cdots+P_i'+\cdots+P_j'+\cdots+P_n$

Where $P_i'=P_{i1};P_{i2}$ and $P_j'=P_{j1};\bar{u}(e');P_{j2}$ in which $e'=e(\underline{V}_i(P_{i1})/V_i)$.

Note again that $e(\underline{V}_i(P_{i1})/V_i)$ is obtained from e by the substitution: every variable x in V_i is replaced by $\underline{x}(P_{i1})$ if x appears in e.

The difference between axiom 13 and 14 is, there may be more than one matched pair in $P_1+P_2+\cdots+P_n$ when $n>3$, and axiom 14 can be concurrently or asynchronously applied to all matched pairs in order to remove them.

The next definition is similar to definition 2:

Definition 3

$P_1+P_2+\cdots+P_n$ is well matched iff all communication attempts belong to a matched pair detectable in the process of applying axiom 14 consecutively.

Theorem 4

$P_1+P_2+\cdots+P_n$ is deadlock-free if it is well matched.

Example 13

$P=\bar{x}(1);\bar{c}(x);\bar{x}(x+1)$

$Q=\bar{y}(6);\bar{z}(\underline{c});\bar{y}(y+z)$

And c is a communication channel between P and Q, we have, by axiom 13

$P+Q=\bar{x}(1);\bar{x}(x+1)+\bar{y}(6);\bar{z}(e');\bar{y}(y+z)$ in which $e'=e(\underline{v}_p(P_1)/v_p)=x(\underline{x}\bar{x}(1)/x)=1$.

By axiom 13

$\underline{x}(P+Q)=\underline{x}(\bar{x}(1);\bar{x}(x+1))=\underline{x}\bar{x}(x+1)(\underline{x}\bar{x}(1)/x)=(x+1)(1/x)=1+1=2$

$\underline{y}(P+Q)=\underline{y}(\bar{y}(6);\bar{z}(e');\bar{y}(y+z))=\underline{y}(\bar{y}(6);\bar{z}(1);\bar{y}(y+z))=(y+z)(6/y,1/z)=7$

and

$\underline{z}(P+Q)=\underline{z}(\bar{y}(6);\bar{z}(e');\bar{y}(y+z))=1$

where $e'=e(\underline{v}_p(P_1)/v_p)=x(\underline{x}\bar{x}(1)/x)=1$.

Example 14

$P_1=\bar{x}(1);\bar{c}(x+1);\bar{x}(x+1)$

$P_2=\bar{y}(2);\bar{z}(\underline{c});\bar{y}(y+z)$

$P_3=\bar{u}(3);\bar{d}(u+1);\bar{u}(u+1)$

$P_4=\bar{v}(\underline{d});\bar{v}(2\cdot v)$

Where $c\in V_{12}$ and $d\in V_{34}$, i.e. c, d are channels respectively between P_1, P_2 and P_3, P_4. It's easy to find that $(\bar{c}(x+1),\bar{z}(\underline{c}))$ and $(\bar{d}(u+1),\bar{v}(\underline{d}))$ are both matched pairs in $P_1+P_2+P_3+P_4$. Applying axiom 14 concurrently to P_1+P_2 and P_3+P_4, we have

$P_1+P_2+P_3+P_4=\bar{x}(1);\bar{x}(x+1)+\bar{y}(2);\bar{z}(e_1');\bar{y}(y+z)+\bar{u}(3);\bar{u}(u+1)+\bar{v}(e_2');\bar{v}(2\cdot v)$

where

$e_1'=(x+1)(\underline{x}\bar{x}(1)/x)=(x+1)(1/x)=1+1=2$

$e_2'=(u+1)(\underline{u}\bar{u}(3)/u)=(u+1)(3/u)=3+1=4$

So, by axiom 12:

$\underline{x}(P_1+P_2+P_3+P_4)=\underline{x}(\bar{x}(1);\bar{x}(x+1))=(x+1)(1/x)=2$

$\underline{y}(P_1+P_2+P_3+P_4)=\underline{y}(\bar{y}(2);\bar{z}(2);\bar{y}(y+z))=2+2=4$

$\underline{z}(P_1+P_2+P_3+P_4)=\underline{z}(\bar{y}(2);\bar{z}(2);\bar{y}(y+z))=2$

$\underline{u}(P_1+P_2+P_3+P_4)=\underline{u}(\bar{u}(3);\bar{u}(u+1))=(u+1)(3/u)=3+1=4$

$\underline{v}(P_1+P_2+P_3+P_4)=\underline{v}(\bar{v}(4);\bar{v}(2\cdot v))=2\cdot v(4/u)=8$

It is desirable to have one sender and many receivers. Let c_1, c_2 are channels between the sender and two receivers. The sending attempt is $\bar{c}_1(e)\bar{c}_2(e)$ and the receiving attempts are $\bar{u}(\underline{c}_1)$ and $\bar{v}(\underline{c}_2)$. We can define $(\bar{c}_1(e)\bar{c}_2(e),\bar{u}(\underline{c}_1),\bar{v}(\underline{c}_2))$ to be a matched triple if both $(\bar{c}_1(e),\bar{v}(\underline{c}_2))$ and $(\bar{c}_2(e),\bar{v}(\underline{c}_2))$ are matched pairs. Axioms on matched triples (or n-triples) for any $n(n\geqslant 3)$ can be easily derived from axiom 13 and axiom 14. We are not going any further here in this paper.

8 Conclusion and Further Work

Main contributions that this paper has made include:

(1)For the first time assignments are expressed as operations on physical objects, and as such, a complete program becomes an operation expression.

(2)For the first time the final value of variable x (and any other variable) after program P is formally

and mathematically given by $\underline{x}\bar{x}(P)$, instead of x'. And as such, semantics of programs is formally given by axioms, which ensure us to relate the final value of every free variable with initial values of free variables. This can be done by symbolic computation that requires neither the invention of predicates that relate final values of variables with their initial values[1], nor the invention of predicates that relate final values of variables with each other[2-3].

(3)It has been made possible to verify program properties by symbolic computations based merely on axioms and the program itself (i.e. operation expression) as explained by examples in this paper.

(4)A new kind of programming languages consisting of operations on physical objects and operators for control mechanisms is now expectable. As a twin product, program verifiers are also due.

We have, in the previous 7 sections, defined several kinds of operation expressions together with axioms on them. It is possible to give a formal definition of operation expressions in terms of Backus Normal Form. We leave it for later time as part of our future work when it gets enriched and mature to become a programming language.

It has taken the author about 10 years to get this work done in the form given here. The author believes that it is not only a significant step forward towards a unified theory of programming, but also a practical step forward in program verification. Our further work also includes the study of formal program verification mechanism.

Acknowledgement:

The National Science Foundation has supported my research for many years. The author is also grateful to the National Research Center for Software Engineering and School of Electronics Engineering and Computer Science of Peking University.

Prof. YANG Fuqing has given the author personal support ever since the author started working in Peking University. Thanks from the author's heart belong to her.

Many thanks go to Dr. HUANG Yu and ZHAO Wen, XU Chunxiang for their helps.

References:

[1] Hoare C A R, He Jifeng. Unifying theories of programming[M]. [S.l.]: Prentice-Hall, 1998.

[2] Mani Chandy K, Misra J. Parallel program design—A foundation[M]. [S.l.]: Addison-Wesley Publishing Company, 1988.

[3] Mani Chandy K, Charpentier M. An experiment in program composition and proof[J]. Formal Methods in System Design, 2002,20(1):7-21.

[4] Hoare C A R. Communicating sequential processes[M]. [S.l.]: Prentice Hall, 1985.

[5] Milner R. Communication and concurrency[M]. [S.l.]: Prentice Hall, 1989.

YUAN Chongyi was born in 1941. He is a professor and doctoral supervisor at School of Electronics Engineering and Computer Science, Peking University. His research interests include Petri-net theory, parallel program design and program theory.

袁崇义(1941-),男,北京大学信息科学技术学院教授,主要研究领域为 Petri 网,并行程序设计,程序理论。