TECHNICAL REPORT SERIES

P. Randell
and
J. J. Hocking

by

PROCEEDINGS STROUDSTONIAN

Abstract:

The present paper provides a framework for the development of a variety of hardware and software tools and techniques for structuring them. It uses the concept of structured design and extended discussion of processes and dataflow diagrams. The present paper, therefore, is a step towards the development of techniques for structuring systems, and as a basis for understanding the complex computer systems, and as a tool for enhancing the understanding of the general concept of a "process" has become increasingly relevant.
Abstract

The concept of a "process" has become increasingly important both as a tool for organizing and understanding complex computer systems, and as a basis for designing their hardware and software tools.

Extended discussion of processes and techniques for structuring them.

The present paper provides a framework for the definition and description of the concept of a "process".

Suggested Keywords

+ Sequential Processes
+ Program
+ Interpreter
+ Hierarchical Structures
+ Co-Routine
+ Cooperating Processes

Library of Congress:

Suggested class numbers (primary classification underlined):

G.31


University of Newcastle upon Tyne, Computing Laboratory.

Randell, Brian

1972

The concept of process has been in standardization for many years. It is known as a "transaction" in computer science.

Although they have not always used that name (3, 12, 29, 33) have successively used "processes" as the basic unit in the design of computer systems. In these systems, a simple decomposition process may be executed as a single operation or a sequence of operations.

In the decomposition of a program into a sequence of operations, the definition of a program is not decomposed, but as a whole, it is decomposed into smaller units.

For example, the decomposition of a program into a sequence of operations is a task in the standard unit for decomposing programs. In fact, the problems of GERTS and other languages have been experienced in their design and implementation. However, the decomposition of large-scale computer systems is a complex task.
Process "runs":

but also appears to the concept of a degree in which
the definition given by Salzer [29] is much more detailed.

As the procedure is executed by a processor,
which moves through the instructions of a procedure
sequence, what is a process is that abstract entity

"a process is a locus of control within an instruction

the notion of control:"

However, don't and Yan Horin [9] agree that a definition on
executing a program is called a process.

"the sequence of actions that a clerk performs in
the basic multi-programming unit under the control
the standpoint of the control program; therefore,
the unit of work for the central processor unit from
the definition given in [27] for tasks is:

It is the essence of the task concept of OS/360, even through
"carryer and it will execute actions."

General a process has two aspects: it is a data
components, which are separated descriptors, in
and for decomposing a discrete event system into
in parallel, the process concept is introduced as
in programs, called processes, cooperate to accomplish
algorithm to introduce the notion of a collection of
the structure of the data themselves. Similarly extrudes
operations on data local to the program, as well as
operations on data remote to the block, as it oper-ates on a
sequence of

"in A block program (block/sequel) a sequence of
and as a "process" in Shima [8]:"
on the design of the "MPC" Multiprogramming System. In this work
the concept of a sequential process is treated effectively. From this point of view, one must distinguish between
quickly the machine has been suggested to an abstracted machine
"sequential or actions that are performed when an input
... a 'sequential process' can be considered as a

mechanism and then an informal one:
Habermann's [18] gives a formal definition in terms of finite state
machines and then an informal one:

"... sequential process is the input/output of an object document
... this reflects the state in a given store after an

... sequential process is the execution of one or more

with the process concept,
It is clear that interactions with the external world can be mimicked
by the processes definition [2]. This is not possible, but does make

"processes are instances of
"processes are instances of

the essential characteristics of a process is that it

precease by Lamppon [23].

This definition was elaborated, although not made any more

and identity a process with the address space.

that, be connected to make use of this correspondence
also between processes and address spaces. It will, in
one correspondence between processes and state-variables, and
is accessible to the processor. There is then, a one-to-
the internal tangible evidence of a process is a pseudo-
"a process is in execution by a pseudo-processor,

..."
The problem of structuring a complex system and of a set of processes.

Methods of structuring a complex system are not a set of processes. The concept of structuring a complex system or a sequence of processes is called a sequential process. A sequential process is a set of instructions that are executed in a specific order. The concept of a sequential process is important in computer science, particularly in the context of programming and algorithm design.
literature, this paper builds its framework up from a set of quite basic definitions, contained in Section 2. The significance of many of these definitions will become apparent only in the later sections, where we introduce formalism only where it seemed to increase clarity and assist understanding. We have restricted ourselves to a few rather basic mathematical concepts, such as sets, sequences, relations, and functions. Examples illustrate both the concepts defined and the notation used for them.
concepts of state variables, state variable sets and states.

Our definitions are based on the well-known [1, 22] more detailed process of presenting our formal definition of "processes". We will discuss each of those concepts in a sequence of states, or to the action function, which is a function of states, or to a computation, which is a function of states, or to a process, or to a computation function, which is a function of states.

helpful to use different names for each. Thus we will refer to the state of a process, or to the state of a computation, which is a function of states, or to a process, or to the state of a computation, which is a function of states.

one of those processes, a complete basis for the definition of some concept of state variables, or as a means of modeling some sequence of stream values, or as a means of modeling some sequence of stream values.

is used to model the states of a system, or to the action function, or to a process, or to the state of a computation, which is a function of states, or to a process, or to the state of a computation, which is a function of states.

The quotients in the previous section have var"ou"sly.
State variables are elementary quantities which can assume certain well-defined values. A named set of state variables constitutes a state variable set. An assignment of values to all the variables in a state variable set defines a state of that set; conversely, a state defines a value for each state variable. The set of possible states for a given state variable set is the state space of that set.

**EXAMPLE:** Consider the state variable set $V = \{x, y\}$ consisting of two variables $x$ and $y$, whose values may be any positive integers. If $x$ is assigned the value 2 and $y$ the value 4, this defines the state $\langle x, y \rangle = \langle 2, 4 \rangle$. When context makes the respective names of the state variable set clear, the state space of this state variable set is the set $S(V) = \{(x, y) | x, y \geq 0\}$, when context specifies the variable names, its members are states such as $\langle 2, 4 \rangle$, and $\langle 3, 9 \rangle$.

Nowell [1] associates with the DEC DDP-8 processor PC (the 12-bit program counter) the state variable set which contains 12-bit quantities such as AC (the accumulator), which is a state variable set with a finite state space for this set is the set of all possible combinations of values of these variables.

A computation on a state variable set is a sequence of states of that set. The first element of the sequence is its initial state, the last (if it is finite), its final state. A previous state on the sequence is its initial state and the final state of the previous state is its initial state.
All computation terminates.

At any point, the action function becomes undefined, the state and action functions to find the immediate successor state; if the state is to be obtained (to find the action function defined for the current state, and then to each successor state, generate a computation from an initial state by applying the action function to the states into actions. We may use an action function to an action function on a state variable set in a mapping.

The immediate successor is \( (2', 2') \)

\[ (2', 2') \rightarrow (2', 2') \]

\[ (2', 2') \rightarrow (2', 2') \]

\[ (2', 2') \rightarrow (2', 2') \]

\[ (2', 2') \rightarrow (2', 2') \]

\[ (2', 2') \rightarrow (2', 2') \]

and so on.

The null action is the empty set (denoted \( \emptyset \)).

The previous state. The null action is the empty set created.

The NULL action is the empty set.

The state is the immediate successor of the action, the state's immediate successor is the action.

The state's immediate successor is the action.

If there are no state variables, the variables of the state are named in the action.

If there are no state variables, the variables of the state are named in the action.

If there are no state variables, the variables of the state are named in the action.

The state's immediate successor is the action.

If there are no state variables, the variables of the state are named in the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.

The state's immediate successor is the action.
EXAMPLE: The computation $g_0$ is generated from the initial state $(2,2)$ by the function $f_1(x,y) = \{(y,4)\}$, whose action function is strictly deterministic. Null actions have no essential effect on computations except to change their length, since we use the number of actions as our indication of "time". Thus, we refer to action functions as single-valued (e.g., $f_1$ in the previous example). Null actions differ only in the presence or absence of null actions as temporal variants. The standard form of a class of temporal variants is the member which generates no null actions; if all members of the class will be called deterministic. The standard form of this action function is $g_1$, where $g_0(2,2) = \{(y,4)\}$ and $g_0$ is undefined elsewhere, and $g_0$ is undefined elsewhere.

EXAMPLE: The computation $g_0$ is generated from the initial state $(2,2)$ by the function $f_1(x,y) = \{(y,4)\}$, whose action function is strictly deterministic. Null actions have no essential effect on computations except to change their length, since we use the number of actions as our indication of "time". Thus, we refer to action functions as single-valued (e.g., $f_1$ in the previous example). Null actions differ only in the presence or absence of null actions as temporal variants. The standard form of a class of temporal variants is the member which generates no null actions; if all members of the class will be called deterministic. The standard form of this action function is $g_1$, where $g_0(2,2) = \{(y,4)\}$ and $g_0$ is undefined elsewhere, and $g_0$ is undefined elsewhere, and $g_0$ is undefined elsewhere.

When an action function is multiple-valued, it would be inappropriate to call it an action function, but this distinction does not seem particularly helpful.
function. Counterintuitively, in the context of our problem, the action function is defined by the successor function, which must be strongly determined. However, this is not always the case. The action function \( A(x) \) is defined by the successor function, \( \{ (n', n) | (n, n) \in R \} \), where \( n' \) is the successor of \( n \).

EXAMPLE: The process \( A = \{ (1, 1), (2, 2) \} \) is strongly determined.

Counterintuitively, it is the action function we use to define the set of processes.

EXAMPLE: The process \( A = \{ (1, 1), (2, 2) \} \) is not strongly determined.

At least one of the generated processes has a set of states which is the same as another generated process. However, the action function of the generated processes has a set of states which is different from another generated process. The action function is defined by the successor function, \( \{ (n', n) | (n, n) \in R \} \), where \( n' \) is the successor of \( n \).

The action function is defined by the successor function, \( \{ (n', n) | (n, n) \in R \} \), where \( n' \) is the successor of \( n \).
processes and the set of computations generated by the processes.

We use a process to model a process by asserting a

relation between the set of possible computations of the

processes (or actions) occur. Thus assertion is generally valid

for all processes, but not for many "natural" processes.

For "natural" processes, this assumption is generally valid

effect of time is denoted by modular arithmetic. By the order in which

interpretation of discrete moments of time, and that the

"interpretation" behavior of the process is determined through the

two dimensions. Yet we wish to relate processes to processes.

A process is a mathematical object, as an abstract,

initial state is a computation of the process. Each successive state (or at what means, the device represents

instance of time) and by what sequence of states following from an

interpretation of the physical state, which initializes and I am

which can be placed in the ordered initial states, and I am

"where is a physical device?" (we leave this term undefined)

relation to that of "process" a part (D1).

We now turn to the concept of "process" and its

and G is undefined elsewhere.

\[ g(0, 2) = g(2, 0) \]

where \[ g \] is the successor function \( g \) corresponds to the

**Example:** The action function \( g \) corresponds to the

successor function. Two functions in Section 4 are more naturally stated in terms of

definitions in Section 4 are more naturally stated in terms of

we generally find action function the more convenient, but some

successor(s). The two functions can thus be used interchangeably.
EXAMPLE: The process $P_4$ of a preceding example is an exact specification for the process $P_4' = (P_4, I_4)$ schematically represented in Figure 1, where the rectangles denote registers which are initially set to the same value and then at discrete times the value that is interpreted as $y$ is replaced by the output of the multiplier.

Figure 1. Schematic representation of the processor $P_4'$. 

15.
Any problem that can be expressed as an array of operations (and possible conditions) that can be executed in parallel, can be thought of as a high-speed serial or parallel problem. The CDC 6600 computer can be thought of as a high-speed serial or parallel system.

EXAMPLE: The CDC 6600 computer can be thought of as a high-speed serial or parallel system. Any problem that can be expressed as an array of operations (and possible conditions) that can be executed in parallel, can be thought of as a high-speed serial or parallel problem. The CDC 6600 computer can be thought of as a high-speed serial or parallel system.
Given a collection of processes that have no state variables in common, then the actions of one process can have no effect on any of the others.

Thus an action of the common system may be composed of actions of any number of component processes. Our definition of combination must be adjustable at times when the actions of the components (some of which may be shared with other processes) are independent of each other.

In the most general case, each processor will act on the state dependent on the relations among the processes, which they model. The rules governing the combination of processes, of course, are extremely complex from sets of simple processes whose action functions are extremely complex from sets of simple processes whose action functions.

Combination allows us to build processes whose actions can be described as recursively defined. We consider processes which can be described as combination of component processes, each modeled by its own process. In this section, it is frequent to view it as a collection of more or less independent processes.

Given a collection of processes that have no state variables, and whose initial state set is the direct state variables' set, the action of the component process is determined by the common state variables. More formally, such a distinct combination is the process whose

3. Types of Combination

COMBINATION OF PROCESSES
\[
\{(1',0')\} = \{s'\}
\]

and

\[
\{(A',z',x')\} \rightarrow \{(1+x',x')\}
\]

\[
\Rightarrow \{(A',z',x')\} = \{(1+x',x')\}
\]

\[
{0,x' \times x_0 | (A'x)} = \{s'\} \tag{1}\]

\[
\Rightarrow \frac{g}{d} + \frac{g}{d} = \frac{g}{d}
\]

Then the disjoint combination \[\{(1')\} = \{s'\}\]

\[
\{(A',z',x')\} = \{(A)\} \rightarrow \{0<x'|x\} = \{s\} \tag{1}\]

Also, let \[\frac{g}{d} \leq \frac{g}{d}\]

\[
\{(1+x',x')\} = (x) \rightarrow \{0<x'|x\} = \{s\}
\]

where

\[
\{(\theta',\theta',\theta')\} = \{\theta'\}, \theta = \frac{g}{d}
\]

**Example:** Let \[\frac{g}{d} \leq \frac{g}{d}\]
On the other hand, given a collection of processes with the same state variable sets, we first extend each of them to the union of their state variable sets, and the changes to the extended process do not change variables that were not in their original variables. Adjusting the initial state set to contain, for each original state, new states having all possible combinations of values for the new state variables.
Consider the synchronous combination
\[(\Lambda x)(\Lambda y). (x+y)\]
and the action function
\[p. s. q = \tau\]
where \(\Lambda x\) = \(\Lambda y\)
and \(\Lambda x\) is defined by
\[p. s. q = \tau\]

Example: Consider the definition of combination, more General, forms of combination.

Section 3.2 that is used in representation, other, apparently, this is still a family of representation systems of combination, and component-process action are de-terminated by the actions of the synchronous combination of \("\parallel\)\) because the actions from each active component-process. We call this form of combination \("\parallel\)\) or \("\parallel\)\) because the actions state variable. A group of processes with the same state are not strictly parallel, and \("\parallel\)\) means of common processes that overlap in both "time" and "space", i.e., that is, \("\parallel\)\) that overlap in both "time" and "space". Another interesting special case of combination involves.

3.4. We will discuss this topic extensively in section 3.4. We will discuss this topic extensively in section 3.4. We will discuss this topic extensively in section 3.4. We will discuss this topic extensively in section 3.4.
of the component processes, provided that these actions are
are unions of actions from the action functions of any subset
state of the combination consists of all the actions that
of their initial state sets. The action function for each
process has an initial state set that is the intersection
More formally, the general combination of a set of
actions between any two actions of the given process.
Many processes exhibit some level of abstraction; many
processes are formed by merging any pair of actions of the other component
processes. Any component process may perform actions that may have
the actions of any one or more of the component processes.
action which follows an applicable state may consist of
action that follows any applicable state. The
the process represents the combination of
model such systems by general asynchronous combination of
Any system, but is unknown to external observers.
primitively, but is unknown to external observers.
in any arbitrary order of different components. In others, an ordering exists in
systems where there may be no strict ordering between the actions.
In many practical
In serial or synchronous combination of processes,
free combinations in parallel,
In any arbitrary order, the above processes are not conflict-free (that is, to have no conflicts
combination is not well-defined. The meaning of the
free combinations in parallel, the
free combinations in parallel. If such a combination
function of each action is ambiguous.
state transition values and union of one value from the action
of their initial state sets. The action function for any
process has an initial state set which is the intersection
More formally, the synchronous combination of a set of
allow actions by other processes between the load and store, then the process (but not the computer) would thus be modeled by the three-action sequence (load, add, instruction) which operates in a single memory cycle. If many computers have an intermediate add to memory example:

are modeled while the operations in the multiplication sequence in which the operands of the multiplication instruction are modeled by a single action, the multiplication instruction is executed with the execution of other instructions. As mean of accessed address and subtracting whatever may be performed on some computers, multiplication is performed over meaningful interactions which do not occur in the real system. Another interaction which do not occur in the real system on the other hand, if the actions are too small, the computer will model the system not be collected in the combination on the other actions are too large, then some interactions of the real the processes they represent. If, on the one hand, the process must correspond to the "interactions between the actions represented by asynchronous combination in the system." For asynchronous combination, the interactions between an action and the next action are all represented as "represented by asynchronous combination.

\[(x+x')A\] or \[\{(x+x')A\} + \{(x+x')A\} = (x+x')A\]

The general combination \[P\] is the next action interaction between two actions represented by asynchronous combinations.
of a process can be determined relative to the “environment.”

rather, it suffices to provide means by which the process
necessitates to introduce the concept of an “absolute” clock?
current state, i.e., renders it inactive. It is still not
successful state and when false “holds” the process in its
which, when true, “permits” the process to proceed to the
completing predicate (depending only on state variables)
which represents them. This may take the form of an
implies some means of control external to the processes
are sometimes active (changing state) and sometimes inactive
clock by which to measure time? the notion that processes
represent by the number of actions which have occurred.
“cocked” combination of processes, there is an auxiliary
When one is considering an isolated processor or a

systems by introducing a new form of process definition.
attention to synchronous combination and model such
with different speeds. However, we may restrict our
would be necessary to model systems interaction processes
the notion of rates of processes; however, implies some
reached by combination of processes, that is adequate.

In general, however, not all processes in a system
machines, or the “parallel” operation of many computers.
the “parallel” operation of many pulse-bound computers.
combination is adequate to model many useful systems, e.g.

Concurrency and Synchrony

3.2 Concurrency and Synchrony

3.2.1 Concurrency and Synchrony

3.2.2 Concurrency and Synchrony

3.2.3 Concurrency and Synchrony

3.2.4 Concurrency and Synchrony
example. In the synchronous combination of
of the serial combination $P_0 = P_0 + P_0 + P_0 + P_0 + P_0$
the effect

ensuring that precisely one of them is active in any
process $P_0$'s clocks $P_0$'s and process $P_0$'s clocks $P_0$'s

exemplifies the mutual exclusion of processes within a
process. Any number of processes may be controlled by a
single checking process. The activity of the checking
process, which belongs to the environment, is controlled
by a variable on which the enabling predicate depends. The
checking process may be combined with a process with
which it may be combined. Since each checking extension
of a process is a process,

\[ z = (z, x') \]

is checked by $\sigma_0(x)$ with the action function $f$ and other
variables $\theta$ and $\theta'$. Similarly, the process $P_0$ with
\[ \{(1-z', x'), (x', x') \} = (z, x') \]

the extension function $f$ and the predicate $\sigma_0$ has
\[ \{ (1-z', x'), (x', x') \} = (z, x') \]

the checking process with the action

example: Consider the process $P_0$ with the action

We define the checked extension of a process $P_0$ an enabling
predicate as follows:
In any action following that state, it is changed by the immediate change of a process in a state. If it is continued, concepts of cambio and significant variables, a variable is thus defined. These definitions are facilitated by introducing the notions of "input" and "output." This leads naturally to precise definitions for the intuitive changes which the environment makes to the state variable. If the behavior will be influenced (perhaps strongly) by the environment, consistency of the remaining processes. In general, each operates on the state variables in an context. Processes which have been combined may be regarded as.

3.3 Context-free Processes

In which actions occur, and the relations among them, which represent them. Rather, we are concerned with the order of events and the structure of the processes. The discussion of physical speeds of processes does not seem to be needed for understanding the structure of the processes. While such a relation is obviously necessary to the study of a process to the existence of an external time.

The above discussion has not referred the processes or tasks of several tasks of programs.

Example: Such multiprogramming monitor is a software instruction stream.

Improvement of a checking process controlling the checking processes which satisfy them. Example: Early example of multi-programming used for checking procedures. This provides a simple checking process controlling the operating system. In the H800 computer, the computer used.
Both input and output variables of both processes are

\[ x \text{ and } z \] 

In the computation of \( P_{1} \) and \( P_{2} \), where \( x \) and \( z \) are

\[ \text{Example:} \]

They are reset by further outputs, etc.

Action, that no outputs are "lost" by not being used before

"seen" by an output action before it is "used" by an input

Connections which assure that each input-output variable is

of these actions for communication will depend on prior

Activity is one which depends on an input variable. The input

of action is one which induces an output variable and an input

communicate with each other or control each other. An output

may represent the only mean by which coexisting processes may

represent another process are called input-output variables, and

functions of the environment. Collectively, the input and output

variables of the environment, symbolsically, the output

variables are those of the environment which are significant

variables of the environment. Symbolsically, the input

in a given environment, the input variables of a process

\((0')z) = \left( z'\omega'x \right) \omega \left( x'z \right) \omega' \left( x'z \right) \omega \left( x'z \right)

\text{Example:} \]

or more states.

of a process, the state of the environment which are immediately significant in one

The state function for that state, the significant variables of a modification of the value that results in a change in the value

is immediately significant to a process in a state if there is

process is in the immediately changed in any state. A variable
they represent any required amount of cooperation while cooperation on processes (and thus, implicitly, on the processes or non-cooperatively. The topic of this section is the choice of desired cooperation includes those which are unimplemented cooperative interaction includes all interactions which are unimplemented and combined processes (and the processes they represent). There are two important categories of interactions among

3.4 Process Interaction

Interactions with the system's users.

Rathers, the input-output variables correspond to matrix and their values are no longer represented by input-output variables. However, if we consider the combined process represented by input-output variables of their associated which transform information between these units are which transform information between these units are another process modelling a disk memory unit, the matrix which is modelled by a process whose environment contains another process modelling a printer as a process

**Example:** Consider a computer manufacturing as a process

Presumably, uses its outputs, which provides its inputs and processes both studied, which provides the input and output world as a process combined with the processes for any given combined with the external world, and we may model the external processes section exist in isolation. They are immediately

If z is the scope of x and x consists of both processes.

**Example:** In the previous combination, x is local to to that process.

This scope consists of a single process, the variable is local.

It is either a channel variable or a significant variable. If the scope of a state variable is the set of processes for which

Least one destination process (potentially) changes its value at least one source process (potentially) uses this value.
the destruction process has received it. That the source process does not over-write a message before transferring messages between processes for RECEIVING to denote destination input-output operations. For cooperation among the cooperating processes, we shall discuss more sophisticated techniques for sequence of successful messages. We will discuss a more sophisticated technique for communication and control.

Simultaneous updates need messages. Multiple receptions of messages, and controlling unacceptable interactions such as over-artifact messages, double output interactions, respectively. However, the unmediated use of input-output actions, as well as communication "I/O operations" in a computer, as well as communication transfer between processes. Note that the input-output actions of the computation processes occur only through input information, by definition, do not occur on such computations, although sometimes they are need in synchronous systems. The general techniques include the number of actions which have occurred. Although special techniques are adequate for general computations, although special techniques must also be employed to satisfy process modularity. Rigidity excluding all interference. To be useful, these

...
set are waiting for each other.
other members of that set. The are all processes in the
set of processes as dependent on the further progress of some
A situation where the further progress of each member of a

* In only slightly more complex situations simple feedback
strategy attenuates
and output operations on the buffer, ensuring that they
and output operation on the buffer, ensuring that they
since there is no conflict in the order of the
the value of the flag changes after each input
since the value of the flag changes after each input
process. (Each message is used precisely once.
process. (Each message is used precisely once.
flag once it can be changed by only one
since at any time it can be changed by only one
operation is assigned to either the source or the
time the flag is assigned to either the source or the
time the flag is assigned to either the source or the
arbiters, there is no conflict on the
arbiters, there is no conflict on the
arbiters, there is no conflict on the
process. (Each message is used precisely once.
process. (Each message is used precisely once.
process. (Each message is used precisely once.

2. There is no conflict on the
2. There is no conflict on the
2. There is no conflict on the
process. (Each message is used precisely once.
process. (Each message is used precisely once.
process. (Each message is used precisely once.

The key message may be sent at any
The key message may be sent at any
The key message may be sent at any

membrane, if any. The new message can consist of:
membrane, if any. The new message can consist of:
membrane, if any. The new message can consist of:

set the
set the
set the

The new message is copied into the buffer.
The new message is copied into the buffer.
The new message is copied into the buffer.

The following actions: Test the flag until the value is
The following actions: Test the flag until the value is
The following actions: Test the flag until the value is

1. In the context of the conversation, the SEND operation can consist of
1. In the context of the conversation, the SEND operation can consist of
1. In the context of the conversation, the SEND operation can consist of

and a one-bit flag variable, which indicates the status
and a one-bit flag variable, which indicates the status
and a one-bit flag variable, which indicates the status

that may be implemented by means of two shared variables:
that may be implemented by means of two shared variables:
that may be implemented by means of two shared variables:

**Example:** A one-way message stream between two processes
execution of operations involving sequences of actions. The
action which actually triggers the hardware to achieve the mul-
tiple operation actions on most processors. It is necessary to use
interleaving to ensure that operations such as SEND and RECEIVE do not correspond to single
operations. Interleaving is sufficient to ensure mutual exclusion. However,
when all such operations are single actions, simple

By delaying all but the highest-priority request
memory hardware to release potential conflicts
memory unit, special priority logic is required to
the CPU and I/O channels (have access to a common
memory) from many competing systems residing on

EXAMPLE: In many computing systems residing on
when necessary, to ensure mutual exclusion of actions.
Computing systems use various forms of interlocking, correspond-
ting to any point at which one of them can be in progress. A vital
trigger (the bus) is necessary for certain sorts of operations (control
and general coordination). The example above describes the
case in which the message has gone to both processes,
interruption, the message fails to be delivered or delivered
reset by the bus. In one, contrary to the stated
and through the bus to one of many. Fortunately, the priority level
zero, the message out of the buffer,
the first tests the flag, finds it zero, and copies the
following interleaving of actions from the two processes:

EXAMPLE: Suppose that both destination processors
some interference as simultaneous RCEIVER operations.
Several actions, interleaving these actions can produce the
occur simultaneously. If these operations consist of
the RCEIVER operation consists of
if the processes are separately compiled, so that no two actions

By judicious use of P and V operations, without having to alternate on semaphores, processes can be made to cooperate. Operation, enqueues it to proceed. No other operations are allowed on semaphores. Thus the use of a P operation can cause a process to become inactive, and a V operation cannot cause a process to become active. This use of the semaphore value would be non-repetitive. Thus the use of the operation P(S) decrements the value of S by 1, as soon as the operation V(S) increments the value of the semaphore S by 1. These are the P and V operations, which are the heart of a process now introduced by primitives for synchronization.

Synchronization is an inherent part of the operation followed by writing in an interruptible operation. The reference to the memory unit, so that it allows readers an extension of the checking process, which requires control. An extension of a very simple loop, this instruction is based on exploits the memory interrupt to allow synchronization. The nest and setup [36] of some computers, a basic instruction (assume known) in busy waiting, is the cooperative process is released in the cooperative. Some reduction in the complexity (although not in the development of the semaphore with the environment while the purpose is to synchronize with the environment.) However, the latter should be considered to achieve any desired mutual exclusions.

The simple memory interrupt available on virtually all [22], and known by the majority of the actions. It has been shown by Pufister [11] that the use of operations to enforce mutual exclusion of other sequences of operations to use some available mutually exclusive
Reverse direction is needed.

But the "buffer problem", then a second semaphore, protecting in the
more than a certain number ahead of the consumers (the "buffer"
part) is not necessary to ensure that the producer's never get
(see set of consumers) can get ahead of the producer.
If the sets of semaphore to become negative ensures that no consumer
hearing the fact that the p operation will not cause the value
the p operation on the corresponding semaphore.

A process releases a message or resource to the environment,
whose value indicates the number that are available. Whenever
from the environment, it performs a p operation on a semaphore
processes. When a process requests a message or a resource
implementation of "producer-consumer" relationships, one

The second use of semaphores is to facilitate the
operation on processes is not necessarily blocked.

Priority, since a p operation immediately follows each critical
point on a lock, no two critical operations can overlap in progress.
Once a lock is acquired, if p operation can reduce a semaphore
by (mutex) since every lock is preceded by a p, the value of mutex
followed by the critical operation is preceded by p. Whatever
the value of mutex, semaphore, initially, initialized to the value one. Each
primitive is mutual exclusion, and requires a

Different diseases two rather different use for these
effectiveness use of the hardware than does busy waiting.

In addition, in most computer systems they result in more
part of ensuring the correct cooperation of a set of processes,
continued activity of the producer, continued progress, a very important
controlling the lock of the lock, and a "lock" (mutex) to protect processes which mutual
without deadlocks, because busy waiting allocated a
result from the cooperative (simple) which is related to the (still)

Performance busy waiting, which is related to the (still)
mutual exclusion

A critical section must not be entered by more than one process at a time. If a process attempts to enter a critical section, it must first test whether the section is already in use. If it is, the process must wait until the section becomes free. If the section is not currently in use, the process can enter it.

EXAMPE: If a process performs an exclusive (mutex)
operation, and cannot recover if it does not.
cannot ensure that a process will complete a critical operation. Any critical operation, i.e., the environment
performs a (mutex) but before performing the corresponding
operation, gets into an infinite loop after

each of the components

system synchronization and depends on the correctness of
their use does not guarantee correctness. The integrity of a
algorithm depends on the demonstration of correctness,
and although they facilitate the demonstration of correctness,
Sequential operations - for example, be used to

execute the instruction for the construction of a

intermediate state, and the processor processes which to
be subsequently a sequence involving what the term "processor processes" shown by Habermann [7]. However, this [14] has recently

been done using P and as the base, as has been

A very basic set of values of sequential operations can be built up, starting from

ordered sets of sequential operations will the appropriate, one within the
derivation of sequential operation, and to the security requirements of

In different circumstances quite different choices as to the
decided for the general case of arbitrary processes becomes critical.

Within the problem of ensuring that they use in appropriate

virtue of the previous with which their case's existence

will seem spectacular and restrictive in situations when, by

primitives, then the concurrently generated P and Q operations

There is another correlation on a single past set of sequential

join between two other processes.

System ensures that no process interacts with a concurrent

for certain types of errors. For example, by message

operations and recovery procedures can be provided concurrently

their structure is independent of the compiler acitivity in the

variables with each other, but only with the processor processes

communication processes are implemented since they do not share

processes, and handed over to a coordinating process. The

information between a set of processes is removed from the

The message passing system used by Brinch Hansen [3], and the

The co-routine concept introduced by Conway [7], and the

reasoning for the coordination of the program-communication operations for facilitated

...
Synchrophasorization schemes is referred to briefly in section 4.5, however, the topic of synchronization is beyond the scope of this paper. Constructive proposals for synchrophasorization factitious use beyond and presented in the various presentations. A similar scheme has been used by Zucher.
and partly software (the interrupt handler routines) operating process
(co-processors) and partly hardware (the interrupt system).

Many computers are an example of the implementation of
central processor registers. The implementation of
must share more information, for example the contents of
computers with additional facilities such as information
instructions. However, the clocking process of
instruction storage may consist of little more than the
program stored in some memory, or in some cases
representing the executive program (monitor).
The clocking process and the communication of basic

concurrently.

co-processors and the various instructions to proceed.
phases of the clocking process may allow.

The clocking process may also be
more sophisticated (e.g., a computer
execution of several processes in parallel in a single run).
The computer architects the clocking process ensures that the processes

Any system for the coordination of the basic processes. On many

associated sequentially. A computer is a clocking process.

The notion of process communication produces an
interaction that is the specification of an algorithm as a
sequence of elementary actions, chosen from the instruction set of a computer. Each instruction can be regarded as the
interaction on the nature of conventional sequential programs.
When more than one program is used, the program switching times, the process's status is central to the operation of the program associated with it. The act of bearing its status must include restoring its status; in the MULTICS system, on the idea of a program status, including address mapping information. This leads to a fixed one-to-one association of address spaces and processes, a somewhat restrictive but still useful definition. In multiprocessor systems this associates a process with a particular processor and many other processors are simultaneously executing that activity, how often its execution is interrupted, or how many processors are involved in various stages of its performance.
the computation defined by the internal states of the
Program is a computation which is very much shorter than
the output of a program (for instance, a trace)

\textbf{Example:}

observable state of a computation
from the application of the interpretation in turn to each
image computation is the sequence of image states resulting

definition: states for which the interpretation is not defined
the image state (under this interpretation) when it is.
The result of applying an interpretation to a state is

mathematically ones.
term involved a mapping from physical quantities into
are purely mathematical, whereas our programs use of the
map states from one space to another. These interpretations
of interpretation to induce (single-valued) functions which

computations of a processor. We will now extend the concept
apply to the behaviour of a device to abstractly
We have already discussed how an interpretation may be

4.1 Interpretations and Images

4. ABSTRACTION AND REFERENCE
Later, we'll discuss means for interpreting such images of processes. 

We'll also devote most of our attention to those computations that are very important subclasses of process images, to which we haven't yet looked. We'll have not explicitly dealt with such.

Understanding deterministic processes in terms of (at least conceptually) difficult forms—in terms of systems whose stable behavior often takes such processes—is the way to study computer images. Most of the world models do not exhibit the desirable images. The problem for studying such a large class of internal states will be an important question further afield of the main problem of this chapter, i.e., generally.

EXAMPLE: From the output of a program, we can usually predict whether it is correct or not: the underlying or the image of the computation.

Information needed to predict further states of other computations is needed to predict further states of other computations. However, since both are partially mathematical, interpretation does not ensure that the image can be interpreted strictly deterministically, our rather general form of

Because when a process is simple and well-behaved (e.g.,

processes, such as assertions, are more amenable to formal proof

computations. However, since both are partially mathematical

processes, that is by asserting a relation between their

images in the same way that we associate a process with a

computations. We may associate a new process with such an

image of a set of computations as the set of images.
EXAMPLE: Complex programs may more easily be understood
by

Generating simple processes

At the highest level of abstraction these abstractions can only yield complex processes. Of course, these abstractions can only yield processes such as those provided by high-level abstract languages. A process in this language is not a description of what a machine can contain, but rather a description of the information that the machine can contain in general. In general, it will contain information about the program. By the very nature of an abstraction, an abstraction can contain no more information than the program which it is an abstraction of.

The proof of a theorem is the execution of the process which is an abstraction of a computation. The output of a trace program is a computation of the form:

\[ (0^z, 2, x) = (z^x, x) \]

The action function translates the trace into a computation, i.e., the trace process or map into single actions, i.e., the machine process of interpretation leads naturally to the theory of "belief."
Computation is the sequence of steps and the image of the successor function, \( f^x \), under the interpretation \( I \). Let \( f^x \) be the successor function of \( f \) and denote it as \( f^x \). The image of \( f^x \) under \( I \) is \( I(f^x) \).

**Example:** Consider the process \( p \) with the successor function \( f^x \). The complete process \( p \) can be mapped into the image space by applying the interpretation \( I \) to the successor function \( f^x \). The image space is obtained by mapping the process from the set of states into the image space.

Any process which is isomorphic to all processes can be answered by studying the corresponding state of computation of the other. This question about a process can be answered by studying the corresponding state of computation of the other.

The states of a process are the state variables whose state is determined (the observable state variables) and the state variables whose values are unknown (the unobservable state variables). In this way, the state variables are partitioned into two sets: the observable state variables and the unobservable state variables. The observable state variables are those whose values are known, and the unobservable state variables are those whose values are unknown.

Arbitrary interpretations may be constructed by applying the interpretation \( I \) to the process.
of certain classes of computability.

In general, the image of a process under a projection function is not a process, since a projection does not preserve the state. Namely, let $\mathcal{L}$ be the sequence $\langle 0, 0, 1, 1, 0, 1, 0, 0, 1 \rangle$, and let $p$ be the projection function. Then $p(\mathcal{L}) = (0, 0, 1, 0, 0, 1, 0)$. However, we can derive a process which is a true inverse. Moreover, we can determine the immediate successors of all states whose image is a given state to determine the strata of computations this image, by using the immediate successors of the state. If this is observable, then the image of the process is observable. If it is not observable, then the image of the process is not observable. If the state selection function is observable at $z = 0$, then the image of the process is observable. If the state selection function is not observable, then the process is reobservable until except that at the immediate successor in the underlying space is derivable by a procedure similar to that for isomorphisms.

**Example:** Recall the process $p$ with $p^5$
The process \( P_2 \) with action function \( \langle I \rangle \) is exactly the \( \langle I \rangle \) of the process \( P_1 \) with action function \( \langle I \rangle \).

Example: Recall the processor \( P \) of the two given interpretations. Composition of the two given interpretations of the same device and a new interpretation which is the image process is realized by a processor which consists of the image under a second interpretation. Now the processor plus an interpretation (and a high-level process which is the processor) is realized by a device low-level process realized by a processor (i.e., by a device). Consider two of these interpretations are realized. Consider a variety of processes under different interpretations. We now present a special case in which realizations of processes may not concern with them.

From users who are not concerned with the interpretation of/\(<I>\) operations or interfaces (e.g., the process order of \( P \) operation or interfaces). To hide the low-level interpretation of the hardware and nondeterministic processes we well have the same deterministic abstract interpretation. In fact, deterministic and nondeterministic processes may have the same general abstract interpretations. Thus a deterministic process may have a nondeterministic abstract interpretation (i.e., all the successors of a state may have a nondeterministic abstract interpretation). A state of a deterministic process may have a nondeterministic abstract interpretation (i.e., a state of a deterministic process may have a nondeterministic abstract interpretation). Although nondeterminisms preserve other the determinancy, computations of \( P \) are finite.
understanding characterized as program instructions, as well.
A general-purpose computer is a processor with the full
processes corresponding to the instructions of the program
from the computer's own combination of the basic
program and being represented by a process which resulted
in section 3.5 we described an executing sequential

be more easily understood.

A function of a processor yields another processor, intended to
common thread is that in each case an appropriate interpreter.
The mapping hardware is special, and a variety of

examples are procedural, interrupt-driven systems, and address
interpreters. Our rather broad definition also introduces such

The function example is a conditional use of the term
interpreted by the architecture of the computer.

interpreted by a micro-program, which is in turn
interpreted by a program in machine-language, which is in turn
may not be known or care that this language is interpreted

EXAMPLE: The use of an interactive console language

we do not choose to identify further structure.

The processor level is, in practice, that level below which
into an interpretation and a still lower-level processor.

something else, i.e., we can always decompose a processor
the distinction between an interpreter and a processor is

for explanation of the behavior of the device. We call such an
we have processes on two levels which simultaneously

instead of retaining the separate interpretation.

defines another processor, which may be treated like any other.

Thus, a processor plus an interpretation may be used to

exactly specifies the processor p. \( \Phi _0, \Phi _1 \)
describing the activity of a special-purpose computer. More
specifically, the use of the term "process" in
Section 1.1 is justified by the need of the term "process" to define
the concept of an "executed program" as defined in
Section 1.1. In other words, the use of
the concept of an "executed process" will result in
different processes being defined. In order to make
sets of values for the instruction variables (i.e., different
interpretation are fixed, different programs (i.e., different
and only the variable values as data,
and only the variable values as data,
will prefer to think of the expressions as instructions,
the values of the expressions, a user of the program
completes them, accept values for variables, and print
even if he is writing a program to accept expressions,
EXAMPLE: To the author of a program, this input is data,
that different programs can be accommodated by different
behavior. This partitioning is also arbitrary, in the sense
variations in order to simplify the task of understanding the
even if a program is written to accept expressions,
PARTITIONING
that we choose to regard as constants. A
values of these instruction variables, but only those state
of such a process will generally not explicitly reflect the
interpretation. Our interpretation of the program
choices for the values of the state variables that represent
arbitrary complex sequences of operations, by constructing
simple checking process can be used to construct (in principle)
will be performed. Thus, sets of simple basic processes and a
that state variable determines which of the basic processes
the next basic process to be activated, the current value of
process indicates that a particular state variable represents
state variables. There is a set of different types of basic
as program status, are represented by values of the (changeable)
General-purpose computers. We think, however, that this includes much more than the currently conventional class of
Our definition of programmable processor absolutely
practical considerations.
Practical processors are simple as a Universal Turing Machine from
Example: It is, of course, these factors which rule out

(iii) The efficiency of the constructed processor.

(iii) The practicality of constructing or obtaining the

(iii) The practicality of constructing or obtaining the

(iii) The practicality of constructing or obtaining the

The ease with which the process required for the

These include

Given programmable processor for a particular application,

several factors which determine the practical utility of a

In addition to any theoretical requirements, there are

Turing Machine (cf. [7]),

unlike it could be programmed to be equivalent to the

could not be remedied by the claimed to be "general-purpose"

important type of programmable processor is, of course,

Example: The outstanding example of a theoretical

That can be constructed using it.

The theoretical importance of a given programmable

and are not observable at the higher level.

variables of the underlying process are instruction variables

Precisely, we use the term programmable processor to describe
A function on processes is the semantics of another
exiting semantics to define the semantics of another
language, i.e., we give a mapping, called a translator.

A description language is a set of descriptions, each of
which represents a process. The function which maps these
A description language is a set of specifications, each of
necessity to have more easily manipulatable representations.
very inconvenient for human manipulation. It is a practical
voluntary on whose manipulation or cores, etc., are generalized.
The forms which a processor requires for the programs

of the processor.

Instruction variables represent choices from the order code
Instruction variables store state, and the values of the
program execution are now states to the cocking processor where
status variables indicate to the cocking processor where
the values of the program

particular processor is a set of initial values for the program

A program for a

program by means of a program, it becomes convenient to
given a programmable processor, it becomes convenient to

4.3 Description of Processes

not eventually be modified or superseded.
however, is not possible to assume that these conditions will
remained useful in the development of present-day computers.
possibilities to be considered, and thoughtfully introduced
variables still have a description of a general-purpose computer. This
classic description of a general-purpose computer, however, is
unnecessarily any fundamentally useful in the analysis of
and order code "must" take. We do not wish to exclude
traditional regarding the forms which their cocking process
definition identifies the most essential characteristics of
We have already discussed structuring a process as a tape into electrical connections between planes. Plane writing machine translates descriptions on paper, programs from "external" form, a back-end description language into programs. A loader translates from the new description language into the old.

Recall our earlier discussion of a checking process:

Creation of Processes

Single-level description

We will have been with one type of computer, the complexity nearly separate them cleanly as well as we may find the complexity nearly of course, necessary to choose languages carefully, and to fix one of our best tools for mastering complexity. It is appropriate to our conception of the behavior at that level, the freedom to use at each level a description language.

Problems on the Level Below.

Programs from "machine" are each implemented (denoted) by a "console machine", the "machine", and the "meta-".

Example: In our previous example of levels, the description language, processor and the hierarchy of programs expressed in these languages. There is a natural correspondence between levels and description languages. For each such programable processor, there is a particular language for this processor to be a programable processor. Each interpreter defines a processor at its level. It is the hierarchy of interpreters built on an underlying processor.

The (one level) problem is to translate a process into another process. The problem is how to map from old specifications to new specifications, preserving the new.
We find the sequential processes associated with the form of only one conversation at a time. This means the console is translated into a resource representation of console. The fact that they share the same physical address as if each process had its private representation of sequential processes on higher levels. We find the "message interpreter" above level 2 synchronized with respect to the data interpreter and the so-called "segment controller" a sequential process having disappeared from the picture. At level 1 we have the different sequential processes, the actual processor, the number of processors actually shared is no longer allocation to one of the processes. Above level 0 we find the responsibility for processor levels implemented by cooperating processes. Outside has been taken to structure the monitor explicitly into the example: In the "THE" operation system [12] great care was completely independent.

Dynamic combination at the next level whose variations are difficult combination at one level to serve as the interpreter for a number of processes it controls by initializing or discarding system status. Why it is helpful to associate close to the notions of "processes" sets of program status variables. This is a further reason
Combination and Abstraction

4.5

recently, some attention has been devoted to the problem of finding restrictions which ensure that a combination of abstract machines and semaphores, cannot be synchronized by the "outside" values of the semaphores, and an interpretation that "loses" the synchronization by means of p and v operations on

EXAMPLE: Consider a combined group of processes that

20] Unfortunately, thus far not generally true.

is the same as the combination of the separate abstractations to assume that an abstraction of a combination of processes is true, now [12]. It is tempting to use the assumption to establish properties of

satisfied) cooperation (i.e., that all environment assumptions are certain assumptions about the environment) plus a proof of constructed from separate proofs for each process (under example, a correctness proof for an entire system), for realizing the complexity of such combinations. It allows, for

separate correctness was insufficient to guarantee correctness obtained from simple ones by combination, and noted that their

Previously, we introduced how very complex processes could be abstraction is a means of avoiding unwanted complexity,
one at a time. Each time, and allows the digital operations to proceed
and involve the atomic process while "polling" the processes. It is
not based on a lower level mutual exclusion invoked
only techniques for overcoming mutual exclusion among
the processes. It seems that the
recursion must terminate. Of
achieve mutual exclusion on a lower level, the
exclusion of operations on each level depend on
be treated as a recursive problem, with the mutual
sections. Thus, this aspect of the correctness of a
any exclusion of operations consisting of a
the availability of many parallel simple operations within
problem. Therefore, the techniques discussed in section
combination. Recall that the techniques discussed in
exclusion of operations in a System involving asynchronous
there are two basic ways of achieving mutual
which each such operation becomes a single action.
operate a separate process. We can select one of the
operations which constitute the input-output operations of
processes. If we ensure the mutual exclusion of the
process that do not involve input-output to a single action,
within actions each achieve an absence of actions within a single
other processes.) Next, we may study the abstraction
process in the input variables, caused by
deterministic changes in the input variables, caused by
separately, (the process with all)
separate (the process with all)
the state variables
cooperation of a group of combined processes. First, we may
suppose that we wish to establish the correctness and
to progress. This technique can substantially reduce the
to satisfy oneself that the image process will always be able
it is an image. In this particular example, it is then sufficient
can progress, so can each of the set of processes of which
progresses by a single image process, and progress that is
progress, one can represent a group of sequential
In this regard, one can represent a group of sequential
The concept of multi-level processes is very useful
to which inference can be minimized
The redundancy of processes of connotative about systems
unumpactualy return to their home position, except in the event there is no possibility of deadlock, so the processes may
be to their home positions, no uncompleted tasks remain
number of further tasks remain. When all processes have terminated
single task, can lead to the generation of an intuitive process, when the processes in these stages into the processes. The process
"coming position,, ready to accept a new task. The process
complete the task without mental fatigue and return to the initial
is presented with a task, the process, under all circumstances, the
of the system. The aim is to show that whenever a process
is satisfied, the system is a process is the logical correspondence
situation in an intuitive manner, to enable the designer to
A feature of this design methodology is the careful use of
of the "THP Multiconfiguring System": The understanding of
The former paper describes the design and structure of the
concept of levels of abstraction plays a central role in the
to some extent, de Secker, de Mendoza, [35], both of
this basis, for example, when the case in the work of
abstraction cannot be carried over to their environment.
important conclusion to stress is that arguments about
combinatorial combinatorial complex systems have relied on
To date, practical applications of abstraction and
53.

which are complex with respect to the organization, they may to

If there are important systems in the world

or perhaps the proposition should be put the other way
described, and even to see such systems and their parts

neatly decomposable, therefore the structure is a major

"The fact, then, that many complex systems have a

level, each of the primary processes as a sequential process,

number of parts in the system as a sequential process, another

modeling of level represent each of the phenomena very

example, for example, of sequential processes on a given level would be chosen

described in terms of that particular abstraction. The number

degree of abstraction, and those actions of the system best

are not, or by comparison to other levels, at least in part, by comparison to other levels.

and eventually become the actual system. The phenomena

modeling of the simulation program, which were met, and

and revealed severe demands on the understanding that an adequate description

The use of multi-level processes described by cruncher

[21] that the win not have considered any situations. Any

that is not that is being tested will be so promising

number of relevant test cases will be so small that he can try

stroungated - that of each stage of the simulation - i.e., so effectively

implemented. It seems to be the destructor's responsibility

of the proof. This feature also notes that its approach has

number of situations which must be considered at each stage
Our aim in this paper has not been to develop yet another computational formalism. Rather, our efforts have been motivated by a belief in the profound importance of complex computing systems, and of the need for a sound conceptual framework within which to discuss and develop structuring techniques. Two of the essential structuring techniques for processes seem to be combination and abstraction; our conceptual framework permits the consideration of both within a uniform formalism.

Our belief in the importance of structuring is based on its usefulness in mastering complexity. This applies whether one is trying to understand an existing system, or to design a proposed new system. The goal is to profit from this "master" by finding better ways of producing better systems, and, as an almost automatic by-product, better methods of documenting systems.

However, it is important to recognise that structuring in itself is not necessarily beneficial; bad or excessive structuring may be valueless or positively harmful.

**EXAMPLE:** A program which has been divided into too many subroutines may not only be unreadable, but may also execute very inefficiently, particularly on a paging system.

The appropriate use of structure is still a creative task, and is in our opinion a central portion of any system designer's responsibility.
performs performant assessments, every few minutes: Level 4

with the operator, on the scale of seconds. Level 5

with a 40-millisecond resolution time? Level 2 communicates

time scales: Level 1 (performing memory management), using a drum

level, Level 0 (effective processor synthesis). On a 50-microsecond

level, a substantial delay longer than the scale of the underlying

system is that each level is concerned with events that occur

levels of the set of levels of abstraction used in the

HMS system, recently discussed by Dijkstra [17]. He

of abstraction may well be implied by a characteristic of

a useful guideline for the choice of effective levels

and Turbak [33], specific systems, e.g., British Telecom [21], Dijkstra [17],

specific systems, e.g. the British Telecom, in the design of

abstracts at following this guideline in the design of

answer, even though there have been some very successful

component processes is a question which still faces a general

question separate from their design. How to choose the

processes separately and considering the cooperation of

different levels of these

and Dijkstra [21], have suggested

these [21] has suggested

are a very few generally accepted guidelines for

part, will not yield to any reasonable analysis: One cannot make such a

distinction completely arbitrary because an unintentional distinction of a system into

the purpose of analysis, each part is treated in turn as the

separate compartment and also understanding (in the framework)

find distinctions such that we can understand each part

When we cannot grasp a system as a whole, we try to
were all in terms of the order code of the underlying processor.

Description of the processor being much smaller than it is
of description languages will normally result in the total
of a program in some description language. Applicable designer
programmable processor. Each interpreter will be defined by
of a hierarchy of interpreters built on an underlying

Another example is a process which is defined by means
with the result that considerable performance gains
achieved. Parallelism (even appropriate process description)
processes, than the conceptual parallelism might well become
defined as the asynchronous combination of a set of simpler
a single, rather complex, sequential process is instead
also clear that if more than one process have been defined as
and decomposed self-contained or required documentation. It is
are fairly obvious, such as increased ease of modification,
some of the possible benefits of using structured

constitute the computer system.

Physical structure of the hardware and software that
also reflects the system's behavior by the actual
if it is beneficial to mirror the conceptual structure of the

declared parallelities can influence the quality of a system,
there are also more direct ways in which structuring can
construction, then we would expect an improved final product

to help them cope with the complications of the system they are

If a design team has been successful in using structured

operating system [26].

the scale has been used by Lluch to structure the CHIDS
interaction. A similar relationship between levels and
force the incorporation of inter-level scheduling
lower levels - otherwise, unnecessary constraints will
abstraction to strictly enforce the detailed segmentation on
this sort of relationship is essential for a given level of
handles job abstraction (several minuses). It appears that

7
Programmers, portfolio programmers, or users of standard
whom they are, for example, maintenance arguments, system
different means of communication with the system, depending on
class of user of a computer system may have quite
different levels of a certain extent this already happens
in several levels. To a certain extent this already happens
system can benefit greatly if it is carefully structured into
Finally, it is worth mentioning that users of a computer
beyond the scope of this paper.

Reliability is a whole subject in itself — detailed discussion
use of the concept of process structuring for another level of the
machine was functioning correctly. However, the general level of the
programmer’s text (and even less so in necessary
which generated the program text (and even less so in necessary
configuration, just some of which are enforced by the compiler
configuration of the system are not particularly helpful in
relevancy in the correspondence program text is by means of
changes with hardware problems, these are because their physical
information that are merely conceptual and of little assistance.

Substitutions that are merely conceptual and of little assistance
attacked from incongruences are found.

provided, where it should be checked, and what formalism should be
processes can provide guidelines as to what readout should be
a complex process into levels and into groups of computer
information, whose constraints can be checked. The substitutions of
error detection is based on the production of redundant
reliability, in the case of hardware failures that may be present
extracted in this paper concerns the problem of structuring that have been
however, one of the most interesting potential benefits of
speed storage, and a program level in core storage.

Furthermore, the levels of program may be reflected in the choice
application packages. However, as has been pointed out by Bridger [2] in his comments on a paper by Bryant [4], these benefits can be greatly diminished if the separation between levels is not properly maintained. Levels which should be invisible to a particular user have an embarrassing tendency to show up when there is a malfunction—the task of minimizing these occurrences and their effects should be regarded as an important responsibility of a system designer.
ACKNOWLEDGEMENTS

This paper is largely based on an earlier report by the authors. The preparation has been greatly aided by the many useful and stimulating discussions that the authors have had with E.W. Dijkstra, G.C. Fricke, D. Gries, and W. Zimmermann, who have generously commented on earlier drafts of this paper. Their suggestions have been incorporated into the present version. In particular, the authors gratefully acknowledge the suggestions of J.R. Burch, P. Goguen, H. von Neumann, and A.W. Goldstine. Many of the criticisms have been replaced by superior ones due to Lauer.

REFERENCES


