Theory of Processes

A.M.Mironov

Contents

Foreword

This book is based on author's lectures on the theory of processes for students of Faculty of Mathematics and Mechanics and Faculty of Computational Mathematics and Cybernetics of Moscow State University.

The book gives a detailed exposition of basic concepts and results of a theory of processes. The presentation of theoretical concepts and results is accompanied with illustrations of their application to solving various problems of verification of processes. Some of these examples are taken from the books [89] and [92].

Along with well-known results there are presented author's results related to verification of processes with message passing, and there are given examples of an application of these results.

Chapter 1

Introduction

1.1 A subject of theory of processes

Theory of processes is a branch of mathematical theory of systems, which studies mathematical models of behavior of dynamic systems, called **processes**.

Informally, a **process** is a model of a behavior, which performs **actions**. Such actions may be, for example

- reception or transmission of any objects, or
- transformation of these objects.

The main advantages of theory of processes as a mathematical apparatus designed to modeling and analysis of dynamic systems, are as follows.

1. An apparatus of theory of processes is well suited for formal description and analysis of behavior of **distributed dynamic systems**, i.e. such systems, which consist of several interacting components, with the following properties:

   - all these components work in parallel, and
   - interaction of these components occurs by sending signals or messages from one component to other component.

   The most important example of a distributed dynamic systems is a computer system. In this system

   (a) one class of components is determined by a set of computer programs, that are executed in this system,
   (b) other class of components is associated with a hardware platform, on the base of which the computer programs are executed,
   (c) the third class of components represents a set information resources (databases, knowledge bases, electronic libraries, etc.) which are used for the operation of this system
   (d) also it can be taken into account a class of components associated with the human factor.

2. Methods of theory of processes allow to analyse with acceptable complexity models with very large and even infinite sets of states. This is possible due to methodology of symbolic transformation of expressions which are symbolic representation of processes.

   The most important examples of models with an infinite set of states are models of computer programs with variables, domains of which have very large size. In many cases, models of such programs can be analyzed more easily, if domains of some variables in these models are represented as infinite sets. For example, a domain of variables of the type `double` is a finite set of real numbers, but this set is very large, and in many cases it is puprosely to replace this finite domain by an infinite domain of all real numbers In some cases a representation of an analyzed program as a model with an infinite set of states greatly simplifies a reasoning about this program. An analysis of a model of this program with a finite (but very large) set of states with use of methods based on explicit or symbolic representation of a set of states can have very high computational complexity, and in some cases a replacement

   - the problem of an analysing of original finite model
   - on the problem of an analysing of the corresponding infinite model by methods which are based on symbolic transformations of expressions describing this model

   can provide a substantial gain in computational complexity.

3. Methods of theory of processes are well suited for investigation of **hierarchical** systems, i.e. such systems that have a multilevel structure.

Each component of such systems is considered as a subsystem, which, in turn, may consist of several subcomponents. Each of these subcomponents can interact

- with other subcomponents, and
- with higher-level systems.

The main sources of problems and objects of results of the theory of processes are distributed computer systems.

Also the theory of processes can be used for modeling and analysis of behavior of systems of different nature, most important examples of which are **organizational systems**. These systems include

- enterprise performance management systems,
- state organizations,
- system of organization of commertial processes (for example, management system of commercial transactions, auctions, etc.)

The processes relating to behavior of such systems are called **business-processes**.

1.2 Verification of processes

The most important class of problems, whose solution intended theory of processes, is related to the problem of verification of processes.

The problem of **verification** of a process consists of a constructing a formal proof that analyzed process has the required properties.

For many processes this problem is extremely important. For instance, the safe operation of such systems as

- control systems of nuclear power stations,
- medical devices with computer control
- board control systems of aircrafts and spacecrafts
- control system of secret databases
- systems of e-business

is impossible without a satisfactory solution of the problem of verification of correctness and security properties of such systems. A violation of these properties in such systems may lead to significant damage to the economy and the human security.

The exact formulation of the problem of verification consist of the following parts.

1. Construction of a process P, which is a mathematical model of behavior of analyzed system.

2. Representation inspected properties in the form of a mathematical object S (called a **specification**).

3. Construction of a **mathematical proof** of a statement that the process P satisfies the specification S.

1.3 Specification of processes

A **specification** of a process represents a description of properties of this process in the form of some mathematical object.

An example of a specification is the requirement of reliability of data transmission through the unreliable medium. It does not specify how exactly should be provided this ensured reliability.

For example, the following objects can be used as a specification.

1. A logical formula which expresses a requirement for an analysed process.

   For example, such a requirement may be a condition that if the process has received some request, then the process will give response to this request after a specified time.

2. Representation of an analyzed process on a higher level of abstraction.

   This type of specifications can be used in multi-level designing of processes: for every level of designing of a process an implementation of the process at this level can be considered as a specification for implementation of this process at the next level of designing.

3. A reference process, on which it is assumed that this process has a given property.

   In this case, the problem of verification consists of a construction of a proof of equivalence of a reference process and an analysed processes.

In a construction of specifications it should be guided the following principles.

1. A property of a process can be expressed in different specification languages (SL), and

   - on one SL it can be expressed in a simple form, and
   - on another SL it can be expressed in a complex form.

   For example, a specification that describes a relationship between input and output values for a program that computes the decomposition of an integer into prime factors, has

   - a complex form in the language of predicate logic, but
   - a simple form, if this specification is express in the form of a standard program.

   Therefore, for representation of properties of processes in the form of specifications it is important to choose a most appropriate SL, which allows to write this specification in a most clear and simple form.

2. If a property of a process initially was expressed in a natural language, then in translation of this prorerty to a corresponding formal specification it is important to ensure consistency between

   - a natural-language description of this property, and
   - its formal specification,

   because in case of non-compliance of this condition results of verification will not have a sense.

Chapter 2

The concept of a process

2.1 Representation of behavior of dynamic systems in the form of processes

One of possible methods of mathematical modeling of a behavior of dynamic systems is to present a behavior of these systems in the form of **processes**.

A process usually does not take into account all details of a behavior of an analyzed system.

A behavior can be represented by different processes reflecting

- different degrees of abstraction in the model of this behavior, and
- different levels of detailization of actions executable by a system.

If a purpose of constructing of a process for representation of behavior of a system is to check properties of this behavior, then a choice of level of detailization of the system's actions must be dependent on the analyzed properties. The construction of a process for representation of a behavior of an analyzed system should take into account the following principles.

1. A description of the process should not be excessively detailed, because as excessive complexity of this description can cause significant computational problems in formal analysis of this process.
2. A description of the process should not be overly simplistic, it should
    - to reflect those aspects of a behavior of the simulated system, that are relevant to analyzed properties, and

- preserve all those properties of the behavior of this system, that are interesting for analysis

because if this condition does not hold, then an analysis of such a process will not make sense.

2.2 Informal concept of a process and examples of processes

Before formulating a precise definition of a process, we give an informal explanation of a concept of a process, and consider simplest examples of processes.

2.2.1 Informal concept of a process

As it was stated above, we understand a process as a model of a behavior of a dynamic system, on some level of abstraction.

A **process** can be thought as a graph P, whose components have the following sense.

- Nodes of the graph P are called **states** and represent situations (or classes of situations), in which a simulated system can be at different times of its functioning.

  One of the states is selected, it is called an **initial state** of the process P.

- Edges of the graph P have labels. These labels represent **actions**, which may be executed by the simulated system.

- An **execution** of the process P is described by a walking along the edges of the graph P from one state to another. The execution starts from the initial state.

  A label of each edge represents an action of the process, executed during the transition from the state at the beginning of the edge to the state at its end.

2.2.2 An example of a process

As a first example of a process, consider a process representing the simplest model of behavior of a vending machine.

We shall assume that this machine has

- a coin acceptor,
- a button, and
- a tray for output of goods.

When a customer wants to buy a good, he

- drops a coin into the coin acceptor,
- presses the button

and then the good appears in the tray.

Assume that our machine sells chocolates for 1 coin per each.

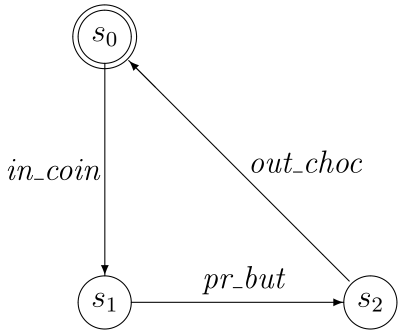
We describe actions of this machine.

- On the initiative of the customer, in the machine may occur the following actions:
    - an input of the coin in the coin acceptor, and
    - a pressing of the button.
- In response, the machine can perform reaction: an output of a chocolate on the tray.

Let us denote the actions by short names:

- an input of a coin we denote by *in_coin*,
- a pressing of the button by *pr_but*, and
- an output of a chocolate by *out_choc*.

Then the process of our vending machine has the following form:



This diagram explains how the vending machine does work:

- at first, the machine is in the state s_0, in this state the machine expects an input of a coin in the coin acceptor
  (the fact that the state s_0 is initial, shown in the diagram by a double circle around the identitifier of this state)

- when a coin appears, the machine goes to the state s_1 and waits for pressing the button

- after pressing the button the machine
  - goes to the state s_2,
  - outputs a chocolate, and
  - returns to the state s_0.

2.2.3 Another example of a process

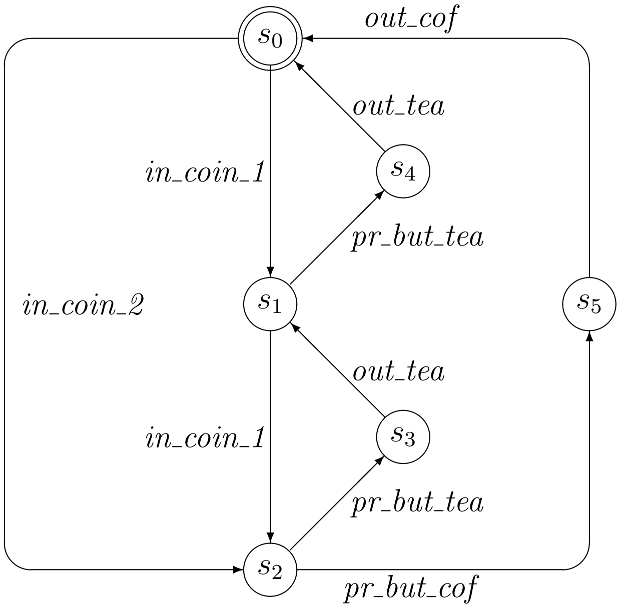
Consider a more complex example of a vending machine, which differs from the previous one that sells two types of goods: tea and coffee, and the cost of tea is 1 ruble, and the cost of coffee is 2 rubles.

The machine has two buttons: one for tea, and another for coffee.

Buyers can pay with coins in denominations of 1 ruble and 2 ruble. These coins will be denoted by the symbols *coin_1* and *coin_2*, respectively.

If a customer dropped in a coin acceptor a coin *coin_1*, then he can only buy a tea. If he dropped a coin *coin_2*, then he can buy a coffee or two of tea. Also it is possible to buy a coffee, dropping in a coin acceptor a couple of coins *coin_1*.

A process of such vending machine has the following form:



For a formal definition of a process we must clarify a concept of an action. This clarification is presented in section 2.3.

2.3 Actions

To define a process P, which is a behavior model of a dynamic system, it must be specified a set $Act(P)$ of **actions**, which can be performed by the process P.

We shall assume that actions of all processes are elements of a certain universal set Act of all possible actions, that can be performed by any process, i.e., for every process P

$$Act(P) \subseteq Act$$

A choice of the set $Act(P)$ of actions of the process P depends on a purpose of a modeling. In different situations, for a representation of a model of an analyzed system in the form of a process it may be choosen different sets of actions.

We shall assume that the set *Act* of actions is subdivided on the following 3 classes.

1. **Input actions**, which are denoted by symbols of the form

   $$\alpha?$$

   The action $\alpha?$ is interpreted as an input of an object with the name α.

2. **Output actions**, which are denoted by symbols of the form

   $$\alpha!$$

   The action $\alpha!$ is interpreted as an output of an object with the name α.

3. **Internal** (or **invisible**) actions, which are denoted by the symbol τ.

   An action of the process P is said to be **internal**, if this action does not related with an interaction of this process with its **environment**, i.e. with processes which are external with respect to the process P, and with which it can interact.

   For example, an internal action can be due to the interaction of components of P.

   In fact, internal actions may be different, but we denote all of them by the same symbol τ. This reflects our desire not to distinguish between all internal actions, because they are not observable outside the process P.

Let *Names* be a set of all names of all objects, which can be used in input or output actions. The set $Names$ is assumed to be infinite.

The set *Act* of all actions, which can be executed by processes, is a disjoint union of the form

$$\begin{array}{rcll} Act & = & & \{\alpha? \mid \alpha \in Names\} \;\cup \\ & & \cup & \{\alpha! \mid \alpha \in Names\} \;\cup \\ & & \cup & \{\tau\} \end{array} \qquad (2.1)$$

Objects, which can be used in input or output actions, may have different nature (both material and not material). For example, they may be

- material resources,
- people
- money
- information
- energy
- etc.

In addition, the concept of an input and an output can have a virtual character, i.e. the words *input* and *output* may only be used as a metaphor, but in reality no input or output of any real object may not occur. Informally, we consider a non-internal action of a process P as

- an **input action**, if this action was caused by a process from an environment of P, and
- an **output action**, if it was caused by P.

For each name $\alpha \in Names$ the actions $\alpha?$ and $\alpha!$ are said to be **complementary**.

We shall use the following notation.

1. For every action $a \in Act \setminus \{\tau\}$ the symbol $\bar{a}$ denotes an action, which is complementary to a, i.e.
$$\overline{\alpha?} \overset{\text{def}}{=} \alpha!, \quad \overline{\alpha!} \overset{\text{def}}{=} \alpha?$$
2. For every action $a \in Act \setminus \{\tau\}$ the string $name(a)$ denotes the name specified in the action a, i.e.
$$name(\alpha?) \overset{\text{def}}{=} name(\alpha!) \overset{\text{def}}{=} \alpha$$
3. For each subset $L \subseteq Act \setminus \{\tau\}$
   - $\overline{L} \overset{\text{def}}{=} \{\overline{a} \mid a \in L\}$
   - $names(L) \overset{\text{def}}{=} \{name(a) \mid a \in L\}$

2.4 Definition of a process

A **process** is a triple P of the form

$$P = (S, s^0, R) \tag{2.2}$$

whose components have the following meanings.

- S is a set whose elements are called **states** of the process P.
- $s^0 \in S$ is a selected state, called an **initial state** of the process P.
- R is a subset of the form

  $$R \subseteq S \times Act \times S$$

  Elements of R are called **transitions**.

  If a transition from R has the form (s_1, a, s_2), then

  - we say that this transition is a transition from the state s_1 to the state s_2 with an execution of the action a,
  - states s_1 and s_2 are called a **start** and an **end** of this transition, respectively, and the action a is called a **label** of this transition, and
  - sometimes, in order to improve a visibility, we will denote this transition by the diagram

  $$s_1 \xrightarrow{a} s_2 \tag{2.3}$$

An **execution** of a process $P = (S, s^0, R)$ is a generation of a sequence of transitions of the form

$$s^0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots$$

with an execution of actions $a_0, a_1, a_2 \ldots$, which are labels of these transitions.

At every step $i \geq 0$ of this execution

- the process P is in some state s_i ($s_0 = s^0$),
- if there is at least one transition from R starting at s_i, then the process P

  - non-deterministically chooses a transition from R starting at s_i, labeled such action a_i, which can be executed at the current time (if there is no such transitions, then the process suspends until at least one such transition will occur)
  - performs the action a_i, and then
  - goes to the state s_{i+1}, which is the end of the selected transition

- if R does not contain transitions starting at s_i, then the process completes its work.

The symbol $Act(P)$ denotes the set of all actions in $Act \setminus \{\tau\}$, which can be executed by the process P, i.e.

$$Act(P) \overset{\text{def}}{=} \{a \in Act \setminus \{\tau\} \mid \exists\ (\, s_1 \xrightarrow{a} s_2 \,) \in R\}$$

Process (2.2) is said to be **finite**, if its components S and R are finite sets.

A finite process can be represented graphically as a diagram, in which

- each state is represented by a circle in the diagram, and an identifier of this state can be written in this circle
- each transition is represented by an arrow connecting beginning of this transition and its end, and a label of this transition is written on this arrow
- an initial state is indicated in some way
  (for example, instead of the usual circle, a double circle is drawn)

Examples of such diagrams contain in sections 2.2.2 and 2.2.3.

2.5 A concept of a trace

Let $P = (S, s^0, R)$ be a process.

A **trace** of the process P is a finite or infinite sequence

$$a_1, a_2, \ldots$$

of elements of Act, such that there is a sequence of states of the process P

$$s_0, s_1, s_2, \ldots$$

with the following properties:

- s_0 coincides with the initial state s^0 of P
- for each $i \geq 1$ the set R contains the transition

$$s_i \xrightarrow{a_i} s_{i+1}$$

A set of all traces of the process P we shall denote by $Tr(P)$.

2.6 Reachable and unreachable states

Let P be a process of the form (2.2).

A state s of the process P is said to be **reachable**, if $s = s^0$, or there is a sequence of transitions of P, having the form

$$s_0 \xrightarrow{a_1} s_1\,, \quad s_1 \xrightarrow{a_2} s_2\,, \quad \ldots \quad s_{n-1} \xrightarrow{a_n} s_n$$

in which $n \geq 1,\ s_0 = s^0$ and $s_n = s$.

A state is said to be **unreachable**, if it is not reachable.

It is easy to see that after removing of all

- unreachable states from S, and
- transitions from R which does contain these unreachable states

the resulting process P' (which is referred as a **reachable part** of the process P) will represent exactly the same behavior, which is represented by the process P. For this reason, we consider such processes P and P' as equal.

2.7 Replacement of states

Let

- P be a process of the form (2.2),
- s be a state from S
- s' be an arbitrary element, which does not belong to the set S.

Denote by P' a process, which is obtained from P by replacement s on s' in the sets and S R, i.e. every transition in R of the form

$$s \xrightarrow{a} s_1 \quad \text{or} \quad s_1 \xrightarrow{a} s$$

is replaced by a transition

$$s' \xrightarrow{a} s_1 \quad \text{or} \quad s_1 \xrightarrow{a} s'$$

respectively.

As in the previous section, it is easy to see that P and P' represent the same behavior, and for this reason, we can consider such processes P and P' as equal.

It is possible to replace not only one state, but arbitrary subset of states of the process P. Such a replacement can be represented as an assignment of a bijection of the form

$$f : S \to S' \tag{2.4}$$

and the result of such replacement is by definition a process P' of the form

$$P' = (S', (s')^0, R') \tag{2.5}$$

where

- $(s')^0 \overset{\text{def}}{=} f(s^0)$, and
- for each pair $s_1, s_2 \in S$ and each $a \in Act$

$$(\, s_1 \xrightarrow{a} s_2 \,) \in R \quad \Leftrightarrow \quad (\, f(s_1) \xrightarrow{a} f(s_2) \,) \in R'.$$

Since the processes P and P' represent the same behavior, we can treat them as equal.

In the literature on the theory of processes such processes P and P' sometimes are said to be **isomorphic**. Bijection (2.4) with the above properties is called an **isomorphism** between P and P'. The process P' is said to be an **isomorphic copy** of the process P.

Chapter 3

Operations on processes

In this chapter we define several algebraic operations on the set of processes.

3.1 Prefix action

The first such operation is called a **prefix action**, this is an unary operation denoted by "$a.$", where a is an arbitrary element of Act.

Let $P = (S, s^0, R)$ be a process and $a \in Act$.

An effect of the operation $a.$ on the process P results to the process, which has the following components:

- a set of states of $a.P$ is obtained from S by an adding a new state $s \notin S$
- an initial state of $a.P$ is the added state s
- a set of transitions of $a.P$ is obtained from R by adding a new transition of the form
$$s \xrightarrow{a} s^0$$

The resulting process is denoted by

$$a.P$$

We illustrate an effect of this operation on the example of a vending machine presented at section 2.2.2. Denote the process, which represents a behavior of this automaton, by P_{vm}.

Extend the set of actions of the vending machine by a new input action

$$enable\,?$$

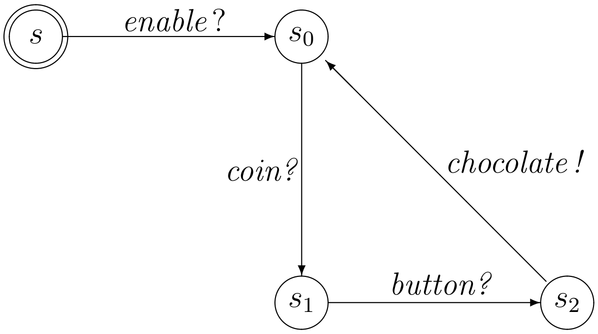
which means an enabling of this machine.

The process *enable*?. P_{vm} represents a behavior of the new vending machine, which in the initial state can not

- accept coins,
- perceive pressing the button, and
- output chocolates.

The only thing that he can is to be enabled. After that, its behavior will be no different from that of the original machine.

A graph representation of *enable*?. P_{vm} has the following form:



3.2 Empty process

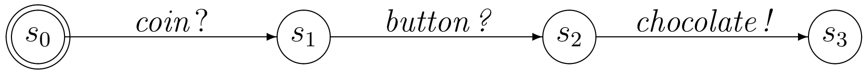
Among all the processes, there is one the most simple. This process has only one state, and has no transitions. To indicate such a process we use a constant (i.e. a 0-ary operation) **0**.

Returning to examples with vending machines, it can be said that the process **0** represents a behavior of a broken vending machine, that is such a machine, which can not execute any action.

By applying the operations of prefix action to the process **0** it is possible to define a behavior of more complex machines. Consider, for example, the following process:

$$P = coin?.button?.chocolate!.\,\mathbf{0}$$

A graph representation of this process is as follows:



This process defines a behavior of a vending machine, which serves exactly one customer, and after this breaks.

3.3 Alternative composition

Next operation on processes is a binary operation, which is called an **alternative composition**.

This operation is used in the case when, having a pair of processes P_1 and P_2, we must construct a process P, which will operate

- either as the process P_1,
- or as the process P_2,

and the choice of a process, according to which P will operate, can be determined

- either by P itself,
- or by an environment in which P does operate.

For example, if P_1 and P_2 have the form

$$\begin{array}{l} P_1 = \alpha\,?\,.\,P_1' \\ P_2 = \beta\,?\,.\,P_2' \end{array} \tag{3.1}$$

and at the initial time an environment of P

- can give P the object α, but
- can not give P the object β

then P will choose a behavior which is only possible in this situation, i.e. will operate according to the process P_1.

Note that in this case it is chosen such a process, first action in which can be executed in the current time. After choosing of P_1, and execution of the action $\alpha\,?$, the process P is obliged to continue its work according to this choice, i.e. it will operate like P_1'. It is possible, that after execution of the action $\alpha?$

- P will not be able to execute any action, working in accordance with P_1'
- though at this time P will be able to execute an action, working in accordance with P_2'.

But at this time P can not change his choice (i.e. can not choose P_2' instead of P_1'). P can only wait until it will be possible to work in accordance with P_1'.

If in the initial time the environment can give P both α and β, then P chooses a process whereby it will work,

- non-deterministically (i.e., arbitrarily), or
- subject to some additional factors.

The exact definition of the operation of alternative composition is as follows.

Let P_1 and P_2 be processes of the form

$$P_i = (S_i, s_i^0, R_i) \quad (i = 1, 2)$$

and the sets of states S_1 and S_2 have no common elements.

An **alternative composition** of processes P_1 and P_2 is a process

$$P_1 + P_2 = (S, s^0, R)$$

whose components are defined as follows.

- S is obtained by adding to the union $S_1 \cup S_2$ a new state s^0, which is an initial state of $P_1 + P_2$
- R contains all transitions from R_1 and R_2, and

- for each transition in R_i $(i = 1, 2)$

$$s_i^0 \xrightarrow{a} s$$

R contains the transition

$$s^0 \xrightarrow{a} s$$

If S_1 and S_2 have common elements, then to define a process $P_1 + P_2$ you first need to replace in S_2 those states that are also in S_1 on new states, and also modify accordingly R_2 and s_2^0.

Consider, for example, vending machine which sells two types of drinks: cola and fanta, and

- if a customer puts in a coin *coin_1*, then the machine issues a glass of cola, and
- if a customer puts in a coin *coin_2*, then a machine gives a glass of fanta

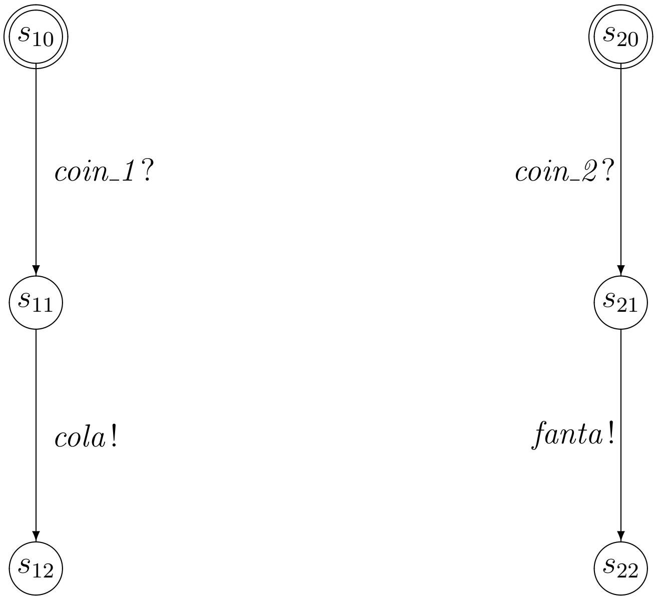
with the machine is broken immediately after the sale of one glass of a drink.

A behavior of this automaton is described by the following process:

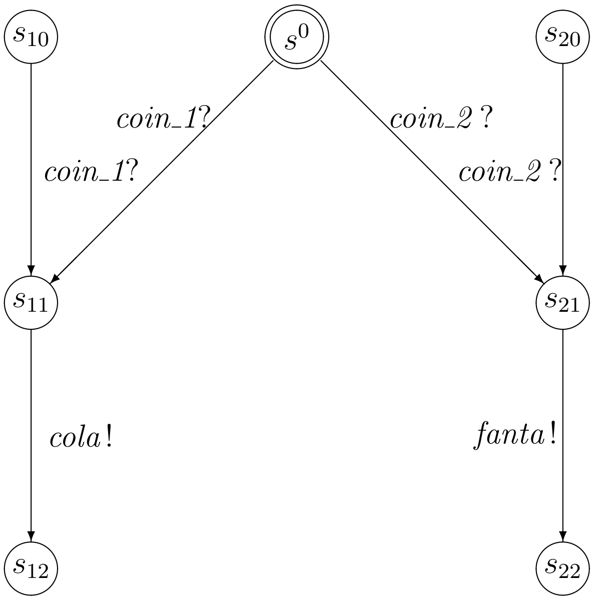
$$\begin{array}{rcl} P_{\text{drink}} & = & \mathit{coin\_1}\,? \,.\, \mathit{cola}\,! \,.\, \mathbf{0} \quad + \\ & + & \mathit{coin\_2}\,? \,.\, \mathit{fanta}\,! \,.\, \mathbf{0} \end{array} \tag{3.2}$$

Consider a graph representation of process (3.2).

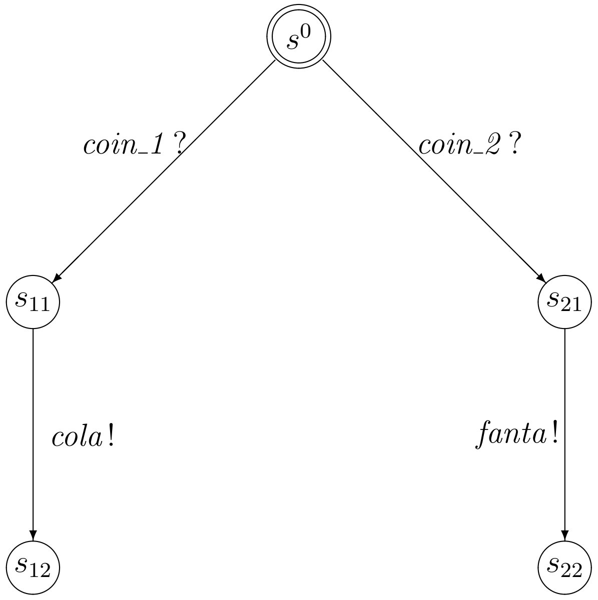
Graph representation of terms in the sum (3.2) have the form



According to a definition of an alternative composition, a graph representation of process (3.2) is obtained by adding to the previous diagram a new state and the corresponding transitions, to result in the following diagram:



Since the states s_{10} and s_{20} are unreachable, it follows that it is possible to delete them and transitions associated with them, resulting in a diagram



which is the desired graph representation of process (3.2).

Consider another example. We describe an exchange machine, which can enter banknotes in denominations of 100 dollars. The machine shall issue

- either 2 banknotes on 50 dollars,
- or 10 banknotes on 10 dollars

and the choice of method of an exchange is carried regardless of the wishes of the customer. Just after one session of an exchange the machine is broken.

$$\begin{aligned} &P_{\text{exchange}} = \\ &= \mathit{1\_on\_1000}\,?\,.(\mathit{2\_on\_500}\,!\,.\mathbf{0} \;+\; \mathit{10\_on\_100}\,!\,.\mathbf{0}) \end{aligned}$$

These two examples show that the operation of an alternative composition can be used to describe at least two fundamentally different situations.

1. First, it can express a dependence of system behavior from the behavior of its environment.

For instance, in the case of a vending machine P_{drink} a behavior of the machine is determined by an action of a purchaser, namely by a denomination of a coin, which a purchaser introduced into the machine.

In this case, a process representing a behavior of the simulated vending machine is **deterministic**, i.e. its behavior is uniquely determined by input actions.

2. In the second, on an example of a machine P_{exchange} we see that for the same input action is possible different response of the machine.

   This is an example of a **nondeterminism**, i.e. an uncertainty of a behavior of a system.

   Uncertainty in a behavior of systems can occur by at least two reasons.

   (a) First, a behavior of systems may depend on **random factors**.

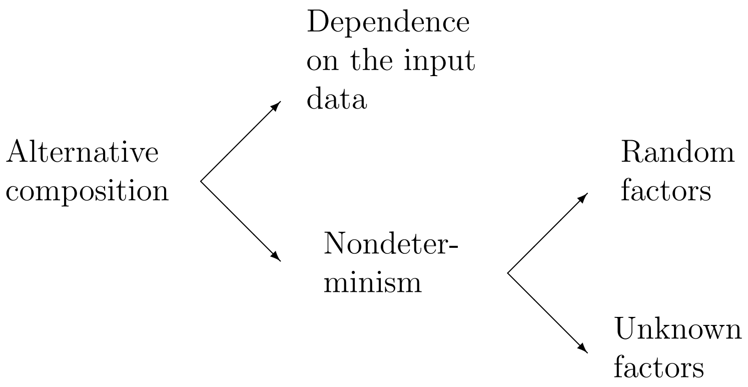
   These factors can be, for example,

   - failures in hardware,
   - conflicts in a computer network
   - absence of banknotes of required value at an ATM
   - or anything else

   (b) Second, a model is always some abstraction or simplification of a real system, and some of the factors influencing a behavior of this system may be eliminated from a consideration.

   In particular, on the example of P_{exchange} we see that a real reason of choosing of a variant of behavior of the machine can be not taken into account in the process, which is a model of a behavior of this machine.

One can schematically represent the above variants of using alternative composition as follows:



3.4 Parallel composition

The operation of parallel composition is used for building models of behavior of dynamic systems, composed of several communicating components.

Before giving a formal definition of this operation, we will discuss the concept of parallel working of a pair of systems Sys_1 and Sys_2, which we consider as components of a system Sys, i.e.

$$Sys \stackrel{\text{def}}{=} \{Sys_1, Sys_2\} \tag{3.3}$$

Let processes P_1 and P_2 represent behaviors of the systems Sys_1 and Sys_2 respectively.

When the system Sys_i $(i = 1, 2)$ works as a part of the system Sys, its behavior is described by the same process P_i.

Denote by $\{P_1, P_2\}$ a process, describing a behavior of (3.3). The purpose of this section is to find an explicit description of $\{P_1, P_2\}$ (i.e. to define a sets of its states and transitions).

Here to simplify the exposition, we identify the concepts

“a process P”, and

“a system whose behavior is described by a process P”

As noted above, an execution of arbitrary process can be interpreted as a bypassing of a graph corresponding to this process, with an execution of actions that are labels of passable edges.

We shall assume that in passage of each edge $s \xrightarrow{a} s'$

- a transition from s to s' occurs instantaneously, and
- an execution of the action a occurs precisely at the time of this transition.

In fact, an execution of each action occurs within a certain period of time, but we shall assume that for each traversed edge $s \xrightarrow{a} s'$

- before the completion of an execution of the action a the process P is in the state s, and
- after the completion of an execution of a the process P instantly transforms into the state s'.

Since an execution of various actions has different durations, then we will assume that the process P is in each state an indefinite period of time during its execution.

Thus, an execution of the process P consists of an alternation of the following two activities:

- waiting for an indefinite period of time in one of the states, and
- instantaneous transition from one state to another.

Waiting in one of the states can occur

- not only because there is an execution of some action at this time,
- but also because the process P can not perform any action at this time.

For example, if

- $P = \alpha\,?.\,P'$, and
- in the initial time there is no a process who can give P an oblect with the name α

then P would wait until some process will give him an oblect with the name α.

As we know, for each process

- its actions are either input, or output, or internal, and

- each input or output action is a result of a communication of this process with other process.

Each input or output action of the process P_i $(i = 1, 2)$

- either is a result of communication of P_i with a process outside of the set $\{P_1, P_2\}$,
- or is a result of communication of P_i with the process P_j, where $j \in \{1, 2\} \setminus \{i\}$.

From the point of view of the process $\{P_1, P_2\}$, actions of the second type are internal actions of this process, because they

- are not a result of a communication of the process $\{P_1, P_2\}$ with its environment, and
- are the result of communication between the components of this process.

Thus, each step of the process $\{P_1, P_2\}$

(a) either is a result of a comminication of one of the processes P_i $(i = 1, 2)$ with a process outside of $\{P_1, P_2\}$,

(b) or is an internal action of P_1 or P_2,

(c) or is an internal action, which is a result of a communication of P_1 and P_2, and this communication has the following form:

   - one of these processes, say P_i, passes to another process P_j $(j \in \{1, 2\} \setminus \{i\})$ some object, and
   - the process P_j at the same time takes this object from the process P_i

   (This kind of a communication is called a **synchronous communication**, or a **handshaking**).

Each possible variant of a behavior of the process P_i $(i = 1, 2)$ can be associated with a **thread** denoted by the symbol σ_i. A **thread** is a vertical line, on which there are drawn points with labels, where

- labels of points represent actions executed by the process P_i, and
- labelled points are arranged in a chronological order, i.e.
  - first point is labelled by a first action of the process P_i,
  - second point (which is located under the first point) is labelled by a second action of the process P_i,
  - etc.

For each labelled point p on the thread, we denote by $act(p)$ a label of this point.

Assume that there is drawn on a plane a couple of parallel threads

$$\sigma_1 \;\; \sigma_2 \tag{3.4}$$

where $\sigma_i \;\; (i = 1, 2)$ represents a possible variant of a behavior of the process P_i in the process $\{P_1, P_2\}$.

Consider those labelled points on the threads from (3.4), which correspond to actions of the type (c), i.e. to communications of processes P_1 and P_2. Let p be one of such points, and, for example, it is on the thread σ_1.

According to the definition of a communication, at the same time, in which there is executed the action $act(p)$, the process P_2 executes a complementary action, i.e. there is a point p' on the thread σ_2, such that

- $act(p') = \overline{act(p)}$, and
- actions $act(p)$ and $act(p')$ execute at the same time.

Note that

- in the thread σ_2 may be several points with the label $\overline{act(p)}$, but exactly one of these points corresponds to the action, which is executed jointly with the action corresponding to the point p, and
- in the thread σ_1 may be several points with the label $act(p)$, but exactly one of these points corresponds to the action, which is executed jointly with the action corresponding to the point p'.

Transform our diagram of threads (3.4) as follows: for each pair of points p, p' with the above properties

- join the points p and p' by an arrow,
  - the start of which is the one of these points, which has a label of the form $\alpha\,!$, and
  - the end of which is another of these points
- draw a label α on this arrow, and
- replace labels of the points p and p' on τ.

The arrow from p to p' is called a **synchronization arrow**. Such arrows usually are drawn horizontally.

After such changes for all pairs of points, which are labelled by actions of the type (c), we will obtain a diagram, which is called a **Message Sequence Chart (MSC)**. This diagram represents one of possible variants of execution of the process $\{P_1, P_2\}$.

We shall denote a set of all MSCs, each of which corresponds to some variant of execution of the process $\{P_1, P_2\}$, as

$$Beh\{P_1, P_2\}$$

Consider the following example of a process of the form $\{P_1, P_2\}$:

- P_1 is a model of a vending machine, whose behavior is given by

  $$P_1 = coin\,?.\ chocolate\,!.\mathbf{0} \tag{3.5}$$

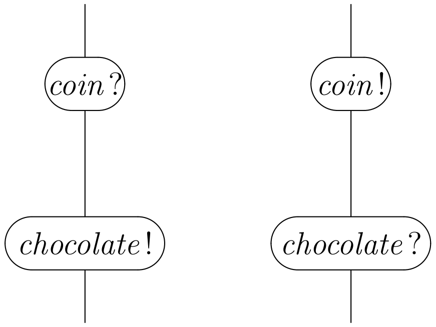
  (i.e., the machine gets a coin, gives a chocolate, and then breaks)

- P_2 is a model of a customer, whose behavior is given by

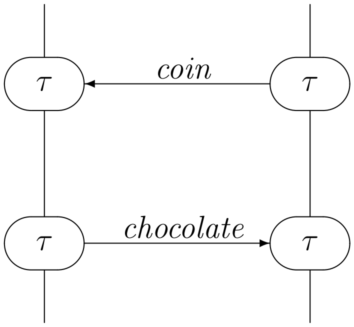
  $$P_2 = coin\,!.\ chocolate\,?.\mathbf{0} \tag{3.6}$$

  (i.e., the customer drops a coin, receives a chocolate, and then ceases to function as customer).

Threads of these processes have the form



If all actions on these threads are actions of the type (c), then this diagram can be transformed into the following MSC:



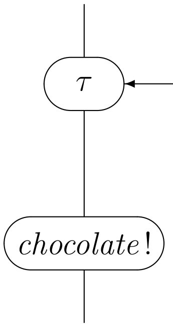
However, it is possible the following variant of execution of the process $\{P_1, P_2\}$:

- first actions of P_1 and P_2 are of the type (c), i.e. the customer drops a coin, and the machine accepts the coin
- second action of automaton P_1 is a communication with a process that is external with respect to $\{P_1, P_2\}$
(that is, for example, a thief walked up to the machine, and took a chocolate, before than the customer P_2 was able to take it)

In this situation, the customer can not execute a second action as an internal action of $\{P_1, P_2\}$. According to a description of the process P_2, in this case two variants of behavior of the customer are possible.

1. The customer will be in a state of endless waiting.

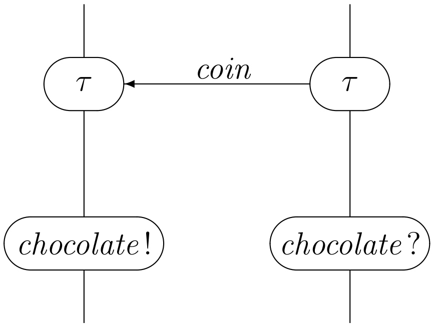
   The corresponding MSC has the form



2. The customer will be able successfully complete its work.

   This would be the case if some process external to to $\{P_1, P_2\}$ will give a chocolate to the customer.

   The corresponding MSC has the form



Now consider the general question: how a process of the form $\{P_1, P_2\}$ can be defined explicitly, i.e. in terms of states and transitions.

At first glance, this question is incorrect, because $\{P_1, P_2\}$ must be a model of a **parallel** execution of the processes P_1 and P_2, in which

- it can be possible a simultaneous execution of actions by both processes P_1, P_2,

- and, therefore, the process $\{P_1, P_2\}$ can execute such actions, which are pairs of actions from the set Act, which can not belong to the set Act (by assumption).

Note on this, that absolute simultaneity holds only for those pairs of actions that generate an internal action of the process $\{P_1, P_2\}$ of the type (c).

For all other pairs of actions of the processes P_1 and P_2, even if they occurred simultaneously (in terms of external observer), we can assume without loss of generality, that one of them happened a little earlier or a little later than another.

Thus, we can assume that the process $\{P_1, P_2\}$ executes consequentially, i.e. under any variant of an execution of the process $\{P_1, P_2\}$ actions executed by them form some linearly ordered sequence

$$tr = (act_1, act_2, \ldots) \tag{3.7}$$

in which the actions are ordered by the time of their execution: at first it was executed act_1, then - act_2, etc.

Because each possible variant of an execution of the process $\{P_1, P_2\}$ can be represented by a MSC, then we can assume that sequence (3.7) can be obtained by some *linearization* of this MSC (i.e., by "pulling" it in a chain).

For a definition of a linearization of a MSC we introduce some auxiliary concepts and notations.

Let C be a MCS. Then

- $Points(C)$ denotes a set of all points belonging to the MSC C,
- for each point $p \in Points(C)$ $act(p)$ denotes an action, ascribed to the point p
- for each pair of points $p, p' \in Points(C)$ the formula

  $$p \to p'$$

  means that one of the following conditions does hold:

  - p and p' are in the same thread, and p' is lower than p, or
  - there is a synchronization arrow from p to p'

- for each pair of points $p, p' \in Points(C)$ the formula

  $$p \leq p'$$

  means that either $p = p'$, or there is a sequence of points $p_1, \ldots, p_k$, such that

  - $p = p_1, \;\; p' = p_k$
  - for each $i = 1, \ldots, k-1 \quad p_i \to p_{i+1}$

The relation $\leq$ on points of a MSC can be regarded as a relation of a chronological order, i.e. the formula $p \leq p'$ can be interpreted as stating that

- the points p and p' are the same or connected by a synchronization arrow
  (i.e. actions in p and p' coincide)
- or an action in the p' occurred later than there was an action in the p.

The exact definition of a linearization of a MSC has the following form. Let

- C be a MSC,
- tr be a sequence of actions of the form (3.7), and
- $Ind(tr)$ be a set of indices of elements of the sequence tr, i.e.

  $$Ind(tr) = \{1, 2, \ldots\}$$

  (this set can be finite or infinite)

The sequence tr is called a **linearization** of the MSC C, if there is a surjective mapping

$$lin : Points(C) \to Ind(tr)$$

satisfying the following conditions.

1. for each pair $p, p' \in Points(C)$

   $$p \leq p' \quad \Rightarrow \quad lin(p) \leq lin(p')$$

2. for each pair $p, p' \in Points(C)$ the equality
$$lin(p) = lin(p')$$
holds if and only if
   - $p = p'$, or
   - there is a synchronization arrow from p to p'
3. $\forall\, p \in Points(C) \quad act(p) = act_{lin(p)}$.

i.e. the mapping *lin*

- preserves the chronological order
- identifies those points of the MSC C, which correspond to one action of $\{P_1, P_2\}$, and
- does not identify any other points.

Denote by $Lin(C)$ the set of all linearizations of the MSC C.

Now the problem of explicit description of the process $\{P_1, P_2\}$ can be formulated as follows: construct a process P, satisfying the condition

$$Tr(P) = \bigcup_{C \in Beh\{P_1, P_2\}} Lin(C) \tag{3.8}$$

i.e. in the process P should be represented all linearizations of any possible joint behavior of processes P_1 and P_2.

Condition (3.8) is justified by the following consideration: because we do not know

- how clocks in the processes P_1 and P_2 are related, and
- what is a length of a stay in each state in which these processes fall

then we must take into account every possible order of an execution of actions, which does not contradict to the relation of a chronological order.

Begin the construction of a process P, satisfying condition (3.8).

Let the processes P_1 and P_2 have the form

$$P_i = (S_i, s_i^0, R_i) \quad (i = 1, 2)$$

Consider any linearization tr of an arbitrary MSC from $Beh\{P_1, P_2\}$

$$tr = (a_1, a_2, \dots)$$

Draw a line, which will be interpreted as a scale of time. Select on this line points p_1, p_2, ... labelled by the actions $a_1, a_2, \dots$ respectively, such that these actions are located on the line in the same order in which they are listed in tr.

Let the symbols $I_0, I_1, I_2, \dots$ denote the following sections of this line:

- I_0 is the set of all points of the line before the point p_1, i.e.

$$I_0 \overset{\text{def}}{=}]-\infty, p_1[$$

- for each $i \geq 1$ the plot I_i consists of points between p_i and p_{i+1}, i.e.

$$I_i \overset{\text{def}}{=}]p_i, p_{i+1}[$$

Each of these sections I_i can be interpreted as an interval of time during which the process P does not perform any action, i.e. at times between p_i and p_{i+1} the processes P_1 and P_2 are in fixed states $(s_1)_i$ and $(s_2)_i$, respectively.

Denote by s_i the pair $((s_1)_i, (s_2)_i)$. This pair can be interpreted as a state of the process P, in which he is at each time from the interval I_i.

By the definition of the sequence tr, we have one of two situations.

1. The action a_i has a type (a) or (b), i.e. a_i was executed by one of the processes included in P.

   There are two cases.

   (a) The action a_i was executed by the process P_1.

   In this case we have the following relation between the states s_i and s_{i+1}:

   - $(s_1)_i \xrightarrow{a_i} (s_1)_{i+1} \quad \in R_1$
   - $(s_2)_{i+1} = (s_2)_i$

   (b) The action a_i was executed by the process P_2.

   In this case we have the following relation between the states s_i and s_{i+1}:

- $(s_2)_i \xrightarrow{a_i} (s_2)_{i+1} \quad \in R_2$
- $(s_1)_{i+1} = (s_1)_i$

2. The action a_i is of the type (c).

   In this case we have the following relation between the states s_i and s_{i+1}:

   - $(s_1)_i \xrightarrow{a} (s_1)_{i+1} \quad \in R_1$
   - $(s_2)_i \xrightarrow{\overline{a}} (s_2)_{i+1} \quad \in R_2$

   for some $a \in Act \setminus \{\tau\}$.

The above properties of the sequence tr can be reformulated as follows: tr is a trace of the process

$$(S, s^0, R) \tag{3.9}$$

whose components are defined as follows:

- $S \overset{\text{def}}{=} S_1 \times S_2 \overset{\text{def}}{=} \{(s_1, s_2) \mid s_1 \in S_1, s_2 \in S_2\}$
- $s^0 \overset{\text{def}}{=} (s_1^0, s_2^0)$
- for

  - each transition $s_1 \xrightarrow{a} s_1'$ from R_1, and
  - each state $s \in S_2$

  R contains the transition

  $$(s_1, s) \xrightarrow{a} (s_1', s)$$

- for

  - each transition $s_2 \xrightarrow{a} s_2'$ from R_2, and
  - each state $s \in S_1$

  R contains the transition

  $$(s, s_2) \xrightarrow{a} (s, s_2')$$

- for each pair of transitions with complementary labels

$$\begin{array}{lll} s_1 \xrightarrow{a} s'_1 & \in R_1 \\ s_2 \xrightarrow{\overline{a}} s'_2 & \in R_2 \end{array}$$

  R contains the transition

$$(s_1, s_2) \xrightarrow{\tau} (s'_1, s'_2)$$

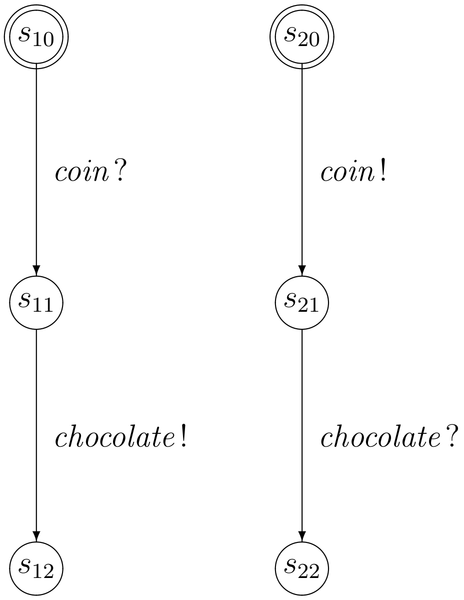
It is easy to show the converse: each trace of process (3.9) is a linearization of some MSC C from the set $Beh\{P_1, P_2\}$.

Thus, an explicit representation of the process $P = \{P_1, P_2\}$ can be defined as process (3.9). This process is called a **parallel composition** of the processes P_1 and P_2, an is denoted as

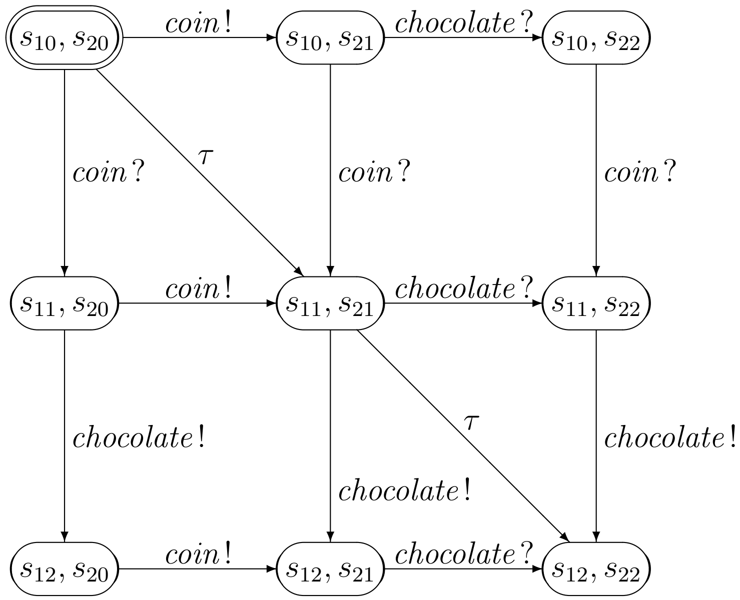
$$P_1 \,|\, P_2$$

We give an example of the process $P_1 \,|\, P_2$, in the case where the processes P_1 and P_2 represent behaviors of a vending machine and a customer (see (3.5) and (3.6)).

A graph representation of these processes have the form



A graph representation of the process $P_1|P_2$ has the form



Note that a size of the set of states of $P_1|P_2$ is equal to a product of sizes of sets of states of P_1 and P_2. Thus, a size of a description of the process $P_1 \mid P_2$ may substantially exceed the total complexity of sizes of descriptions of its components, P_1 and P_2. This may make impossible to analyze this process, if it is represented in an explicit form, because of its high complexity.

Therefore, in practical problems of an analysis of processes of the form $P_1 \mid P_2$, instead of an explicit construction of $P_1 \mid P_2$ there is constructed a process, in which each MSC from $Beh\{P_1, P_2\}$

- is not represented by all possible linearizations, but
- is represented by at least one linearization.

A complexity of such process can be significantly less in comparison with a complexity of the process $P_1|P_2$.

A construction of a process of this kind makes sense, for example, if an analyzed property φ of the process $P_1 \mid P_2$ has the following quality: for arbitrary $C \in Beh\{P_1, P_2\}$

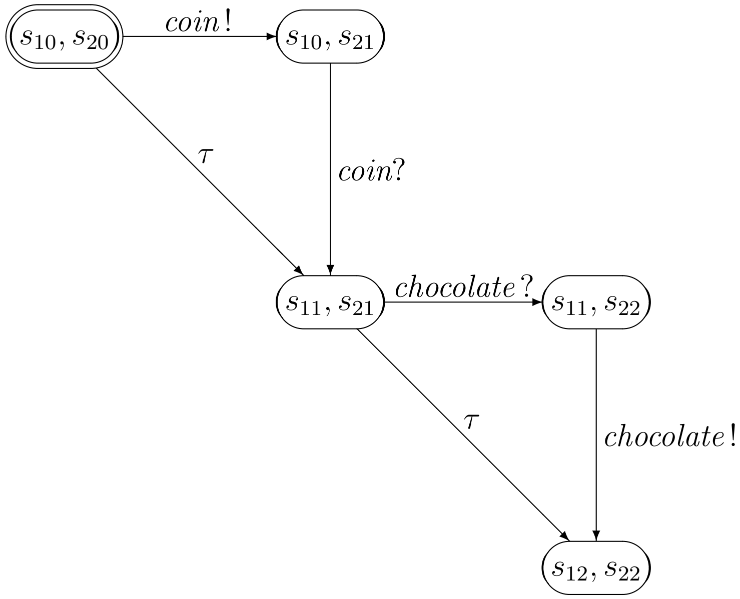
- if φ holds *for one of linearizations* of C,

- then φ holds *for all linearizations* of C.

Typically, a process in which each MSC from $Beh\{P_1, P_2\}$ is represented by at least one linearization, is constructed as a certain **subprocess** of the process $P_1|P_2$, i.e. is obtained from $P_1|P_2$ by removing of some states and associated transitions. Therefore, such processes are said to be **reduced**.

The problem of constructing of reduced processes is called a **partial order reduction**. This problem has been intensively studied by many leading experts in the field of verification.

Consider, for example, a reduced process for the above process $P_1 \,|\, P_2$, consisting of a vending machine and the customer.



In conclusion, we note that the problem of analyzing of processes consisting of several communicating components, most often arises in situations where such components are computer programs and hardware devices of a computer system. A communication between programs in such system is implemented by **mediators**, i.e. by certain processes which can communicate synchronously with programs.

Communications between programs are usually implemented by the following two ways.

1. **Communication through shared memory.**

In this case, mediators are memory cells accessed by both programs.

A communication in this case can be implemented as follows: one program writes an information in these cells, and other program reads contents of cells.

2. **Communicaton by sending messages.**

   In this case, a mediator is a channel, which can be used by programs for the following actions:

   - sending a message to the channel, and
   - receiving of a message from the channel.

   The channel may be implemented as a buffer storing several messages. Messages in the channel can be organized on the principle of queue (i.e., messages leave the channel in the same order in which they had come).

3.5 Restriction

Let

- $P = (S, s^0, R)$ be a process, and
- L be a subset of the set $Names$.

A **restriction** of P with respect to L is the process

$$P \setminus L = (S, s^0, R')$$

which is obtained from P by removing of those transitions that have labels with the names from L, i.e.

$$R' \overset{\text{def}}{=} \left\{ (s \xrightarrow{a} s') \in R \;\middle|\; \begin{array}{l} a = \tau, \text{ or} \\ name(a) \notin L \end{array} \right\}$$

As a rule, the operation of a restriction is used together with the operation of parallel composition, for representation of processes that

- consist of several communicating components, and

- a communication between these components must satisfy certain restrictions.

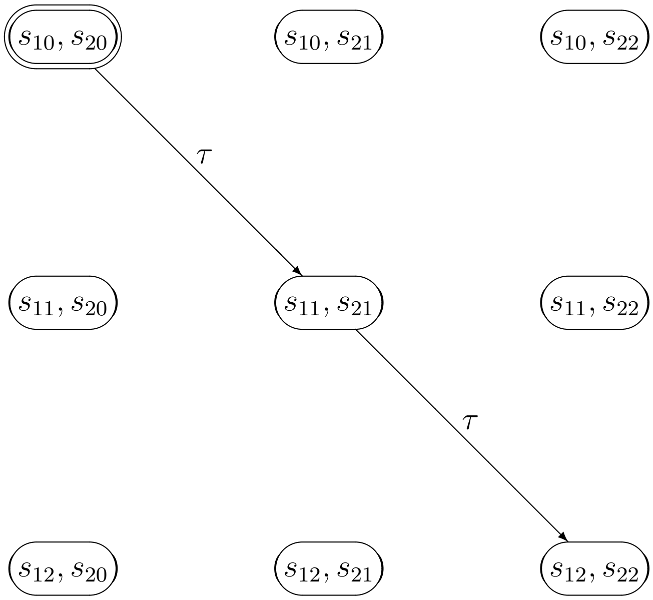
For example, let processes P_1 and P_2 represent a behavior of a vending machine and a customer respectively, which were discussed in the previous section.

We would like to describe a process, which is a model of such parallel execution of processes $P_1$and P_2, at which these processes can execute actions associated with buying and selling of a chocolate only jointly.

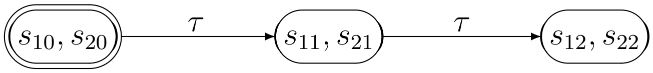
The desired process can be obtained by an application to the process $P_1|P_2$ the operation of a restriction with respect to the set of names of all actions related to buying and selling of a chocolate. This process is described by the expression

$$P \overset{\text{def}}{=} (P_1|P_2) \setminus \{coin, chocolate\} \tag{3.10}$$

A graph representation of process (3.10) has the form



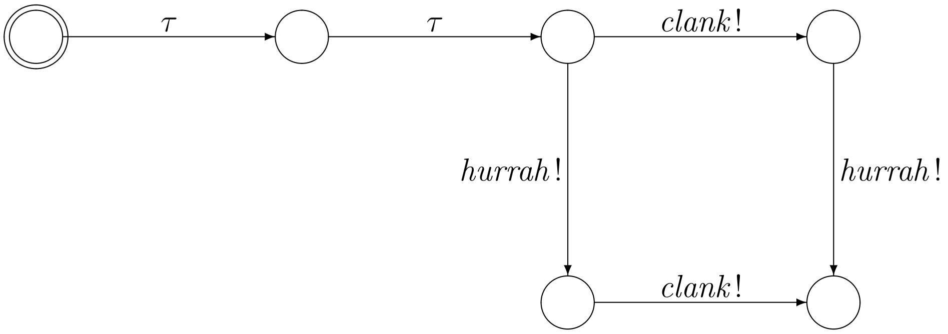
After removing unreachable states we get a process with the following graph representation:



Consider another example. Change a definition of a vending machine and a customer: let them also to send a signal indicating successful completion of their work. For example, these processes may have the following form:

$$P_1 \overset{\text{def}}{=} coin?.chocolate!.clank!.\mathbf{0}$$
$$P_2 \overset{\text{def}}{=} coin!.chocolate?.hurrah!.\mathbf{0}$$

In this case, a graph representation of process (3.10), after a removal of unreachable states, has the form



This process allows execution only those non-internal actions that are not related to buying and selling a chocolate.

Note that in this case

- in process (3.10) a nondeterminism is present, although
- in the components of P_1 and P_2 a nondeterminism is absent.

The cause of a nondeterminism in (3.10) is our incomplete knowledge about the simulated system: because we do not have a precise knowledge about a duration of actions *clank*! and *hurrah*!, then the model of the system should allow any order of execution of these actions.

3.6 Renaming

The last operation that we consider is an unary operation, which is called a **renaming**.

To define this operation, it is necessary to define a mapping of the form

$$f : Names \to Names \tag{3.11}$$

An effect of the operation of renaming on process P is changing labels of transitions of P:

- any label of the form $\alpha\,?$ is replaced on $f(\alpha)\,?$, and
- any label of the form $\alpha\,!$ is replaced on $f(\alpha)\,!$

The resulting process is denoted by $P[f]$.

We shall refer any mapping of the form (3.11) also as a **renaming**.

If a renaming f acts non-identically only on the names

$$\alpha_1, \ldots, \alpha_n$$

and maps them to the names

$$\beta_1, \ldots, \beta_n$$

respectively, then the process $P[f]$ can be denoted also as

$$P[\beta_1/\alpha_1, \ldots, \beta_n/\alpha_n]$$

The operation of renaming can be used, for example, in the following situation: this operation allows to use several copies of a process P as different components in constructing of a more complex process P'. Renaming serves for prevention of collisions between names of actions used in different occurrences of P in P'.

3.7 Properties of operations on processes

In this section we give some elementary properties of defined above operations on processes. All these properties have a form of equalities. For the first two properties, we give their proof, other properties are listed without comments in view of their evidence.

Recall (see section 2.7), that we consider two processes as equal, if

- they are isomorphic, or
- one of these processes can be obtained from another by removing some of unreachable states and transitions which contain unreachable states.

1. Operation $+$ is associative, i.e. for any processes P_1, P_2 and P_3 the following equality holds:

$$(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3) \tag{3.12}$$

Indeed, let the processes P_i $(i = 1, 2, 3)$ have the form

$$P_i = (S_i, s_i^0, R_i) \quad (i = 1, 2, 3) \tag{3.13}$$

and their sets of states S_1, S_2 and S_3 are pairwise disjoint. Then both sides of equality (3.12) are equal to the process $P = (S, s^0, R)$, whose components are defined as follows:

- $S \overset{\text{def}}{=} S_1 \cup S_2 \cup S_3 \cup \{s^0\}$, where s^0 is a new state (which does not belong to S_1, S_2 and S_3)
- R contains all transitions from R_1, R_2 and R_3
- for each transition from R_i $(i = 1, 2, 3)$ of the form

$$s_i^0 \xrightarrow{a} s$$

R contains the transition $s^0 \xrightarrow{a} s$

The property of associativity of the operation $+$ allows to use expressions of the form

$$P_1 + \ldots + P_n \tag{3.14}$$

because for any parenthesization of the expression (3.14) we shall get one and the same process.

A process, which is a value of expression (3.14) can be described explicitly as follows.

Let the processes P_i $(i = 1, \ldots, n)$ have the form

$$P_i = (S_i, s_i^0, R_i) \quad (i = 1, \ldots, n) \tag{3.15}$$

with the sets of states $S_1, \ldots, S_n$ are pairwise disjoint. Then a process, which is a value of the expression (3.14), has the form

$$P = (S, s^0, R)$$

where the components S, s^0, R are defined as follows:

- $S \overset{\text{def}}{=} S_1 \cup \ldots \cup S_n \cup \{s^0\}$, where s^0 is a new state (which does not belong to $S_1 \ldots, S_n$)
- R contains all transitions from $R_1, \ldots, R_n$
- for each transition from R_i $(i = 1, \ldots, n)$ of the form

  $$s_i^0 \xrightarrow{a} s$$

  R contains the transition $s^0 \xrightarrow{a} s$

2. The operation $|$ is associative, i.e. for any processes P_1, P_2 and P_3 the following equality holds:

   $$(P_1 \,|\, P_2) \,|\, P_3 = P_1 \,|\, (P_2 \,|\, P_3) \tag{3.16}$$

   Indeed, let the processes P_i $(i = 1, 2, 3)$ have the form (3.13). Then both sides of (3.16) are equal to the process $P = (S, s^0, R)$ whose components are defined as follows:

   - $S \overset{\text{def}}{=} S_1 \times S_2 \times S_3 \overset{\text{def}}{=}$
     $\overset{\text{def}}{=} \{(s_1, s_2, s_3) \mid s_1 \in S_1, s_2 \in S_2, s_3 \in S_3\}$
   - $s^0 \overset{\text{def}}{=} (s_1^0, s_2^0, s_3^0)$
   - for
     - each transition $s_1 \xrightarrow{a} s_1'$ from R_1, and
     - each pair of states $s_2 \in S_2, s_3 \in S_3$

     R contains the transition

     $$(s_1, s_2, s_3) \xrightarrow{a} (s_1', s_2, s_3)$$

   - for
     - each transition $s_2 \xrightarrow{a} s_2'$ from R_2, and
     - each pair of states $s_1 \in S_1, s_3 \in S_3$

     R contains the transition

     $$(s_1, s_2, s_3) \xrightarrow{a} (s_1, s_2', s_3)$$

   - for

  - each transition $s_3 \xrightarrow{a} s'_3$ from R_3, and
  - each pair of states $s_1 \in S_1, s_2 \in S_2$

  R contains the transition

$$(s_1, s_2, s_3) \xrightarrow{a} (s_1, s_2, s'_3)$$

- for
  - each pair of transitions with complementary labels

$$\begin{array}{llll} s_1 & \xrightarrow{a} & s'_1 & \in R_1 \\ s_2 & \xrightarrow{\overline{a}} & s'_2 & \in R_2 \end{array}$$

    and
  - each state $s_3 \in S_3$

  R contains the transition

$$(s_1, s_2, s_3) \xrightarrow{\tau} (s'_1, s'_2, s_3)$$

- for
  - each pair of transitions with complementary labels

$$\begin{array}{llll} s_1 & \xrightarrow{a} & s'_1 & \in R_1 \\ s_3 & \xrightarrow{\overline{a}} & s'_3 & \in R_3 \end{array}$$

    and
  - each state $s_2 \in S_2$

  R contains the transition

$$(s_1, s_2, s_3) \xrightarrow{\tau} (s'_1, s_2, s'_3)$$

- for
  - each pair of transitions with complementary labels

$$\begin{array}{llll} s_2 & \xrightarrow{a} & s'_2 & \in R_2 \\ s_3 & \xrightarrow{\overline{a}} & s'_3 & \in R_3 \end{array}$$

    and
  - each state $s_1 \in S_1$

R contains the transition

$$(s_1, s_2, s_3) \xrightarrow{\tau} (s_1, s_2', s_3')$$

The property of associativity of the operation $\mid$ allows to use expressions of the form

$$P_1 \mid \ldots \mid P_n \tag{3.17}$$

because for any parenthesization of the expression (3.17) we shall get one and the same process.

A process, which is a value of expression (3.17) can be described explicitly as follows.

Let the processes P_i $(i = 1, \ldots, n)$ have the form (3.15). Then a process, which is a value of the expression (3.17), has the form

$$P = (S, s^0, R)$$

where the components S, s^0, R are defined as follows:

- $S \stackrel{\text{def}}{=} S_1 \times \ldots \times S_n \stackrel{\text{def}}{=}$
  $\stackrel{\text{def}}{=} \{(s_1, \ldots, s_n) \mid s_1 \in S_1, \ldots, s_n \in S_n\}$
- $s^0 \stackrel{\text{def}}{=} (s_1^0, \ldots, s_n^0)$
- for
  - each $i \in \{1, \ldots, n\}$
  - each transition $s_i \xrightarrow{a} s_i'$ from R_i, and
  - each list of states

    $$s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n$$

    where $\forall\, j \in \{1, \ldots, n\} \quad s_j \in S_j$

  R contains the transition

  $$(s_1, \ldots, s_n) \xrightarrow{a} (s_1, \ldots, s_{i-1}, s_i', s_{i+1}, \ldots, s_n)$$

- for
  - each pair of indices $i, j \in \{1, \ldots, n\}$, where $i < j$

- each pair of transitions with complementary labels of the form

  $$\begin{array}{lll} s_i \xrightarrow{a} s'_i & \in R_i \\ s_j \xrightarrow{\overline{a}} s'_j & \in R_j \end{array}$$

  and
- each list of states

  $$s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_{j-1}, s_{j+1}, \ldots, s_n$$

  where $\forall\, k \in \{1, \ldots, n\} \quad s_k \in S_k$

R contains the transition

$$(s_1, \ldots, s_n) \xrightarrow{\tau} \left(\begin{array}{l} s_1, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots, s_{j-1}, s'_j, \\ s_{j+1}, \ldots, s_n \end{array} \right)$$

3. The operation $+$ is commutative, i.e. for any processes P_1 and P_2 the following equality holds:

   $$P_1 + P_2 = P_2 + P_1$$

4. The operation $\mid$ is commutative, i.e. for any processes P_1 and P_2 the following equality holds:

   $$P_1 \,|\, P_2 = P_2 \,|\, P_1$$

5. $\mathbf{0}$ is a neutral element with respect to the operation $\mid$:

   $$P \,|\, \mathbf{0} = P$$

   The operation $+$ has a similar property, in this property there is used a concept of strong equivalence of processes (defined below) instead of equality of processes . This property, as well as the property of idempotency of the operation $+$ are proved in section 4.5 (theorem 4).

6. $\mathbf{0} \setminus L = \mathbf{0}$

7. $\mathbf{0}[f] = \mathbf{0}$

8. $P \setminus L = P$, if $L \cap names(Act(P)) = \emptyset$.

   (recall that $Act(P)$ denotes a set of actions $a \in Act \setminus \{\tau\}$, such that P contains a transition with the label a)

9. $(a.P) \setminus L = \begin{cases} \mathbf{0}, & \text{if } a \neq \tau \text{ and } name(a) \in L \\ a.(P \setminus L), & \text{otherwise} \end{cases}$

10. $(P_1 + P_2) \setminus L = (P_1 \setminus L) + (P_2 \setminus L)$

11. $(P_1 \,|\, P_2) \setminus L = (P_1 \setminus L) \,|\, (P_2 \setminus L)$, if

$$L \cap names(Act(P_1) \cap \overline{Act(P_2)}) = \emptyset$$

12. $(P \setminus L_1) \setminus L_2 = P \setminus (L_1 \cup L_2)$

13. $P[f] \setminus L = (P \setminus f^{-1}(L))[f]$

14. $P[id] = P$, where id is an identity function

15. $P[f] = P[g]$, if restrictions of functions f and g on the set $names(Act(P))$ are equal.

16. $(a.P)[f] = f(a).(P[f])$

17. $(P_1 + P_2)[f] = P_1[f] + P_2[f]$

18. $(P_1 \,|\, P_2)[f] = P_1[f] \,|\, P_2[f]$, if a restriction of f on the set

$$names(Act(P_1) \cup Act(P_2))$$

is an injective mapping.

19. $(P \setminus L)[f] = P[f] \setminus f(L)$, if the mapping f is an injective mapping.

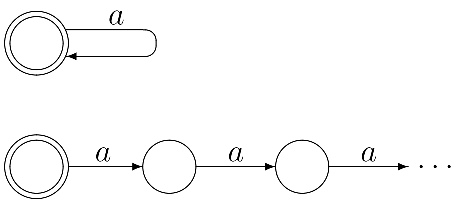
20. $P[f][g] = P[g \circ f]$

Chapter 4

Equivalences of processes

4.1 A concept of an equivalence of processes

The same behavior can be represented by different processes. For example, consider two processes:



The first process has only one state, and the second has infinite set of states, but these processes represent the similar behavior, which consists of a perpetual execution of the actions a.

One of important problems in the theory of processes consists of a finding of an appropriate definition of equivalence of processes, such that processes are equivalent according to this definition if and only if they represent a similar behavior.

In this chapter we present several definitions of equivalence of processes. In every particular situation a choice of an appropriate variant of the concept of equivalence of processes should be determined by a particular understanding (i.e. related to this situation) of a similarity of a behavior of processes.

In sections 4.2 and 4.3 we introduce concepts of trace equivalence and strong equivalence of processes. These concepts are used in situations where all actions executing in the processes that have equal status.

In sections 4.8 and 4.9 we consider other variants of the concept of equivalence of processes: namely, observational equivalence and observational congruence. These concepts are used in situations when we consider the invisible action τ as negligible, i.e. when we assume that two traces are equivalent, if one of them can be obtained from another by insertions and/or deletions of τ.

With each possible definition of equivalence of processes there are related two natural problems.

1. Recognition for two given processes, whether they are equivalent.
2. Construction for a given process P such a process P', which is the least complicated (for example, has a minimum number of states) among all processes that are equivalent to P.

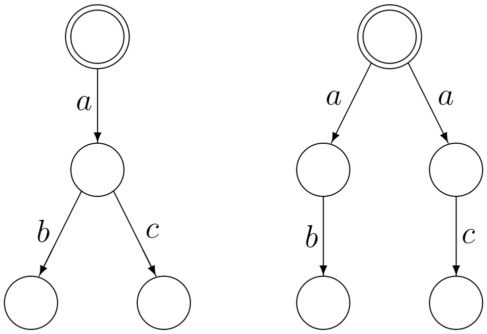
4.2 Trace equivalence of processes

As mentioned above, we would like to consider two processes as equivalent, if they describe a same behavior. So, if we consider a behavior of a process as a generation of a trace, then one of necessary conditions of equivalence of processes P_1 and P_2 is coincidence of sets of their traces:

$$Tr(P_1) = Tr(P_2) \tag{4.1}$$

In some situations, condition (4.1) can be used as a definition of equivalence of P_1 and P_2.

However, the following example shows that this condition does not reflect one important aspect of an execution of processes.



(4.2)

Sets of traces of these processes are equal:

$$Tr(P_1) = Tr(P_2) = \{\varepsilon, a, ab, ac\}$$

(where ε is an empty sequence).

However, these processes have the following essential difference:

- in the left process, after execution of a first action (a) there is a possibility to choose next action (b or c), while
- in the right process, after execution of a first action there is no such possibility:
  - if a first transition occurred on the left edge, then a second action can only be the action b, and
  - if a first transition occurred on the right edge, then a second action can only be the action c

  i.e. a second action was predetermined before execution of a first action.

If we do not wish to consider these processes as equivalent, then condition (4.1) must be enhanced in some a way. One version of such enhancement is described below. In order to formulate it, define the notion of a trace from a state of a process.

Each variant of an execution of a process $P = (S, s^0, R)$ we interpret as a generation of a sequence of transitions

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots \tag{4.3}$$

starting from the initial state s^0 (i.e. $s_0 = s^0$).

We can consider a generation of sequence (4.3) not only from the initial state s^0, but from arbitrary state $s \in S$, i.e. consider a sequence of the form (4.3), in which $s_0 = s$. The sequence $(a_1, a_2, \ldots)$ of labels of these transitions we shall call a **trace starting at** s. A set of all such traces we denote by $Tr_s(P)$.

Let P_1 and P_2 be processes of the form

$$P_i = (S_i, s_i^0, R_i) \qquad (i = 1, 2)$$

Consider a finite sequence of transitions of P_1 of the form

$$s_1^0 = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n \quad (n \geq 0) \tag{4.4}$$

(the case $n = 0$ corresponds to the empty sequence of transitions (4.4), in which $s_n = s_1^0$).

The sequence (4.4) can be considered as an initial phase of execution of the process P_1, and every trace from $Tr_{s_n}(P_1)$ can be considered as a continuation of this phase.

The processes P_1 and P_2 are said to be **trace equivalent**, if

- for each initial phase (4.4) of an execution of the process P_1 there is an initial phase of an execution of the process P_2
$$s_2^0 = s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s'_n \tag{4.5}$$
with the following properties:
  - (4.5) has the same trace $a_1 \ldots a_n$, as (4.4), and
  - at the end of (4.5) there is the same choice of further execution that at the end of (4.4), i.e.
$$Tr_{s_n}(P_1) = Tr_{s'_n}(P_2) \tag{4.6}$$
- and a symmetrical condition holds: for each sequence of transitions of P_2 of the form (4.5) there is a sequence of transitions of P_1 of the form (4.4), such that (4.6) holds.

These conditions have the following disadvantage: they contain

- unlimited sets of sequences of transitions of the form (4.4) and (4.5), and
- unlimited sets of traces from (4.6).

Therefore, checking of these conditions seems to be difficult even when the processes P_1 and P_2 are finite.

There is a problem of finding of necessary and sufficient conditions of trace equivalency, that can be algorithmically checked for given processes P_1 and P_2 in the case when these processes are finite.

Sometimes there is considered an equivalence between processes which is obtained from the trace equivalence by a replacement of condition (4.6) on the weaker condition:
$$Act(s_n) = Act(s'_n)$$
where for each state s $Act(s)$ denotes a set all actions $a \in Act$, such that there is a transition starting at s with the label a.

4.3 Strong equivalence

Another variant of the concept of equivalence of processes is **strong equivalence**. To define the concept of strong equivalence, we introduce auxiliary notations.

After the process

$$P = (S, s^0, R) \tag{4.7}$$

has executed its first action, and turn to a new state s^1, its behavior will be indistinguishable from a behavior of the process

$$P' \overset{\text{def}}{=} (S, s^1, R) \tag{4.8}$$

having the same components as P, except of an initial state.

We shall consider the diagram

$$P \xrightarrow{a} P' \tag{4.9}$$

as an abridged notation of the statement that

- P and P' are processes of the form (4.7), and (4.8) respectively, and
- R contains the transition $s^0 \xrightarrow{a} s^1$.

(4.9) can be interpreted as a statement that the process P can

- execute the action a, and then
- behave like the process P'.

A concept of strong equivalence is based on the following understanding of equivalence of processes: if we consider processes P_1 and P_2 as equivalent, then it must be satisfied the following condition:

- if one of these processes P_i can
  - execute some action $a \in Act$,
  - and then behave like some process P'_i
- then the other process $P_j \quad (j \in \{1, 2\} \setminus \{i\})$ also must be able
  - execute the same action a,

  - and then behave like some process P'_j, which is equivalent to P'_i.

Thus, the desired equivalence must be a a binary relation μ on the set of all processes, the following properties.

(1) If $(P_1, P_2) \in \mu$, and

$$P_1 \xrightarrow{a} P'_1 \tag{4.10}$$

for some process P'_1, then there is a process P'_2, such that

$$P_2 \xrightarrow{a} P'_2 \tag{4.11}$$

and

$$(P'_1, P'_2) \in \mu \tag{4.12}$$

(2) symmetric property: if $(P_1, P_2) \in \mu$, and for some process P'_2 (4.11) holds, then there is a process P'_1, such that (4.10) and (4.12) hold.

Denote by the symbol $\mathcal{M}$ a set of all binary relations, which possess the above properties.

The set $\mathcal{M}$ is nonempty: it contains, for example, a diagonal relation, which consists of all pairs of the form (P, P), where P is an arbitrary process.

The question naturally arises: which of the relations from $\mathcal{M}$ can be used for a definition of strong equivalence?

We suggest the most simple answer to that question: we will consider P_1 and P_2 as strongly equivalent if and only if there exists at least one relation $\mu \in \mathcal{M}$, which contains the pair (P_1, P_2).

Thus, we define the desired relation of strong equivalence on the set of all processes as the union of all relations from $\mathcal{M}$. This relation is denoted by $\sim$.

It is not so difficult to prove that

- $\sim \in \mathcal{M}$, and
- $\sim$ is an equivalence relation, because
  - reflexivity of $\sim$ follows from the fact that the diagonal relation belongs to $\mathcal{M}$,
  - symmetry of $\sim$ follows from the fact that if $\mu \in \mathcal{M}$, then $\mu^{-1} \in \mathcal{M}$
  - transitivity of $\sim$ follows from the fact that if $\mu_1 \in \mathcal{M}$ and $\mu_2 \in \mathcal{M}$, then $\mu_1 \circ \mu_2 \in \mathcal{M}$.

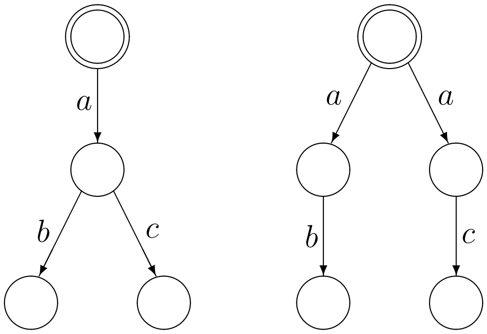
If processes P_1 and P_2 are strongly equivalent, then this fact is denoted by

$$P_1 \sim P_2$$

It is easy to prove that if processes P_1 and P_2 are strongly equivalent they they are trace equivalent.

To illustrate the concept of strong equivalence we give a couple of examples.

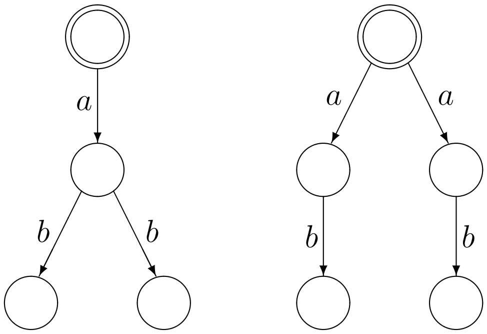
1. The processes



(4.13)

are not strongly equivalent, because they are not trace equivalent.

2. Processes



are strongly equivalent.

4.4 Criteria of strong equivalence

4.4.1 A logical criterion of strong equivalence

Let Fm be a set of **formulas** defined as follows.

- The symbols $\top$ and $\bot$ are formulas from Fm.
- If $\varphi \in Fm$, then $\neg\varphi \in Fm$.
- If $\varphi \in Fm$ and $\psi \in Fm$, then $\varphi \wedge \psi \in Fm$.
- If $\varphi \in Fm$, and $a \in Act$, then $\langle a\rangle\varphi \in Fm$.

Let P be a process, and $\varphi \in Fm$. A **value** of the formula φ on the process P is an element $P(\varphi)$ of the set $\{0,1\}$ defined as follows.

- $P(\top) \overset{\text{def}}{=} 1, \;\; P(\bot) \overset{\text{def}}{=} 0$
- $P(\neg\varphi) \overset{\text{def}}{=} 1 - P(\varphi)$
- $P(\varphi \wedge \psi) \overset{\text{def}}{=} P(\varphi) \cdot P(\psi)$
- $P(\langle a\rangle\varphi) \overset{\text{def}}{=} \begin{cases} 1, & \text{if there is a process } P' : \\ & P \xrightarrow{a} P', \; P'(\varphi) = 1 \\ 0, & \text{otherwise} \end{cases}$

A **theory** of the process P is a subset $Th(P) \subset Fm$, defined as follows:

$$Th(P) = \{\varphi \in Fm \mid P(\varphi) = 1\}$$

Theorem 1.
Let P_1 and P_2 be finite processes. Then

$$P_1 \sim P_2 \quad \Leftrightarrow \quad Th(P_1) = Th(P_2)$$

Proof.
Let $P_1 \sim P_2$. The statement that for each $\varphi \in Fm$ the equality $P_1(\varphi) = P_2(\varphi)$ holds, can be proven by induction on the structure of φ.
Prove the implication "$\Leftarrow$". Suppose that

$$Th(P_1) = Th(P_2) \tag{4.14}$$

Let μ be a binary relation on the set of all processes, defined as follows:

$$\mu \overset{\text{def}}{=} \{(P_1, P_2) \mid Th(P_1) = Th(P_2)\}$$

We prove that μ satisfies the definition of strong equivalence. Let this does not hold, that is, for example, for some $a \in Act$

(a) there is a process P_1', such that

$$P_1 \xrightarrow{a} P_1'$$

(b) but there is no a process P_2', such that

$$P_2 \xrightarrow{a} P_2' \tag{4.15}$$

and $Th(P_1') = Th(P_2')$.

Condition (b) can be satisfied in two situations:

1. There is no a process P_2', such that (4.15) holds.
2. There exists a process P_2', such that (4.15) holds, but for each such process P_2'

$$Th(P_1') \neq Th(P_2')$$

We show that in both these situations there is a formula $\varphi \in Fm$, such that

$$P_1(\varphi) = 1, \quad P_2(\varphi) = 0$$

that would be contrary to assumption (4.14).

1. If the first situation holds, then we can take as φ the formula $\langle a \rangle \top$.
2. Assume that the second situation holds. Let

$$P_{2,1}', \ldots, P_{2,n}'$$

be a list of all processes P_2' satisfying (4.15).

By assumption, for each $i = 1, \ldots, n$, the inequality

$$Th(P_1') \neq Th(P_{2,i}')$$

holds, i.e. for each $i = 1, \ldots, n$ there is a formula φ_i, such that

$$P_1'(\varphi_i) = 1, \qquad P_{2,i}'(\varphi_i) = 0$$

In this situation, we can take as φ the formula $\langle a \rangle (\varphi_1 \wedge \ldots \wedge \varphi_n)$. ■

For example, let P_1 and P_2 be processes (4.13). As stated above, these processes are not strongly equivalent. The following formula can be taken as a justification of the statement that $P_1 \not\sim P_2$:

$$\varphi \stackrel{\text{def}}{=} \langle a\rangle(\langle b\rangle\top \wedge \langle c\rangle\top)$$

It is easy to prove that $P_1(\varphi) = 1$ and $P_2(\varphi) = 0$.

There is a problem of finding for two given processes P_1 and P_2 a list of formulas of a smallest size

$$\varphi_1, \ldots, \varphi_n$$

such that $P_1 \sim P_2$ if and only if

$$\forall\, i = 1, \ldots, n \qquad P_1(\varphi_i) = P_2(\varphi_i)$$

4.4.2 A criterion of strong equivalence, based on the notion of a bisimulation

Theorem 2.

Let P_1 and P_2 be a couple of processes of the form

$$P_i = (S_i, s_i^0, R_i) \qquad (i = 1, 2)$$

Then $P_1 \sim P_2$ if and only if there is a relation

$$\mu \subseteq S_1 \times S_2$$

satisfying the following conditions.

0. $(s_1^0, s_2^0) \in \mu$.

1. For each pair $(s_1, s_2) \in \mu$ and each transition from R_1 of the form

$$s_1 \xrightarrow{a} s_1'$$

   there is a transition from R_2 of the form

$$s_2 \xrightarrow{a} s_2'$$

   such that $(s_1', s_2') \in \mu$.

2. For each pair $(s_1, s_2) \in \mu$ and each transition from R_2 of the form

$$s_2 \xrightarrow{a} s_2'$$

there is a transition from R_1 of the form

$$s_1 \xrightarrow{a} s_1'$$

such that $(s_1', s_2') \in \mu$.

A relation μ, satisfying these conditions, is called a **bisimulation (BS)** between P_1 and P_2.

4.5 Algebraic properties of strong equivalence

Theorem 3.

Strong equivalence is a congruence, i.e., if $P_1 \sim P_2$, then

- for each $a \in Act \quad a.P_1 \sim a.P_2$
- for each process $P \quad P_1 + P \sim P_2 + P$
- for each process $P \quad P_1|P \sim P_2|P$
- for each $L \subseteq Names \quad P_1 \setminus L \sim P_2 \setminus L$
- for each renaming $f \quad P_1[f] \sim P_2[f]$

Proof.

As it was stated in section 4.4.2, the statement

$$P_1 \sim P_2$$

is equivalent to the statement that there is a BS μ between P_1 and P_2. Using this μ, we construct a BS for justification of each of the foregoing relationships.

- Let $s^0_{(1)}$ and $s^0_{(2)}$ be initial states of the processes $a.P_1$ and $a.P_2$ respectively.

  Then the relation
  $$\{(s^0_{(1)}, s^0_{(2)})\} \ \cup \ \mu$$
  is a BS between $a.P_1$ and $a.P_2$.

- Let
  - $s^0_{(1)}$ and $s^0_{(2)}$ be initial states of $P_1 + P$ and $P_2 + P$ respectively, and
  - S be a set of states of the process P.

  Then
  - the relation
  $$\{(s^0_{(1)}, s^0_{(2)})\} \;\cup\; \mu \;\cup\; Id_S$$
  is a BS between $P_1 + P$ and $P_2 + P$, and
  - the relation
  $$\{((s_1, s), (s_2, s)) \mid (s_1, s_2) \in \mu,\; q \in S\}$$
  is a BS between $P_1|P$ and $P_2|P$.
- The relation μ is a BS
  - between $P_1 \setminus L$ and $P_2 \setminus L$, and
  - between $P_1[f]$ and $P_2[f]$. ■

Theorem 4.
Each process $P = (S, s^0, R)$ has the following properties.

1. $P + \mathbf{0} \sim P$
2. $P + P \sim P$

Proof.

1. Let s^0_0 be an initial state of the process $P + \mathbf{0}$.

   Then the relation
   $$\{(s^0_0, s^0)\} \;\cup\; Id_S$$
   is a BS between $P + \mathbf{0}$ and P.

2. By definition of the operation "+", processes in the left side of the statement $P + P \sim P$ should be considered as two disjoint isomorphic copies of P of the form

$$P_{(i)} = (S_{(i)}, s^0_{(i)}, R_{(i)}) \quad (i = 1, 2)$$

   where $S_{(i)} = \{s_{(i)} \mid s \in S\}$.

   Let s^0_0 be an initial state of the process $P + P$.

   Then the relation

$$\{(s^0_0, s^0)\} \,\cup\, \{(s_{(i)}, s) \mid s \in S,\ i = 1, 2\}$$

   is a BS between $P + P$ and P. ■

Below for

- each process $P = (S, s^0, R)$, and
- each state $s \in S$

we denote by $P(s)$ the process (S, s, R), which is obtained from P by a replacement of an initial state.

Theorem 5.

Let $P = (S, s^0, R)$ be a process, and a set of all its transitions, starting from s^0, has the form

$$\{\, s^0 \xrightarrow{a_i} s^i \mid i = 1, \ldots, n\}$$

Then

$$P \;\sim\; a_1.P_1 + \ldots + a_n.P_n \tag{4.16}$$

where for each $i = 1, \ldots, n$

$$P_i \overset{\text{def}}{=} P(s^i) \overset{\text{def}}{=} (S, s^i, R)$$

Proof.

(4.16) holds because there is a BS between left and right sides of (4.16).

For a construction of this BS we replace all the processes P_i in the right side of (4.16) on their disjoint copies, i.e. we can consider that for each $i = 1, \ldots, n$

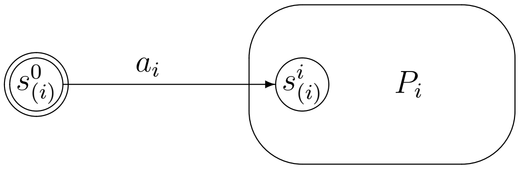
- the process P_i has the form

$$P_i = (S_{(i)}, s^i_{(i)}, R_{(i)})$$

  where all the sets $S_{(1)}, \ldots, S_{(n)}$ are disjoint, and

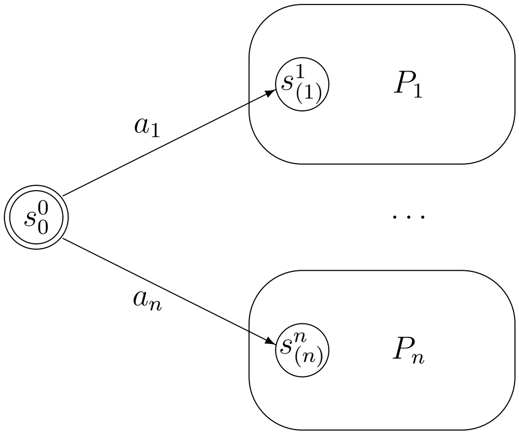
- a corresponding bijection between S and $S_{(i)}$ maps each state $s \in S$ to a state, denoted by the symbol $s_{(i)}$.

Thus, we can assume that each summand $a_i.P_i$ in the right side of (4.16) has the form



and sets of states of these summands are pairwise disjoint.

According to the definition of the operation +, the right side of (4.16) has the form



BS between left and right sides of (4.16) has be defined, for example, as the relation

$$\{(s^0, s^0_0)\} \;\cup\; \{(s, s_{(i)}) \mid s \in S,\; i = 1, \ldots, n\} \quad \blacksquare$$

Theorem 6 (expansion theorem).

Let P be a process of the form

$$P \;=\; P_1 \,|\, \ldots \,|\, P_n \tag{4.17}$$

where for each $i \in \{1, \dots, n\}$ the process P_i has the form

$$P_i = \sum_{j=1}^{n_i} a_{ij}.\, P_{ij} \tag{4.18}$$

Then P is strongly equivalent to a sum of

1. all processes of the form

$$a_{ij}.\, \Big(P_1 \,|\, \dots \,|\, P_{i-1} \,|\, P_{ij} \,|\, P_{i+1} \,|\, \dots \,|\, P_n\Big) \tag{4.19}$$

2. and all processes of the form

$$\tau. \left(\begin{array}{c} P_1 \,|\, \dots \,|\, P_{i-1} \,|\, P_{ik} \,|\, P_{i+1} \,|\, \dots \\ \dots \,|\, P_{j-1} \,|\, P_{jl} \,|\, P_{j+1} \,|\, \dots \,|\, P_n \end{array} \right) \tag{4.20}$$

where $1 \leq i < j \leq n, \quad a_{ik}, a_{jl} \neq \tau$, and $a_{ik} = \overline{a_{jl}}$.

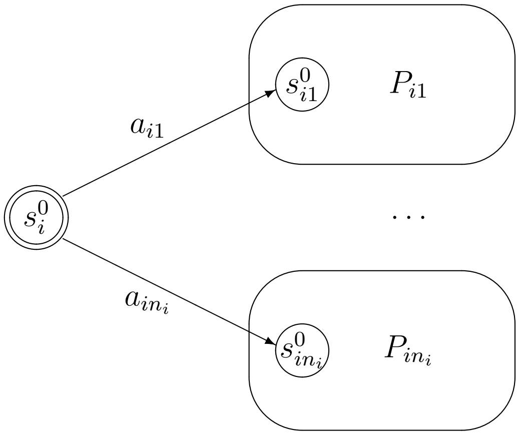
Proof.

By theorem 5, P is strongly equivalent to a sum, each summand of which corresponds to a transition starting from the initial state s^0 of the process P. For each transition of P of the form

$$s^0 \xrightarrow{a} s$$

this sum contains the summand $a.P(s)$.

According to (4.18), for each $i = 1, \dots, n$ the process P_i has the form



where $s_i^0, s_{i1}^0, \ldots, s_{in_i}^0$ are initial states of the processes

$$P_i, \;\; P_{i1}, \;\; \ldots, \;\; P_{in_i}$$

respectively.

Let

- S_i be a set of states of the process P_i, and
- S_{ij} (where $j = 1, \ldots, n_i$) be a set of states of the process P_{ij}.

We can assume that S_i is a disjoint union of the form

$$S_i = \{s_i^0\} \;\cup\; S_{i1} \;\cup\; \ldots \;\cup\; S_{in_i} \tag{4.21}$$

According to the description of a process of the form (4.17), which is presented in item 2 of section 3.7, we can assume that components of P have the following form.

- A set of states of the process P has the form
$$S_1 \times \ldots \times S_n \tag{4.22}$$
- An initial state s^0 of P is a list
$$(s_1^0, \ldots, s_n^0)$$
- Transitions of P, starting from its initial state, are as follows.
    - Transitions of the form
    $$s^0 \xrightarrow{a_{ij}} (s_1^0, \ldots, s_{i-1}^0, s_{ij}^0, s_{i+1}^0, \ldots, s_n^0) \tag{4.23}$$
    - Transitions of the form
    $$s^0 \xrightarrow{\tau} \left(\begin{array}{c} s_1^0, \ldots, s_{i-1}^0, s_{ik}^0, s_{i+1}^0, \ldots \\ \ldots s_{j-1}^0, s_{jl}^0, s_{j+1}^0, \ldots, s_n^0 \end{array} \right) \tag{4.24}$$
    where $1 \leq i < j \leq n, \quad a_{ik}, a_{jl} \neq \tau$, and $a_{ik} = \overline{a_{jl}}$.

Thus, there is an one-to-one correspondence between

- the set of transitions of the process P, starting from s^0, and

- the set of summands of the form (4.19) and (4.20).

For the proof of theorem 6 it is enough to prove that

- For each $i = 1, \ldots, n$, and each $j = 1, \ldots, n_i$ the following equivalence holds:
$$\begin{array}{l} P(s_1^0, \ldots, s_{i-1}^0, s_{ij}^0, s_{i+1}^0, \ldots, s_n^0) \sim \\ \sim \Big(P_1 \,|\, \ldots \,|\, P_{i-1} \,|\, P_{ij} \,|\, P_{i+1} \,|\, \ldots \,|\, P_n\Big) \end{array} \tag{4.25}$$

- for
  - any i, j, such that $1 \leq i < j \leq n$, and
  - any $k = 1, \ldots, n_i, \;\; l = 1, \ldots, n_j$

  the following equivalence holds:
$$\begin{array}{l} P\left(\begin{array}{c} s_1^0, \ldots, s_{i-1}^0, s_{ik}^0, s_{i+1}^0, \ldots \\ \ldots s_{j-1}^0, s_{jl}^0, s_{j+1}^0, \ldots, s_n^0 \end{array}\right) \sim \\ \sim \left(\begin{array}{c} P_1 \,|\, \ldots \,|\, P_{i-1} \,|\, P_{ik} \,|\, P_{i+1} \,|\, \ldots \\ \ldots \,|\, P_{j-1} \,|\, P_{jl} \,|\, P_{j+1} \,|\, \ldots \,|\, P_n \end{array}\right) \end{array} \tag{4.26}$$

We shall prove only (4.25) ((4.26) can be proven similarly).

A set of states of the process
$$\Big(P_1 \,|\, \ldots \,|\, P_{i-1} \,|\, P_{ij} \,|\, P_{i+1} \,|\, \ldots \,|\, P_n\Big) \tag{4.27}$$
has the form
$$S_1 \times \ldots \times S_{i-1} \times S_{ij} \times S_{i+1} \times \ldots \times S_n \tag{4.28}$$

(4.21) implies that $S_{ij} \subseteq S_i$, i.e. set (4.28) is a subset of set (4.22) of states of the process
$$P(s_1^0, \ldots, s_{i-1}^0, s_{ij}^0, s_{i+1}^0, \ldots, s_n^0) \tag{4.29}$$

We define the desired BS μ between processes (4.27) and (4.29) as the diagonal relation
$$\mu \overset{\text{def}}{=} \{(s, s) \mid s \in (4.28)\}$$
Obviously,

- a pair of initial states of processes (4.27) and (4.29) belongs to μ,

- each transition of the process (4.27) is also a transition of the process (4.29), and
- if a start of some transition of the process (4.29) belongs to the subset (4.28), then the end of this transition also belongs to the subset (4.28) (to substantiate this claim we note that for each transition of P_i, if its start belongs to S_{ij}, then its end also belongs to S_{ij}).

Thus, μ is a BS, and this proves the claim (4.25). ■

The following theorem is a strengthening of theorem 6. To formulate it, we will use the following

notation. If $f : Names \to Names$ is a renaming, then the symbol f denotes also a mapping of the form

$$f : Act \to Act$$

defined as follows.

- $\forall\, \alpha \in Names \quad f(\alpha!) \overset{\text{def}}{=} f(\alpha)!,\ f(\alpha?) \overset{\text{def}}{=} f(\alpha)?$
- $f(\tau) \overset{\text{def}}{=} \tau$

Theorem 7.

Let P be a process of the form

$$P \;=\; \Big(P_1[f_1] \,|\, \dots \,|\, P_n[f_n]\Big) \setminus L$$

where for each $i \in \{1, \dots, n\}$

$$P_i \sim \sum_{j=1}^{n_i} a_{ij}.\, P_{ij}$$

Then P is strongly equivalent to a sum of

1. all processes of the form

$$f_i(a_{ij}).\left(\left(\begin{array}{l} P_1[f_1] \,|\, \dots \\ \dots \,|\, P_{i-1}[f_{i-1}] \,|\, P_{ij}[f_i] \,|\, P_{i+1}[f_{i+1}] \,|\, \dots \\ \dots \,|\, P_n[f_n] \end{array} \right) \setminus L \right)$$

where $a_{ij} = \tau$ or $name(f_i(a_{ij})) \notin L$, and

2. all processes of the form

$$\tau.\left(\left(\begin{array}{l} P_1[f_1] \,|\, \ldots \\ \ldots \,|\, P_{i-1}[f_{i-1}] \,|\, P_{ik}[f_i] \,|\, P_{i+1}[f_{i+1}] \,|\, \ldots \\ \ldots \,|\, P_{j-1}[f_{j-1}] \,|\, P_{jl}[f_j] \,|\, P_{j+1}[f_{j+1}] \,|\, \ldots \\ \ldots \,|\, P_n[f_n] \end{array}\right) \setminus L\right)$$

where $1 \leq i < j \leq n, \quad a_{ik}, a_{jl} \neq \tau$, and $f_i(a_{ik}) = \overline{f_j(a_{jl})}$.

Proof.
This theorem follows directly from

- the previous theorem,
- theorem 3,
- properties 6, 9, 10, 16 and 17 from section 3.7, and
- the first assertion from theorem 4.

4.6 Recognition of strong equivalence

4.6.1 Relation $\mu(P_1, P_2)$

Let P_1, P_2 be a couple of processes of the form

$$P_i = (S_i, s_i^0, R_i) \qquad (i = 1, 2)$$

Define an operator $'$ on the set of all relations from S_1 to S_2, that maps each relation $\mu \subseteq S_1 \times S_2$ to the relation $\mu' \subseteq S_1 \times S_2$, defined as follows:

$$\mu' \stackrel{\text{def}}{=} \left\{ \begin{array}{l|l} & \forall a \in Act \\ & \forall s_1' \in S_1 : (s_1 \xrightarrow{a} s_1') \in R_1 \\ & \quad \exists s_2' \in S_2 : \left\{ \begin{array}{l} (s_2 \xrightarrow{a} s_2') \in R_2 \\ (s_1', s_2') \in \mu \end{array} \right. \\ (s_1, s_2) \in & \\ \in S_1 \times S_2 & \forall s_2' \in S_2 : (s_2 \xrightarrow{a} s_2') \in R_2 \\ & \quad \exists s_1' \in S_1 : \left\{ \begin{array}{l} (s_1 \xrightarrow{a} s_1') \in R_1 \\ (s_1', s_2') \in \mu \end{array} \right. \end{array} \right\}$$

It is easy to prove that for each $\mu \subseteq S_1 \times S_2$

$$\begin{array}{l}\mu \text{ satisfies conditions 1 and 2} \\ \text{from the definition of a BS}\end{array} \quad \Leftrightarrow \quad \mu \subseteq \mu'$$

Consequently,

$$\mu \text{ is a BS between } P_1 \text{ and } P_2 \quad \Leftrightarrow \quad \left\{ \begin{array}{l} (s_1^0, s_2^0) \in \mu \\ \mu \subseteq \mu' \end{array} \right.$$

It is easy to prove that the operator $'$ is monotone, i.e.

$$\text{if } \mu_1 \subseteq \mu_2, \text{ then } \mu_1' \subseteq \mu_2'.$$

Let μ_{max} be a union of all relations from the set

$$\{\mu \subseteq S_1 \times S_2 \mid \mu \subseteq \mu'\} \tag{4.30}$$

Note that the relation μ_{max} belongs to the set (4.30), since for every $\mu \in (4.30)$ from

- the inclusion $\mu \subseteq (\bigcup\limits_{\mu \in (4.30)} \mu) = \mu_{max}$, and
- monotonicity of $'$

it follows that for each $\mu \in (4.30)$

$$\mu \subseteq \mu' \subseteq \mu'_{max}$$

So $\mu_{max} = \bigcup\limits_{\mu \in (4.30)} \mu \subseteq \mu'_{max}$, i.e. $\mu_{max} \in (4.30)$.

Note that the following equality holds

$$\mu_{max} = \mu'_{max}$$

because

- the inclusion $\mu_{max} \subseteq \mu'_{max}$, and
- monotonicity of $'$

imply the inclusion

$$\mu'_{max} \subseteq \mu''_{max}$$

i.e. $\mu'_{max} \in$ (4.30), whence, by virtue of maximality of μ_{max}, we get the inclusion

$$\mu'_{max} \subseteq \mu_{max}$$

Thus, the relation μ_{max} is

- a greatest element of the partially ordered set (4.30) (where a partial order is the relation of inclusion), and
- a greatest fixed point of the operator $'$.

We shall denote this relation by

$$\mu(P_1, P_2) \tag{4.31}$$

From theorem 2 it follows that

$$P_1 \sim P_2 \quad \Leftrightarrow \quad (s_1^0, s_2^0) \in \mu(P_1, P_2)$$

From the definition of the relation $\mu(P_1, P_2)$ it follows that this relation consists of all pairs $(s_1, s_2) \in S_1 \times S_2$, such that

$$P_1(s_1) \sim P_2(s_2)$$

The relation $\mu(P_1, P_2)$ can be considered as a **similarity measure** between P_1 and P_2.

4.6.2 A polynomial algorithm for recognizing of strong equivalence

Let P_1 and P_2 be processes of the form

$$P_i = (S_i, s_i^0, R_i) \qquad (i = 1, 2)$$

If the sets S_1 and S_2 are finite, then the problem of checking of statement

$$P_1 \sim P_2 \tag{4.32}$$

obviously is algorithmically solvable: for example, you can iterate over all relations $\mu \subseteq S_1 \times S_2$ and for each of them verify conditions 0, 1 and 2 from the definition of BS. The algorithm finishes its work when

- it is found a relation $\mu \subseteq S_1 \times S_2$ which satisfies conditions of 0, 1 and 2 from the definition of BS, in this case the algorithm gives the answer

$$P_1 \sim P_2$$

or

- all relations $\mu \subseteq S_1 \times S_2$ are checked, and none of them satisfy conditions of 0, 1 and 2 from the definition of BS. In this case, the algorithm gives the answer
$$P_1 \not\sim P_2$$

If $P_1 \not\sim P_2$, then the above algorithm will give the answer after checking of all relations from S_1 to S_2, the number of which is

$$2^{|S_1|\cdot|S_2|}$$

(where for every finite set S we denote by $|S|$ a number of elements of S), i.e. this algorithm has exponential complexity.

The problem of checking $P_1 \sim P_2$ can be solved by more efficient

algorithm, which has polynomial complexity. To construct such an algorithm, we consider the following sequence of relations from S_1 to S_2:

$$\{\mu_i \mid i \geq 1\} \tag{4.33}$$

where $\mu_1 \overset{\text{def}}{=} S_1 \times S_2$, and $\forall\, i \geq 1 \quad \mu_{i+1} \overset{\text{def}}{=} \mu_i'$.

From

- the inclusion $\mu_1 \supseteq \mu_2$, and

- the monotonicity of the operator $'$

it follows that
$$\begin{array}{l} \mu_2 = \mu_1' \supseteq \mu_2' = \mu_3 \\ \mu_3 = \mu_2' \supseteq \mu_3' = \mu_4 \\ \text{etc.} \end{array}$$

Thus, the sequence (4.33) is monotone:

$$\mu_1 \supseteq \mu_2 \supseteq \ldots$$

Since all members of sequence (4.33) are subsets of the finite set $S_1 \times S_2$, then this sequence can not decrease infinitely, it will be stabilized at some member, i.e. there is an index $i \geq 1$, such tha

$$\mu_i = \mu_{i+1} = \mu_{i+2} = \ldots$$

We prove that the relation μ_i (where i is the above index) coincides with the relation $\mu(P_1, P_2)$.

- Since $\mu_i = \mu_{i+1} = \mu_i'$, i.e. μ_i is a fixed point of the operator $'$, then

  $$\mu_i \subseteq \mu(P_1, P_2) \tag{4.34}$$

  since $\mu(P_1, P_2)$ is the largest fixed point of the operator $'$.

- For each $j \geq 1$ the inclusion

  $$\mu(P_1, P_2) \subseteq \mu_j \tag{4.35}$$

  holds, because

  - inclusion (4.35) holds for $j = 1$, and
  - if inclusion (4.35) holds for some j, then on the reason of monotonicity of the operator $'$, the following equalities hold:

    $$\mu(P_1, P_2) = \mu(P_1, P_2)' \subseteq \mu_j' = \mu_{j+1}$$

    i.e. inclusion (4.35) holds for $j + 1$.

  In particular, (4.35) holds for $j = i$.

The equality

$$\mu_i = \mu(P_1, P_2) \tag{4.36}$$

follows from (4.34) and (4.35) for $j = i$.

Thus, the problem of checking of the statement $P_1 \sim P_2$ can be solved by

- finding a first member μ_i of sequence (4.33), which satisfies the condition $\mu_i = \mu_{i+1}$, and
- checking the condition

  $$(s_1^0, s_2^0) \in \mu_i \tag{4.37}$$

The algorithm gives the answer

$$P_1 \sim P_2$$

if and only if (4.37) holds.

For a calculation of terms of the sequence (4.33) the following algorithm can be used. This algorithm computes a relation μ' for a given relation $\mu \subseteq S_1 \times S_2$.

```
$\mu' := \emptyset$
loop for each $(s_1, s_2) \in \mu$
    include $:= \top$
    loop for each $s'_1, a :\ s_1 \xrightarrow{a} s'_1$
        found $:= \bot$
        loop for each $s'_2 :\ s_2 \xrightarrow{a} s'_2$
            found $:=$ found $\vee (s'_1, s'_2) \in \mu$
        end of loop
        include $:=$ include $\wedge$ found
    end of loop
    loop for each $s'_2, a :\ s_2 \xrightarrow{a} s'_2$
        found $:= \bot$
        loop for each $s'_1 :\ s_1 \xrightarrow{a} s'_1$
            found $:=$ found $\vee (s'_1, s'_2) \in \mu$
        end of loop
        include $:=$ include $\wedge$ found
    end of loop
    if include then $\mu' := \mu' \cup \{(s_1, s_2)\}$
end of loop
```

Note that this algorithm is correct only when $\mu' \subseteq \mu$ (which occurs in the case when this algorithm is used to calculate terms of the sequence (4.33)). In a general situation the outer loop must have the form

$$\textbf{loop for each}\ \ (s_1, s_2) \in S_1 \times S_2$$

Estimate a complexity of the algorithm.

Let A be the number

$$\max(|Act(P_1)|, |Act(P_2)|) + 1$$

- The outer loop does no more than $|S_1| \cdot |S_2|$ iterations.
- Both loops contained in the external loop make max $|S_1| \cdot |S_2| \cdot A$ iterations.

Therefore, a complexity of this algorithm can be evaluated as

$$O(|S_1|^2 \cdot |S_2|^2 \cdot A)$$

Since for a calculation of a member μ_i of sequence (4.33), on which (4.33) is stabilized, we must calculate not more than $|S_1| \cdot |S_2|$ members of this sequence, then, consequently, the desired relation $\mu_i = \mu(P_1, P_2)$ can be calculated during

$$O(|S_1|^3 \cdot |S_2|^3 \cdot A)$$

4.7 Minimization of processes

4.7.1 Properties of relations of the form $\mu(P, P)$

Theorem 8.

For each process $P \stackrel{\text{def}}{=} (S, s^0, R)$ the relation $\mu(P, P)$ is an equivalence.

Proof.

1. **Reflexivity** of the relation $\mu(P, P)$ follows from the fact that the diagonal relation
$$Id_S = \{(s, s) \mid s \in S\}$$
satisfy conditions 1 and 2 from the definition of BS, i.e.
$$Id_S \in (4.30).$$
2. **Symmetry** of the relation $\mu(P, P)$ follows from the fact that if a relation μ satisfies conditions 1 and 2 from the definition of BS, then the inverse relation μ^{-1} also satisfies these conditions, that is,
$$\text{if } \mu \in (4.30), \text{ then } \mu^{-1} \in (4.30).$$

3. **Transitivity** of the relation $\mu(P,P)$ follows from the fact that the product
$$\mu(P,P) \circ \mu(P,P)$$
satisfies conditions 1 and 2 from the definition of BS, i.e.
$$\mu(P,P) \circ \mu(P,P) \subseteq \mu(P,P) \qquad \blacksquare$$

Let $P_\sim$ be a process, whose components have the following form.

- Its states are equivalence classes of the set S of states of P, corresponding to the equivalence $\mu(P,P)$.
- Its initial state is the class $[s^0]$, which contains the initial state s^0 of P.
- A set of its transitions consists of all transitions of the form
$$[s_1] \xrightarrow{a} [s_2]$$
where $s_1 \xrightarrow{a} s_2$ is an arbitrary transition from R.

The process $P_\sim$ is said to be a **factor-process** of the process P with respect to the equivalence $\mu(P,P)$.

Theorem 9.
For each process P the relation
$$\mu \overset{\text{def}}{=} \{\ (s,[s])\ \mid s \in S\}$$
is BS between P and $P_\sim$.

Proof.
Check the properties 0, 1, 2 from the definition of BS for the relation μ.
Property 0 holds by definition of an initial state of the process $P_\sim$.
Property 1 holds by definition of a set of transitions of $P_\sim$.
Let us prove property 2. Let $P_\sim$ contains a transition
$$[s] \xrightarrow{a} [s']$$
Prove that there is a transition in R of the form
$$s \xrightarrow{a} s''$$

such that $(s'', [s']) \in \mu$, i.e. $[s''] = [s']$, i.e.

$$(s'', s') \in \mu(P, P)$$

From the definition of a set of transitions of the process $P_\sim$ it follows that R contains a transition of the form

$$s_1 \xrightarrow{a} s_1' \tag{4.38}$$

where $[s_1] = [s]$ and $[s_1'] = [s']$, i.e.

$$\begin{array}{ll} (s_1, s) \in \mu(P, P) & \text{and} \\ (s_1', s') \in \mu(P, P) & \end{array}$$

Since $\mu(P, P)$ is a BS, then from

- (4.38) $\in R$, and
- $(s_1, s) \in \mu(P, P)$

it follows that R contains a transition of the form

$$s \xrightarrow{a} s_1'' \tag{4.39}$$

where $(s_1'', s_1') \in \mu(P, P)$.

Since $\mu(P, P)$ is transitive, then from

$$\begin{array}{ll} (s_1'', s_1') \in \mu(P, P) & \text{and} \\ (s_1', s') \in \mu(P, P) & \end{array}$$

it follows that

$$(s_1'', s') \in \mu(P, P)$$

Thus, as the desired state s'' it can taken the state s_1''. ■

From theorem 9 it follows that for each process P

$$P \sim P_\sim$$

4.7.2 Minimal processes with respect to $\sim$

A process P is said to be **minimal with respect to** $\sim$, if

- each its state is reachable, and
- $\mu(P, P) = Id_S$
  (where S is a set of states of P).

Below minimal processes with respect to $\sim$ are called simply **minimal processes**.

Theorem 10.
Let the processes P_1 and P_2 minimal, and $P_1 \sim P_2$.
Then P_1 and P_2 are isomorphic.

Proof.
Suppose that P_i $(i = 1, 2)$ has the form (S_i, s_i^0, R_i), and let $\mu \subseteq S_1 \times S_2$ be BS between P_1 and P_2.
Since $\mu^(- 1)$ is also BS, and composition of BSs is BS, then

- $\mu \circ \mu^{-1}$ is BS between P_1

  and P_1 , and
- $\mu^{-1} \circ \mu$ is BS between P_2 and P_2

whence, using definition of the relations $\mu(P_i, P_i)$, and the definition of a minimal process, we get the inclusions

$$\begin{array}{l} \mu \circ \mu^{-1} \subseteq \mu(P_1, P_1) = Id_{S_1} \\ \mu^{-1} \circ \mu \subseteq \mu(P_2, P_2) = Id_{S_2} \end{array} \tag{4.40}$$

Prove that the relation μ is functional, i.e. for each $s \in S_1$ there is a unique element $s' \in S_2$, such that $(s, s') \in \mu$.

- If $s = s_1^0$, then we define $s' \overset{\text{def}}{=} s_2^0$.
- If $s \neq s_1^0$ then, since every state in P_1 is reachable, then there is a path in P_1 of the form
$$s_1^0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s$$

Since μ is BS, then there is a path in P_2 of the form

$$s_2^0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s'$$

and $(s, s') \in \mu$.

Thus, in both cases there is an element $s' \in S_2$, such that $(s, s') \in \mu$.

Let us prove the uniqueness of the element s' with the property $(s, s') \in \mu$.

If there is an element $s'' \in S_2$, such that $(s, s'') \in \mu$, then $(s'', s) \in \mu^{-1}$, which implies

$$(s'', s') \in \mu^{-1} \circ \mu = Id_{S_2}$$

so $s'' = s'$.

For similar reasons, the relation μ^{-1} is also functional.

From conditions (4.40) it is easy to deduce bijectivity of the mapping, which corresponds to the relation μ. By the definition of BS, this implies that P_1 and P_2 are isomorphic. ■

Theorem 11.
Let

- a process P_2 is obtained from a process P_1 by removing of unreachable states, and
- $P_3 \overset{\text{def}}{=} (P_2)_\sim$.

Then the process P_3 is minimal, and

$$P_1 \sim P_2 \sim P_3$$

Proof.
Since each state of P_2 is reachable, then from the definition of transitions of a factor-process, it follows that each state of P_3 is also achievable.

Now, we prove that

$$\mu(P_3, P_3) = Id_{S_3} \tag{4.41}$$

i.e. suppose that $(s', s'') \in \mu(P_3, P_3)$, and prove that $s' = s''$.

From the definition of a factor-process it follows that there are states $s_1, s_2 \in S_2$, such that

$$\begin{array}{rcl} s' & = & [s_1] \\ s'' & = & [s_2] \end{array}$$

where $[\cdot]$ denotes an equivalence class with respect to $\mu(P_2, P_2)$.

From theorem 9 it follows that

$$\begin{array}{rcl} (s_1, s') & \in & \mu(P_2, P_3) \\ (s'', s_2) & \in & \mu(P_3, P_2) \end{array}$$

Since a composition of BSs is also BS, then the composition

$$\mu(P_2, P_3) \circ \mu(P_3, P_3) \circ \mu(P_3, P_2) \tag{4.42}$$

is BS between P_2 and P_2, so

$$(4.42) \subseteq \mu(P_2, P_2) \tag{4.43}$$

Since $(s_1, s_2) \in (4.42)$, then, in view of (4.43), we get:

$$s' = [s_1] = [s_2] = s''$$

In conclusion, we note that

- the statement $P_1 \sim P_2$ is obvious, and
- the statement $P_2 \sim P_3$ follows from theorem 9. ■

4.7.3 An algorithm for minimizing of finite processes

The algorithm described in section 4.6.2 can be used to solve the problem of **minimizing of finite processes**, which has the following form: for a given finite process P build a process Q with the smallest number of states, which is strongly equivalent to P.

To build the process Q, first there is constructed a process P', obtained from P by removing of unreachable states. The process Q has the form $P'_\sim$.

A set of states of the process P' can be constructed as follows. Let P has the form

$$P = (S, s^0, R)$$

Consider the sequence of subsets of the set S

$$S_0 \subseteq S_1 \subseteq S_2 \subseteq \ldots \tag{4.44}$$

defined as follows.

- $S_0 \overset{\text{def}}{=} \{s^0\}$
- for each $i \geq 0$ the set S_{i+1} is obtained from S_i by adding all states $s' \in S$, such that

$$\exists s \in S,\ \exists a \in Act: \ (s \xrightarrow{\quad a \quad} s') \ \in R$$

Since S is finite, then the sequence (4.44) can not increase infinitely. Let S_i be a member of the sequence (4.44), where this sequence is stabilized. It is obvious that

- all states from S_i are reachable, and
- all states from $S \setminus S_i$ are unreachable.

Therefore, a set of states of the process P' is the set S_i.

Let S' be a set of states of the process P'.

Note that for a computation of the relation $\mu(P', P')$ it is necessary to calculate no more than $|S'|$ members of sequence (4.33), because

- each relation in the sequence (4.33) is an equivalence (since if a binary relation μ on the set of states of a process is an equivalence, then the relation μ' is also an equivalence), and
- - each member of the sequence (4.33) defines a partitioning of the set S', and
  - for each $i \geq 1$, if $\mu_{i+1} \neq \mu_i$, then a partitioning corresponding to μ_{i+1} is a refinement of a partitioning corresponding to μ_i,

  and it is easy to show that a number of such refinements is no more than $|S'|$.

Theorem 12.

The process $P'_{\sim}$ has the smallest number of states among all finite processes that are strongly equivalent to P.

Proof.

Let

- P_1 be a finite process, such that $P_1 \sim P$, and
- P'_1 be a reachable part of P_1.

As it was established above,

$$P_1 \sim P_1' \sim (P_1')_\sim$$

Since $P \sim P' \sim P'_\sim$ and $P \sim P_1$, then, consequently,

$$P'_\sim \sim (P_1')_\sim \tag{4.45}$$

As it was proved in theorem 11, the processes $P'_\sim$ and $(P_1')_\sim$ are minimal. From this and from (4.45), by virtue of theorem 10 we get that the processes $P'_\sim$ and $(P_1')_\sim$ are isomorphic. In particular, they have same number of states.

Since

- a number of states of the process $(P_1')_\sim$ does not exceed a number of states of the process P_1' (since states of the process $(P_1')_\sim$ are classes of a partitioning of the set of states of the process P_1'), and
- a number of states the process P_1' does not exceed a number of states of the process P_1 (since a set of states of the process P_1' is a subset of a set of states of the process P_1)

then, consequently, a number of states of the process $P'_\sim$ does not exceed a number of states of the process P_1. ■

4.8 Observational equivalence

4.8.1 Definition of observational equivalence

Another variant of the concept of equivalence of processes is **observational equivalence**. This concept is used in those situations where we consider the internal action τ as negligible, and consider two traces as the same, if one of them can be obtained from another by insertions and/or deletions of internal actions τ.

For a definition of the concept of observable equivalence we introduce auxiliary notations.

Let P and P' be processes.

1. The notation

$$P \xrightarrow{\tau^*} P' \tag{4.46}$$

means that

- either $P = P'$
- or there is a sequence of processes

  $$P_1, \dots, P_n \quad (n \geq 2)$$

  such that

  - $P_1 = P, \quad P_n = P'$
  - for each $i = 1, \dots, n-1$

  $$P_i \xrightarrow{\tau} P_{i+1}$$

(4.46) can be interpreted as the statement that the process P may imperceptibly turn into a process P'.

2. For every action $a \in Act \setminus \{\tau\}$ the notation

$$P \xrightarrow{a_\tau} P' \tag{4.47}$$

means that there are processes P_1 and P_2 with the following properties:

$$P \xrightarrow{\tau^*} P_1\,, \quad P_1 \xrightarrow{a} P_2\,, \quad P_2 \xrightarrow{\tau^*} P'$$

(4.47) can be interpreted as the statementthat the process P may

- execute a sequence of actions, such that
  - the action a belongs to this sequence, and
  - all other actions in this sequence are internal

  and then
- turn into a process P'.

If (4.47) holds, then we say that the process P may

- **observably execute** the action a, and then
- turn into a process P'.

The concept of observational equivalence is based on the following understanding of equivalence of processes: if we consider processes P_1 and P_2 as equivalent, then they must satisfy the following conditions.

1.
- If one of these processes P_i may imperceptibly turn into some process P'_i,
- then another process $P_j \quad (j \in \{1,2\} \setminus \{i\})$ also must be able imperceptibly turn into some process P'_j, which is equivalent to P'_i.

2.
- If one of these processes P_i may
  - observable execute some action $a \in Act \setminus \{\tau\}$, and then
  - turn into a process P'_i
- then the other process $P_j \quad (j \in \{1,2\} \setminus \{i\})$ must be able
  - observably execute the same action a, and then
  - turn into a process P'_j, which is equivalent to P'_i.

Using notations (4.46) and (4.47), the above informally described concept of observational equivalence can be expressed formally as a binary relation μ on the set of all processes, which has the following properties.

(1) If $(P_1, P_2) \in \mu$, and for some process P'_1

$$P_1 \xrightarrow{\tau} P'_1 \tag{4.48}$$

then there is a process P'_2, such that

$$P_2 \xrightarrow{\tau^*} P'_2 \tag{4.49}$$

and

$$(P'_1, P'_2) \in \mu \tag{4.50}$$

(2) symmetric property: If $(P_1, P_2) \in \mu$, and for some process P'_2

$$P_2 \xrightarrow{\tau} P'_2 \tag{4.51}$$

then there is a process P'_1, such that

$$P_1 \xrightarrow{\tau^*} P'_1 \tag{4.52}$$

and (4.50).

(3) If $(P_1, P_2) \in \mu$, and for some process P_1'

$$P_1 \xrightarrow{a} P_1' \tag{4.53}$$

then there is a process P_2', such that

$$P_2 \xrightarrow{a_\tau} P_2' \tag{4.54}$$

and (4.50).

(4) symmetric property: If $(P_1, P_2) \in \mu$, and for some process P_2'

$$P_2 \xrightarrow{a} P_2' \tag{4.55}$$

then there is a process P_1', such that

$$P_1 \xrightarrow{a_\tau} P_1' \tag{4.56}$$

and (4.50).

Let $\mathcal{M}_\tau$ be a set of all binary relations on the set of processes, which have the above properties.

The set $\mathcal{M}_\tau$ is not empty: it contains, for example, the diagonal relation, which consists of all pairs (P, P), where P is an arbitrary process.

As in the case of strong equivalence, the natural question arises about what kind of a relationship, within the set $\mathcal{M}_\tau$, can be used for a definition of the concept of observational equivalence.

Just as in the case of strong equivalence, we offer the following answer to this question: we will consider P_1 and P_2 as observationally equivalent if and only if there is a relation $\mu \in \mathcal{M}_\tau$, that contains the pair (P_1, P_2), i.e. we define a relation of observational equivalence on the set of all processes as the union of all relations from$\mathcal{M}_\tau$. This relation is denoted by the symbol $\approx$.

It is easy to prove that

- $\approx \in \mathcal{M}_\tau$,
- $\approx$ is an equivalence relation, because
  - reflexivity of $\approx$ follows from the fact that the diagonal relation belongs to $\mathcal{M}_\tau$,

- symmetry of $\approx$ follows from the fact that if $\mu \in \mathcal{M}_\tau$, then $\mu^{-1} \in \mathcal{M}_\tau$
- transitivity of $\approx$ follows from the fact that if $\mu_1 \in \mathcal{M}_\tau$ and $\mu_2 \in \mathcal{M}_\tau$, then $\mu_1 \circ \mu_2 \in \mathcal{M}_\tau$.

If processes P_1 and P_2 are observationally equivalent, then this fact is indicated by

$$P_1 \approx P_2$$

It is easy to prove that if processes P_1 and P_2 are strongly equivalent, then they are observationally equivalent.

4.8.2 Logical criterion of observational equivalence

A **logical criterion of observational equivalence** is similar to the analogous criterion from section 4.4.1. In this criteria it is used the same set Fm of formulas. The notion of a value of a formula on a process differs from the analogous notion in section 4.4.1 only for formulas of the form $\langle a\rangle\varphi$:

- a value of the formula $\langle\tau\rangle\varphi$ on the process P is equal to

$$\left\{\begin{array}{ll} 1, & \text{if there is a process } P' : \\ & \quad P \xrightarrow{\tau^*} P' \,,\ P'(\varphi) = 1 \\ 0, & \text{otherwise} \end{array}\right.$$

- a value of the formula $\langle a\rangle\varphi$ (where $a \neq \tau$) on P is equal to

$$\left\{\begin{array}{ll} 1, & \text{if there is a process } P' : \\ & \quad P \xrightarrow{a_\tau} P' \,,\ P'(\varphi) = 1 \\ 0, & \text{otherwise} \end{array}\right.$$

For each process P the notation $Th_\tau(P)$ denotes a set of all formulas which have a value 1 on the process P (with respect to the modified definition of the notion of a value of a formula on a process).

Theorem 13 .
Let P_1 and P_2 be finite processes. Then

$$P_1 \approx P_2 \quad \Leftrightarrow \quad Th_\tau(P_1) = Th_\tau(P_2) \quad \blacksquare$$

As in the case of $\sim$, there is a problem of finding for two given processes P_1 and P_2 a list of formulas of a smallest size

$$\varphi_1, \ldots, \varphi_n$$

such that $P_1 \approx P_2$ if and only if

$$\forall\, i = 1, \ldots, n \qquad P_1(\varphi_i) = P_2(\varphi_i)$$

Using theorem 13, we can easily prove that

$$\text{for each process } P \qquad P \;\approx\; \tau.P \tag{4.57}$$

Note that,

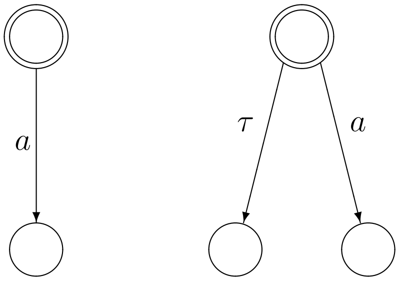
- according to (4.57), the following statement holds:

$$\mathbf{0} \;\approx\; \tau.\,\mathbf{0}$$

- however, the statement

$$\mathbf{0} + a.\mathbf{0} \;\approx\; \tau.\,\mathbf{0} + a.\mathbf{0} \quad (\text{where } a \neq \tau) \tag{4.58}$$

  does not hold, what is easy to see by considering the graph representation of left and right sides of (4.58):



  A formula, which takes different values on these processes, may have, for example, the following form:

$$\neg\langle\tau\rangle\neg\langle a\rangle\top$$

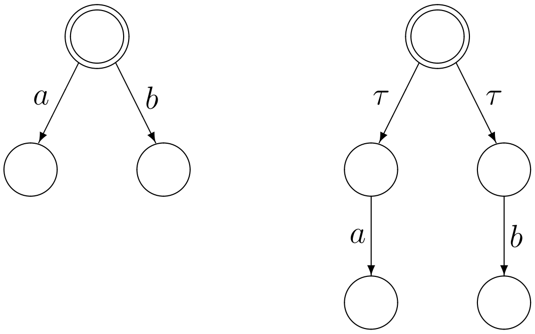
Thus, the relation $\approx$ is not a congruence, as it does not preserve the operation $+$.

Another example: if $a, b \in Act \setminus \{\tau\}$ and $a \neq b$, then

$$a.\mathbf{0} + b.\mathbf{0} \quad \not\approx \quad \tau.a.\mathbf{0} + \tau.b.\mathbf{0}$$

although $a.\mathbf{0} \approx \tau.a.\mathbf{0}$ and $b.\mathbf{0} \approx \tau.b.\mathbf{0}$.

A graph representation of these processes has the form



The fact that these processes are not observationally equivalent is substantiated by the formula

$$\langle\tau\rangle\neg\langle a\rangle\top$$

4.8.3 A criterion of observational equivalence based on the concept of an observational BS

For the relation $\approx$ there is an analog of the criterion based on the concept of BS (theorem 2 in section 4.4.2). For its formulation we shall introduce auxiliary notations.

Let $P = (S, s^0, R)$ be a process, and s_1, s_2 be a pair of its states. Then

- the notation

  $$s \xrightarrow{\tau^*} s'$$

  means that

  - either $s = s'$,

– or there is a sequence of states

$$s_1, \ldots, s_n \quad (n \geq 2)$$

such that $s_1 = s, \;\; s_n = s'$, and $\forall\, i = 1, \ldots, n-1$

$$(\, s_i \xrightarrow{\tau} s_{i+1} \,) \;\; \in R$$

- the notation

$$s \xrightarrow{a_\tau} s' \quad (\text{where } a \neq \tau)$$

means that there are states s_1 and s_2, such that

$$s \xrightarrow{\tau^*} s_1\,, \quad s_1 \xrightarrow{a} s_2\,, \quad s_2 \xrightarrow{\tau^*} s'\,.$$

Theorem 14 .
Let P_1 and P_2 be processes of the form

$$P_i = (S_i, s_i^0, R_i) \qquad (i = 1, 2)$$

Then $P_1 \approx P_2$ if and only if there is a relation

$$\mu \subseteq S_1 \times S_2$$

satisfying the following conditions.

0. $(s_1^0, s_2^0) \in \mu$.

1. For each pair $(s_1, s_2) \in \mu$ and each transition from R_1 of the form

$$s_1 \xrightarrow{\tau} s_1'$$

there is a state $s_2' \in S_2$, such that

$$s_2 \xrightarrow{\tau^*} s_2'$$

and

$$(s_1', s_2') \in \mu \tag{4.59}$$

2. For each pair $(s_1, s_2) \in \mu$ and each transition from R_2 of the form

$$s_2 \xrightarrow{\tau} s_2'$$

there is a state $s_1' \in S_1$, such that

$$s_1 \xrightarrow{\tau^*} s_1'$$

and (4.59).

3. For each pair $(s_1, s_2) \in \mu$ and each transition from R_1 of the form
$$s_1 \xrightarrow{a} s_1' \quad (a \neq \tau)$$
there is a state $s_2' \in S_2$, such that
$$s_2 \xrightarrow{a_\tau} s_2'$$
and (4.59).

4. For each pair $(s_1, s_2) \in \mu$ and each transition from R_2 of the form
$$s_2 \xrightarrow{a} s_2' \quad (a \neq \tau)$$
there is a state $s_1' \in S_1$, such that
$$s_1 \xrightarrow{a_\tau} s_1'$$
and (4.59).

A relation μ, satisfying these conditions, is called an **observational BS (OBS)**between P_1 and P_2.

4.8.4 Algebraic properties of observational equivalence

Theorem 15.

The relation of observational equivalence preserves all operations on processes except for the operation $+$, i.e. if $P_1 \approx P_2$, then

- for each $a \in Act \quad a.P_1 \approx a.P_2$
- for each process $P \quad P_1|P \approx P_2|P$
- for each $L \subseteq Names \quad P_1 \setminus L \approx P_2 \setminus L$
- for each renaming $f \quad P_1[f] \approx P_2[f]$

Proof.

As it was established in section 4.8.3, the statement $P_1 \approx P_2$ is equivalent to the following statement: there is an OBS μ between P_1 and P_2. Using this μ, we construct OBSs for justification of each of the foregoing statements.

- Let $s^0_{(1)}$ and $s^0_{(2)}$ be initial states of the processes $a.P_1$ and $a.P_2$ respectively.

  Then the relation

  $$\{((s_1, s), (s_2, s)) \mid (s_1, s_2) \in \mu,\ q \in S\}$$

  is an OBS between $P_1|P$ and $P_2|P$.

- Let S be a set of states of the process P. Then the relation

  $$\{((s_1, s), (s_2, s)) \mid (s_1, s_2) \in \mu,\ q \in S\}$$

  is an OBS between $P_1|P$ and $P_2|P$.

- the relation μ is an OBS
  - between $P_1 \setminus L$ and $P_2 \setminus L$, and
  - between $P_1[f]$ and $P_2[f]$. ■

4.8.5 Recognition of observational equivalence and minimization of processes with respect to $\approx$

The problems of

1. recognition for two given finite processes, whether they are observationally equivalent, and
2. construction for a given finite process P such a process Q, that has the smallest number of states among all processes, which are observationally equivalent to P

can be solved on the base of a theory that is analogous to the theory contained in sections 4.6 and 4.7.

We will not explain in detail this theory, because it is analogous to the theory for the case $\sim$. In this theory, for any pair of processes

$$P_i = (S_i, s^0_i, R_i) \qquad (i = 1, 2)$$

also it is determined an operator $'$ on relations from S_1 to S_2, that maps each relation $\mu \subseteq S_1 \times S_2$ to the relation μ'_τ, such that

$$\begin{array}{l}\mu \text{ satisfies conditions 1, 2, 3, 4} \\ \text{from the definition of OBS}\end{array} \quad \Leftrightarrow \quad \mu \subseteq \mu'_\tau$$

In particular,

$$\mu \text{ is OBS between } P_1 \text{ and } P_2 \quad \Leftrightarrow \quad \begin{cases} (s_1^0, s_2^0) \in \mu \\ \mu \subseteq \mu'_\tau \end{cases}$$

Let $\mu_\tau(P_1, P_2)$ be a union of all relations from the set

$$\{\mu \subseteq S_1 \times S_2 \mid \mu \subseteq \mu'_\tau\} \tag{4.60}$$

The relation $\mu_\tau(P_1, P_2)$ is the greatest element (with respect to an inclusion) of the set (4.60), and has the property

$$P_1 \approx P_2 \quad \Leftrightarrow \quad (s_1^0, s_2^0) \in \mu_\tau(P_1, P_2)$$

From the definition of the relation $\mu_\tau(P_1, P_2)$ follows that it consists of all pairs $(s_1, s_2) \in S_1 \times S_2$, such that

$$P_1(s_1) \approx P_2(s_2)$$

The relation $\mu_\tau(P_1, P_2)$ can be considered as another similarity measure between P_1 and P_2.

These is a polynomial algorithm of a computation of the relation $\mu_\tau(P_1, P_2)$. This algorithm is similar to the corresponding algorithm from section 4.6.2. For constructing of this algorithm it should be considered the following consideration. For checking the condition

$$s \xrightarrow{\tau^*} s'$$

(where s, s' are states of a process P) it is enough to analyze sequences of transitions of the form

$$s \xrightarrow{\tau} s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \ldots$$

length of which does not exceed a number of states of the process P.

4.8.6 Other criteria of equivalence of processes

For proving that processes P_1 and P_2 are strongly equivalent or observationally equivalent, the following criteria can be used. In some cases, use of these criteria for proving of an appropriate equivalence between P_1 and P_2 is much easier than all other methods.

A binary relation μ on the set of processes is said to be

- BS (mod $\sim$), if $\mu \subseteq (\sim \mu \sim)'$
- OBS (mod $\sim$), if $\mu \subseteq (\sim \mu \sim)'_\tau$
- OBS (mod $\approx$), if $\mu \subseteq (\approx \mu \approx)'_\tau$

It is easy to prove that

- if μ is BS (mod $\sim$), then $\mu \subseteq \sim$, and
- if μ is OBS (mod $\sim$ or mod $\approx$), then $\mu \subseteq \approx$.

Thus, to prove $P_1 \sim P_2$ or $P_1 \approx P_2$ it is enough to find a suitable

- BS (mod $\sim$), or
- OBS (mod $\sim$ or mod $\approx$)

respectively, such that

$$(P_1, P_2) \in \mu$$

4.9 Observational congruence

4.9.1 A motivation of the concept of observational congruence

As stated above, a concept of equivalence of processes can be defined not uniquely. In the previous sections have already been considered different types of equivalence of processes. Each of these equivalences reflects a certain point of view on what types of a behavior should be considered as equal.

In addition to these concepts of equivalence of processes, it can be determined, for example, such concepts of equivalence, that

- take into account a duration of an execution of actions, i.e., in particular, one of conditions of equivalence of processes P_1 and P_2 can be as follows:
  - if one of these processes P_i may, within a some period of time imperceptibly turn into a process P'_i,

  - then the other process $P_j \quad (j \in \{1,2\} \setminus \{i\})$ must be able for approximately the same amount of time imperceptibly turn into a process P'_j, which is equivalent to P'_i
    (where the concept of "approximately the same amount of time" can be clarified in different ways)

- or take into account the property of **fairness**, i.e. processes can not be considered as equivalent, if

  - one of them is fair, and
  - another is not fair

  where one of possible definitions of fairness of processes is as follows: a process is said to be **fair** if there is no an infinite sequence of transitions of the form

  $$s_0 \xrightarrow{\tau} s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \ldots$$

  such that the state s_0 is reachable, and for each $i \geq 0$

  $$Act(s_i) \setminus \{\tau\} \neq \emptyset$$

  Note that observational equivalence does not take into account the property of fairness: there are two processes P_1 and P_2, such that

  - $P_1 \approx P_2$, but
  - P_1 is fair, and P_2 is not fair.

  For example

  - $P_1 = a.\mathbf{0}$, where $a \neq \tau$,
  - $P_2 = a.\mathbf{0} \,|\, \tau^*$, where the process τ^* has one state and one transition with a label τ

- etc.

In every particular situation, a decision about which a concept of equivalence of processes is best used, essentially depends on the purposes for which this concept is intended.

In this section we define another kind of equivalence of processes called an **observational congruence**. This equivalence is denoted by $\overset{+}{\approx}$. We define this equivalence, based on the following conditions that it must satisfy.

1. Processes that are equivalent with respect to $\overset{+}{\approx}$, must be observationally equivalent.

2. Let

   - a process P is constructed as a composition of processes

     $$P_1, \ldots, P_n$$

     that uses operations

     $$a., \quad +, \quad |, \quad \backslash L, \quad [f] \tag{4.61}$$

   - and we replace one of components of this composition (for example, the process P_i), on other process P'_i, which is equivalent to P_i.

   A process which is obtained from P by this replacement, must be equivalent to the original process P.

It is easy to prove that an equivalence μ on the set of processes satisfies the above conditions if and only if

$$\left\{\begin{array}{l} \mu \ \subseteq \approx \\ \mu \ \text{is a congruence} \\ \qquad\quad \text{with respect to operations (4.61)} \end{array}\right. \tag{4.62}$$

There are several equivalences which satisfy conditions (4.62). For example,

- ithe diagonal relation (consisting of pairs of the form (P, P)), and
- strong equivalence ($\sim$)

satisfy these conditions.

Below we prove that among all equivalences satisfying conditions (4.62), there is greatest equivalence (with respect to inclusion). It is natural to consider this equivalence as the desired equivalence ($\overset{+}{\approx}$).

4.9.2 Definition of a concept of observational congruence

To define a concept of observational congruence, we introduce an auxiliary notation.

Let P and P' be a couple of processes. The notation

$$P \xrightarrow{\tau^+} P'$$

means that there is a sequence of processes

$$P_1, \ldots, P_n \quad (n \geq 2)$$

such that

- $P_1 = P, \quad P_n = P'$, and
- for each $i = 1, \ldots, n-1$

$$P_i \xrightarrow{\tau} P_{i+1}$$

We shall say that processes P_1 and P_2 are in a relation of **observational congruence** and denote this fact by

$$P_1 \overset{+}{\approx} P_2$$

if the following conditions hold.

(0) $P_1 \approx P_2$.

(1) If, a process P_1' is such that

$$P_1 \xrightarrow{\tau} P_1' \tag{4.63}$$

then there is a process P_2', such that

$$P_2 \xrightarrow{\tau^+} P_2' \tag{4.64}$$

and

$$P_1' \approx P_2' \tag{4.65}$$

(2) Symmetrical condition: if a process P_2' is such that

$$P_2 \xrightarrow{\tau} P_2' \tag{4.66}$$

then there is a process P_1', such that

$$P_1 \xrightarrow{\tau^+} P_1' \tag{4.67}$$

and (4.65).

It is easy to prove that observational congruence is an equivalence relation.

4.9.3 Logical criterion of observational congruence

A **logical criterion of observational congruence** of two processes is produced by a slight modification of the logical criterion of observational equivalence from section 4.8.2.

A set of formulas Fm^+, which is used in this criterion, is an extension of the set of formulas Fm from section 4.4.2. Fm^+ is obtained from Fm by adding a modal connective $\langle\tau^+\rangle$.

The set Fm^+ is defined as follows.

- Every formula from Fm belongs to Fm^+.
- For every formula $\varphi \in Fm$ the string

$$\langle\tau^+\rangle\varphi$$

is a formula from Fm^+.

For every formula $\varphi \in Fm^+$ and every process P a **value** of φ on P is denoted by $P(\varphi)$ and is defined as follows.

- If $\varphi \in Fm$, then $P(\varphi)$ is defined as in section 4.8.2.
- If $\varphi = \langle\tau^+\rangle\psi$, where $\psi \in Fm$, then

$$P(\varphi) \overset{\text{def}}{=} \begin{cases} 1, & \text{if there is a process } P' : \\ & \quad P \xrightarrow{\tau^+} P',\ P'(\psi) = 1 \\ 0, & \text{otherwise} \end{cases}$$

For each process P we denote by $Th^+_\tau(P)$ a set of all formulas $\varphi \in Fm^+$, such that $P(\varphi) = 1$.

Theorem 16.
Let P_1 and P_2 be finite processes. Then

$$P_1 \overset{+}{\approx} P_2 \quad \Leftrightarrow \quad Th^+_\tau(P_1) = Th^+_\tau(P_2) \quad \blacksquare$$

As in the case of $\sim$ and $\approx$, there is a problem of finding for two given processes P_1 and P_2 a list of formulas of a smallest size

$$\varphi_1, \ldots, \varphi_n \ \in Fm^+$$

such that $P_1 \overset{+}{\approx} P_2$ if and only if

$$\forall \, i = 1, \ldots, n \qquad P_1(\varphi_i) = P_2(\varphi_i)$$

4.9.4 Criterion of observational congruence based on the concept of observational BS

We shall use the following notation. Let

- P be a process of the form (S, s^0, R), and
- s_1, s_2 be a pair of states from S.

Then the notation

$$s \xrightarrow{\tau^+} s'$$

means that there is a sequence of states

$$s_1, \ldots, s_n \quad (n \geq 2)$$

such that $s_1 = s$, $\ s_n = s'$, and for each $i = 1, \ldots, n-1$

$$(\, s_i \xrightarrow{\tau} s_{i+1} \,) \ \in R$$

Theorem 17 .
Let P_1, P_2 be a pair of processes of the form

$$P_i = (S_i, s^0_i, R_i) \qquad (i = 1, 2)$$

The statement $P_1 \overset{+}{\approx} P_2$ holds if and only if there is a relation

$$\mu \subseteq S_1 \times S_2$$

satisfying the following conditions.

0. μ is an OBS between P_1 and P_2
(the concept of an OBS is described in section 4.8.3).

1. For each transition from R_1 of the form

$$s_1^0 \xrightarrow{\tau} s_1'$$

there is a state $s_2' \in S_2$, such that

$$s_2^0 \xrightarrow{\tau^+} s_2'$$

and

$$(s_1', s_2') \in \mu \tag{4.68}$$

2. For each transition from R_2 of the form

$$s_2^0 \xrightarrow{\tau} s_2'$$

there exists a state $s_1' \in S_1$, such that

$$s_1^0 \xrightarrow{\tau^+} s_1'$$

and (4.68). ■

Below the string OBS$^+$ is an abbreviated notation of the phrase

"an OBS satisfying conditions 1 and 2 of theorem 17".

4.9.5 Algebraic properties of observational congruence

Theorem 18.

The observational congruence is a congruence with respect to all operaions on processes, i.e. if $P_1 \overset{+}{\approx} P_2$, then

- for each $a \in Act$ $\quad a.P_1 \overset{+}{\approx} a.P_2$
- for each process P $\quad P_1 + P \overset{+}{\approx} P_2 + P$
- for each process P $\quad P_1 | P \overset{+}{\approx} P_2 | P$
- for each $L \subseteq Names$ $\quad P_1 \setminus L \overset{+}{\approx} P_2 \setminus L$

- for each renaming $f \quad P_1[f] \overset{+}{\approx} P_2[f]$

Proof.

As it was stated in section 4.9.4, the statement $P_1 \overset{+}{\approx} P_2$ holds if and only if there is OBS$^+$ μ between P_1 and P_2. Using this μ, for each of the above statements we shall justify this statement by construction of corresponding OBS$^+$.

- Let $s^0_{(1)}$ and $s^0_{(2)}$ be initial states of the processes $a.P_1$ and $a.P_2$ respectively.

  Then the relation
  $$\{(s^0_{(1)}, s^0_{(2)})\} \ \cup \ \mu$$
  is OBS$^+$ between $a.P_1$ and $a.P_2$

- Let

  - $s^0_{(1)}$ and $s^0_{(2)}$ be initial states of $P_1 + P$ and $P_2 + P$ respectively, and
  - S be denote a set of states of the process P.

  Then the relation
  $$\{(s^0_{(1)}, s^0_{(2)})\} \ \cup \ \mu \ \cup \ Id_S$$
  is OBS$^+$ between $P_1 + P$ and $P_2 + P$.

- Let S be a set of states of the process P. Then the relation
  $$\{((s_1, s), (s_2, s)) \mid (s_1, s_2) \in \mu, \ q \in S\}$$
  is OBS$^+$ between $P_1|P$ and $P_2|P$.

- The relation μ is OBS$^+$

  - between $P_1 \setminus L$ and $P_2 \setminus L$, and
  - between $P_1[f]$ and $P_2[f]$. ■

Theorem 19.

For any processes P_1 and P_2

$$P_1 \approx P_2 \quad \Leftrightarrow \quad \begin{cases} P_1 \overset{+}{\approx} P_2 & \text{or} \\ P_1 \overset{+}{\approx} \tau.P_2 & \text{or} \\ \tau.P_1 \overset{+}{\approx} P_2 & \end{cases}$$

Proof.
The implication "←" follows from

- the inclusion $\overset{+}{\approx}\ \subseteq \approx$, and
- the fact that

$$\text{for any process } P \quad P \approx \tau.P \tag{4.69}$$

Prove the implication " →". Suppose

$$P_1 \approx P_2 \tag{4.70}$$

and

$$\text{it is not true that } P_1 \overset{+}{\approx} P_2 \tag{4.71}$$

(4.71) can occur, for example, in the following case:

$$\begin{array}{l} \text{there is a process } P_1', \text{ such that} \\ P_1 \xrightarrow{\tau} P_1' \end{array} \tag{4.72}$$

and

$$\begin{array}{l} \text{there is no a process } P_2' \approx P_1', \\ \text{such that } P_2 \xrightarrow{\tau^+} P_2' \end{array} \tag{4.73}$$

We shall prove that in this case

$$P_1 \overset{+}{\approx} \tau.P_2$$

According to the definition of observational congruence, we must prove that conditions (0), (1) and (2) from this definition are satisfied.

(0) : $P_1 \approx \tau.P_2$.

This condition follows from (4.70) and (4.69).

(1) : if there is a process P_1' such that

$$P_1 \xrightarrow{\tau} P_1' \tag{4.74}$$

then there is a process $P_2' \approx P_1'$ such that

$$\tau.P_2 \xrightarrow{\tau^+} P_2' \tag{4.75}$$

From (4.70), (4.74), and from the definition of observational equivalence it follows that these is a process $P_2' \approx P_1'$ such that

$$P_2 \xrightarrow{\tau^*} P_2' \tag{4.76}$$

(4.75) follows from $\tau.P_2 \xrightarrow{\tau} P_2$ and (4.76).

(2) : if there is a process P_2' such that

$$\tau.P_2 \xrightarrow{\tau} P_2' \tag{4.77}$$

then there is a process $P_1' \approx P_2'$ such that

$$P_1 \xrightarrow{\tau^+} P_1'$$

From the definition of the operation of prefix actions and from (4.77) we get the equality

$$P_2' = P_2$$

Thus, we must prove that

$$\begin{array}{l}\text{for some process } P_1' \approx P_2\\ \text{the formula } P_1 \xrightarrow{\tau^+} P_1' \text{ holds}\end{array} \tag{4.78}$$

Let P_1' be a process that is referred in the assumption (4.72). From the assumption (4.70) we get

$$\begin{array}{l}\text{there is a process } P_2' \approx P_1',\\ \text{such that } P_2 \xrightarrow{\tau^*} P_2'\end{array} \tag{4.79}$$

Comparing (4.79) and (4.73), we get the equality $P_2' = P_2$, i.e., we have proved (4.78).

(4.71) may be true also on the reason that

- there is a process P_2', such that $P_2 \xrightarrow{\tau} P_2'$, and
- there is no a process $P_1' \approx P_2'$, such that

$$P_1 \xrightarrow{\tau^+} P_1'$$

In this case, by similar reasoning it can be proven that

$$\tau.P_1 \overset{+}{\approx} P_2 \quad \blacksquare$$

Theorem 20.
The relation $\overset{+}{\approx}$ coincides with the relation

$$\{(P_1, P_2) \mid \forall\, P \quad P_1 + P \approx P_2 + P\} \tag{4.80}$$

Proof.
The inclusion $\overset{+}{\approx}\ \subseteq$ (4.80) follows from the fact that

- $\overset{+}{\approx}$ is a congruence (i.e., in particular, $\overset{+}{\approx}$ preserves the operation "+"), and
- $\overset{+}{\approx}\ \subseteq \approx$.

Prove the inclusion

$$(4.80)\ \subseteq\ \overset{+}{\approx}$$

Let $(P_1, P_2) \in (4.80)$.
Since for each process P the following statement holds

$$P_1 + P \approx P_2 + P \tag{4.81}$$

then, setting in (4.81) $P \overset{\text{def}}{=} \mathbf{0}$, we get

$$P_1 + \mathbf{0} \approx P_2 + \mathbf{0} \tag{4.82}$$

Since

- for each process P the following statement holds:

$$P + \mathbf{0} \sim P$$

- and , furthermore, $\sim\ \subseteq \approx$

then from (4.82) we get

$$P_1 \approx P_2 \tag{4.83}$$

If it is not true that $P_1 \overset{+}{\approx} P_2$, then from (4.83) on the reason of theorem 19 we get that

- either $P_1 \overset{+}{\approx} \tau.P_2$,
- or $\tau.P_1 \overset{+}{\approx} P_2$

Consider, for example, the case

$$P_1 \overset{+}{\approx} \tau.P_2 \tag{4.84}$$

(the other case is considered analogously).

Since $\overset{+}{\approx}$ is a congruence, then from (4.84) it follows that for any process P

$$P_1 + P \overset{+}{\approx} \tau.P_2 + P \tag{4.85}$$

From

- (4.81), (4.85), and
- the inclusion $\overset{+}{\approx}\ \subseteq \approx$

it follows that for any process P

$$P_2 + P \approx \tau.P_2 + P \tag{4.86}$$

Prove that

$$P_2 \overset{+}{\approx} \tau.P_2 \tag{4.87}$$

(4.87) equivalent to the following statement: there is a process $P_2' \approx P_2$, such that

$$P_2 \xrightarrow{\tau^+} P_2' \tag{4.88}$$

Since the set $Names$ is infinite (by an assumption from section 2.3), then there is an action $b \in Act \setminus \{\tau\}$, which does not occur in P_2.

Statement (4.86) must be true in the case when P has the form $b.\mathbf{0}$, i.e. the following statement must be true:

$$P_2 + b.\mathbf{0} \approx \tau.P_2 + b.\mathbf{0} \tag{4.89}$$

Since

$$\tau.P_2 + b.\mathbf{0} \xrightarrow{\tau} P_2$$

then

- from (4.89), and
- from the definition of the relation $\approx$

it follows that there is a process $P_2' \approx P_2$ such that

$$P_2 + b.\mathbf{0} \xrightarrow{\tau^*} P_2' \tag{4.90}$$

The case $P_2 + b.\mathbf{0} = P_2'$ is impossible, because

- the left side of this equality does contain the action b, and
- the right side of this equality does not contain the action b.

Consequently, on the reason of (4.90), we get the statement

$$P_2 + b.\mathbf{0} \xrightarrow{\tau^+} P_2' \tag{4.91}$$

From the definition of the operation $+$, it follows that (4.91) is possible if and only if (4.88) holds.

Thus, we have proved that there is a process $P_2' \approx P_2$ such that (4.88) holds, i.e. we have proved (4.87).

(4.84) and (4.87) imply that $P_1 \overset{+}{\approx} P_2$. ■

Theorem 21 .

$\overset{+}{\approx}$ is the greatest congruence contained in $\approx$, i.e. for each congruence ν on the set of all processes the following implication holds:

$$\nu \subseteq \approx \quad \Rightarrow \quad \nu \subseteq \overset{+}{\approx}$$

Proof.

Prove that if $(P_1, P_2) \in \nu$, then $P_1 \overset{+}{\approx} P_2$.

Let $(P_1, P_2) \in \nu$. Since ν is a congruence, then

$$\text{for each process } P \quad (P_1 + P, P_2 + P) \in \nu \tag{4.92}$$

If $\nu \subseteq \approx$, then from (4.92) it follows that

$$\text{for each process } P \quad P_1 + P \approx P_2 + P \tag{4.93}$$

According to theorem 20, (4.93) implies that $P_1 \overset{+}{\approx} P_2$. ■

Theorem 22 .

The relations $\sim$, $\approx$ and $\overset{+}{\approx}$ have the following property:

$$\sim \;\subseteq\; \overset{+}{\approx} \;\subseteq\; \approx \tag{4.94}$$

Proof.
The inclusion $\overset{+}{\approx} \;\subseteq \approx$ holds by definition of $\overset{+}{\approx}$.
The inclusion $\sim \subseteq \; \overset{+}{\approx}$ follows from

- the inclusion $\sim \subseteq \approx$, and
- from the fact that if processes P_1, P_2 are such that

  $$P_1 \sim P_2$$

  then this pair of processes satisfies conditions from the definition of the relation $\overset{+}{\approx}$. ■

Note that both inclusions in (4.94) are proper:

- $a.\tau.\mathbf{0} \not\sim a.\mathbf{0}$, but $a.\tau.\mathbf{0} \overset{+}{\approx} a.\mathbf{0}$
- $\tau.\mathbf{0} \overset{+}{\not\approx} \mathbf{0}$, but $\tau.\mathbf{0} \approx \mathbf{0}$

Theorem 23 .

1. If $P_1 \approx P_2$, then for each $a \in Act$

   $$a.P_1 \overset{+}{\approx} a.P_2$$

   In particular, for each process P

   $$a.\tau.P \overset{+}{\approx} a.P \tag{4.95}$$

2. For any process P

   $$P + \tau.P \overset{+}{\approx} \tau.P \tag{4.96}$$

3. For any processes P_1 and P_2, and any $a \in Act$

   $$a.(P_1 + \tau.P_2) + a.P_2 \overset{+}{\approx} a.(P_1 + \tau.P_2) \tag{4.97}$$

4. For any processes P_1 and P_2

$$P_1 + \tau.(P_1 + P_2) \overset{+}{\approx} \tau.(P_1 + P_2) \tag{4.98}$$

Proof.

For each of the above statements we shall construct an OBS$^+$ between its left and right sides.

1. As it was stated in theorem 14 (section 4.8.3), the statement $P_1 \approx P_2$ is equivalent to the statement that there is an OBS μ between P_1 and P_2.

   Let $s^0_{(1)}$ and $s^0_{(2)}$ be initial states of the processes $a.P_1$ and $a.P_2$ respectively.

   Then the relation
   $$\{(s^0_{(1)}, s^0_{(2)})\} \cup \mu$$
   is an OBS$^+$ between $a.P_1$ and $a.P_2$.

   (4.95) follows from

   - the above statement, and
   - the statement $\tau.P \approx P$, which holds according to (4.57).

2. Let P has the form
   $$P = (S, s^0, R)$$
   and let $S_{(1)}$? $S_{(2)}$ be duplicates of the set S in the processes P and $\tau.P$ respectively, which contain in the left side of the statement (4.96). Elements of these duplicates will be denoted by $s_{(1)}$ and $s_{(2)}$ respectively, where s is an arbitrary element of the set S.

   Let s^0_l and s^0_r be initial states of the processes in the left and right sides of (4.96) respectively. Then the relation

   $$\{(s^0_l, s^0_r)\} \cup \{(s_{(i)}, s) \mid s \in S,\ i = 1, 2\}$$

   is OBS$^+$ between left and right sides of the statement (4.96).

3. Let $P_i = (S_i, s^0_i, R_i) \quad (i = 1, 2)$. We can assume that $S_1 \cap S_2 = \emptyset$. Let

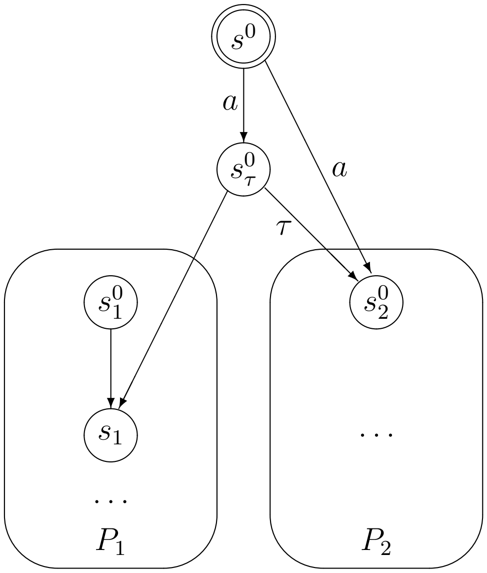
- s^0_τ be an initial state of the process
$$P_1 + \tau.P_2 \tag{4.99}$$
- s^0 be an initial state of the process
$$a.(P_1 + \tau.P_2) \tag{4.100}$$

Note that (4.100) coincides with the right side of (4.97).

The left side of (4.97) is strongly equivalent to the process P', which is obtained from (4.100) by adding the transition

$$s^0 \xrightarrow{a} s^0_2$$

it is easily to make sure in this by considering the graph representation of the process P', which has the form



It is easy to prove that the process P' is observationally congruent to the process (4.100). The sets of states of these processes can be considered as duplicates $S_{(1)}$ and $S_{(2)}$ of one and the same set S, and OBS^+ between P' and (4.100) has the form

$$\{(s_{(1)}, s_{(2)}) \mid s \in S\} \tag{4.101}$$

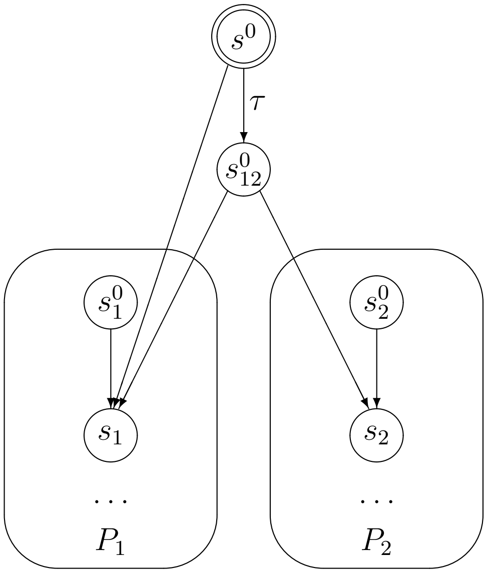
Since

- according to theorem 22, we have the inclusion $\sim \subseteq \overset{+}{\approx}$, and
- (4.100) coincides with the right part of (4.97),

then we have proved that the left and right sides of the statement (4.97) are observationally congruent. ■

4. Reasonings in this case are similar to the reasonings in the previous case. We will not explain them in detail, only note that

   - left part of the statement (4.98) is strongly equivalent to the process P', which has the following graph representation:



   where

     - s^0_1 and s^0_2 are initial states of the processes P_1 and P_2, and
     - s^0_{12} is an initial state of the process $P_1 + P_2$

   - the right part of the statement (4.98) (which we denote by P'') is obtained from P' by removing of transitions of the form

$$s^0 \longrightarrow s_1$$

It is easy to prove that $P' \overset{+}{\approx} P''$. Sets of states of these processes can be considered as duplicates $S_{(1)}$ and $S_{(2)}$ of one and the same set S, and OBS$^+$ between P' and P'' has the form (4.101). ■

4.9.6 Recognition of observational congruence

To solve the problem of recognition for two given finite processes, whether they are observationally congruent, it can be used the following theorem.

Theorem 24.
Let P_1 and P_2 be finite processes. The statement

$$P_1 \overset{+}{\approx} P_2$$

holds if and only if

$$\left\{\begin{array}{l} (s_1^0, s_2^0) \in \mu_\tau(P_1, P_2) \\ \mu_\tau(P_1, P_2) \text{ is an OBS}^+ \end{array}\right. \quad \blacksquare$$

4.9.7 Minimization of processes with respect to observational congruence

To solve the problem of minimizing of finite processes with respect to observational congruence the following theorems can be used.

Theorem 25.
Let $P = (S, s^0, R)$ be a process.

Define a **factor-process** $P_\approx$ of the process P with respect to the equivalence $\mu_\tau(P, P)$, as a process with the following components.

- States of $P_\approx$ are equivalence classes of the set S with respect to the equivalence $\mu_\tau(P, P)$.
- An initial state of $P_\approx$ is the class $[s^0]$.
- Transitions of the process $P_\approx$ have the form

  $$[s_1] \xrightarrow{a} [s_2]$$

  where $s_1 \xrightarrow{a} s_2$ is an arbitrary transition from R.

Then $P \overset{+}{\approx} (P_\approx)$. $\blacksquare$

Theorem 26.
Let P' be a process which is obtained from a process P by removing of unreachable states. Then $P'_\approx$ has the smallest number of states among all processes that are observationally congruent to P. $\blacksquare$

Chapter 5

Recursive definitions of processes

In some cases, it is more convenient to describe a process by a recursive definition, intsead of explicit description of sets of its states and transitions. In the present chapter we introduce a method of description of processes by recursive definitions.

5.1 Process expressions

In order to formulate a notion of recursive description of a process we introduce a notion of a **process expression**.

A set PE of **process expressions (PE)** is defined inductively, i.e. we define

- elementary PEs, and
- rules for constructing new PEs from existing ones.

Elementary PEs have the following form.

process constants:

We assume that there is given a countable set of process constants, and each of them is associated with a certain process, which is called a **value** of this constant.

Each process constant is a PE.

There is a process constant, whose value is the empty process **0**. This constant is denoted by the same symbol **0**.

process names:
We assume that there is given a countable set **of process names**, and each process name is a PE.

Rules for constructing new PEs from existing ones have the following form.

prefix action:
For each $a \in Act$ and each PE P the string $a.P$ is a PE.

choice:
For any pair of PEs P_1, P_2 the string $P_1 + P_2$ is a PE.

parallel composition:
For any pair of PEs P_1, P_2 the string $P_1 \,|\, P_2$ is a PE.

restriction:
For each subset $L \subseteq Names$ and each PE P the string $P \setminus L$ is a PE.

renaming:
For each renaming f and each PE P the string $P[f]$ is a PE.

5.2 A notion of a recursive definition of processes

A **recursive definition (RD) of processes** is a list of formal equations of the form

$$\left\{\begin{array}{l} A_1 = P_1 \\ \dots \\ A_n = P_n \end{array}\right. \tag{5.1}$$

where

- $A_1, \dots, A_n$ are different process names, and
- $P_1, \dots, P_n$ are PEs, satisfying the following condition: for every $i = 1, \dots, n$ each process name, which has an occurrence in P_i, coincides with one of the names of $A_1, \dots, A_n$.

We shall assume that for each process name A there is a unique RD such that A has an occurrence in this RD.

In section 5.5 we define a correspondence, which associates with each PE P some process $[\![P]\!]$. To define this correspondence, we shall give first

- a notion of an **embedding** of processes, and
- a notion of a **limit** of a sequence of embedded processes.

5.3 Embedding of processes

Let P_1 and P_2 be processes of the form

$$P_i = (S_i, s_i^0, R_i) \quad (i = 1, 2) \tag{5.2}$$

The process P_1 is said to be **embedded** to the process P_2, if there is an injective mapping $f : S_1 \to S_2$, such that

- $f(s_1^0) = s_2^0$, and
- for any $s', s'' \in S_1$ and any $a \in Act$

$$(s' \xrightarrow{a} s'') \in R_1 \quad \Leftrightarrow \quad (f(s') \xrightarrow{a} f(s'')) \in R_2$$

For each pair of processes P_1, P_2 the notation

$$P_1 \hookrightarrow P_2$$

is an abridged notation of the statement that P_1 is embedded to P_2.

If the processes P_1 and P_2 have the form (5.2), and $P_1 \hookrightarrow P_2$, then we can identify P_1 with its image in P_2, i.e. we can assume that

- $S_1 \subseteq S_2$
- $s_1^0 = s_2^0$
- $R_1 \subseteq R_2$.

Theorem 27. Let $P_1 \hookrightarrow P_2$. Then

- $a.P_1 \hookrightarrow a.P_2$

- $P_1 + P \hookrightarrow P_2 + P$
- $P_1 \,|\, P \hookrightarrow P_2 \,|\, P$
- $P_1 \setminus L \hookrightarrow P_2 \setminus L$
- $P_1[f] \hookrightarrow P_2[f]$. $\blacksquare$

Below we consider expressions which are built from

- processes, and
- symbols of operations on processes ($a.$, $+$, $\,|\,$, $\setminus L$, $[f]$).

We call such expressions as **expressions over processes**. For each expression over processes it is defined a process which is a value of this expression. In the following reasonings we shall denote an expression over the process and its value by the same symbol.

Theorem 28 .
Let

- P be an expression over processes,
- $P_1, \ldots, P_n$ be a list of all processes occurred in P
- $P'_1, \ldots, P'_n$ be a list of processes such that

$$\forall\, i = 1, \ldots, n \quad P_i \hookrightarrow P'_i$$

- P' be an expression which is obtained from P by a replacement for each $i = 1, \ldots, n$ each occurrence of the process P_i to the corresponding process P'_i.

Then $P \hookrightarrow P'$.

Proof.
This theorem is proved by induction on a structure of the expression P. We prove that for each subexpression Q of the expression P

$$Q \hookrightarrow Q' \tag{5.3}$$

where Q' is a subexpression of the expression P', which corresponds to the subexpression Q.

base of induction:

If $Q = P_i$, then $Q' = P'_i$, and (5.3) holds by assumption.

inductive step:

From theorem 27 it follows that for each subexpression Q of the expression P the following implication holds: if for each proper subexpression Q_1 of Q the following statement holds

$$Q_1 \hookrightarrow Q'_1$$

then (5.3) holds.

Thus, (5.3) holds for each subexpression Q of P. In particular, (5.3) holds for P. ■

5.4 A limit of a sequence of embedded processes

Let $\{P_k \mid k \geq 0\}$ be a sequence of processes, such that

$$\forall k \geq 0 \quad P_k \hookrightarrow P_{k+1} \tag{5.4}$$

A sequence $\{P_k \mid k \geq 0\}$ satisfying condition (5.4) is called a **a sequence of embedded processes**.

Define a process $\lim\limits_{k\to\infty} P_k$, which is called a **limit** of the sequence of embedded processes $\{P_k \mid k \geq 0\}$.

Let the processes P_k $(k \geq 0)$ have the form

$$P_k = (S_k, s^0_k, R_k)$$

On the reason of (5.4), we can assume that $\forall k \geq 0$

- $S_k \subseteq S_{k+1}$
- $s^0_k = s^0_{k+1}$
- $R_k \subseteq R_{k+1}$

i.e. the components of the processes P_k $(k \geq 0)$ have the following properties:

- $S_0 \subseteq S_1 \subseteq S_2 \subseteq \ldots$
- $s^0_0 = s^0_1 = s^0_2 = \ldots$
- $R_0 \subseteq R_1 \subseteq R_2 \subseteq \ldots$

The process $\lim\limits_{k\to\infty} P_k$ has the form

$$(\bigcup_{k\geq 0} S_k, s^0_0, \bigcup_{k\geq 0} R_k)$$

It is easy to prove that for each $k \geq 0$

$$P_k \hookrightarrow \lim_{k\to\infty} P_k$$

Theorem 29.
Let $\{P_k \mid k \geq 0\}$ and $\{Q_k \mid k \geq 0\}$ be sequences of embedded processes. Then

- $\lim\limits_{k\to\infty}(a.P_k) = a.(\lim\limits_{k\to\infty} P_k)$
- $\lim\limits_{k\to\infty}(P_k + Q_k) = (\lim\limits_{k\to\infty} P_k) + (\lim\limits_{k\to\infty} Q_k)$
- $\lim\limits_{k\to\infty}(P_k \,|\, Q_k) = (\lim\limits_{k\to\infty} P_k) \,|\, (\lim\limits_{k\to\infty} Q_k)$
- $\lim\limits_{k\to\infty}(P_k \setminus L) = (\lim\limits_{k\to\infty} P_k) \setminus L$
- $\lim\limits_{k\to\infty}(P_k[f]) = (\lim\limits_{k\to\infty} P_k)[f]$ ■

Let

- P be a PE,
- A_1, ..., A_n be a list of all process names occurred in P.

Then for every n–tuple of processes P_1, ..., P_n the notation

$$P(P_1/A_1, \ldots, P_n/A_n)$$

denotes an expression over processes (as well as its value) obtained from P by replacement for each $i = 1, \ldots, n$ each occurrence of the process name A_i on the corresponding process P_i.

Theorem 30.
Let

- P be a PE, and
- A_1, ..., A_n be a list of all process names occurred in P.

Then for every list of sequences of embedded processes of the form

$$\{P_1^{(k)} \mid k \geq 0\}, \quad \ldots \quad \{P_n^{(k)} \mid k \geq 0\}$$

the following equality holds:

$$\begin{aligned} &P((\lim_{k\to\infty} P_1^{(k)})/A_1, \ldots, (\lim_{k\to\infty} P_n^{(k)})/A_n) = \\ &= \lim_{k\to\infty} P(P_1^{(k)}/A_1, \ldots, P_n^{(k)}/A_n) \end{aligned}$$

Proof.

This theorem is proved by induction on the structure of the PE P, using theorem 29. ■

5.5 Processes defined by process expressions

In this section we describe a rule which associates with each PE P a process $[\![P]\!]$, which is defined by this PE.

If P is a process constant, then $[\![P]\!]$ is a value of this constant.

If P has one of the following forms

$$a.P_1, \quad P_1 + P_2, \quad P_1 \,|\, P_2, \quad P_1 \setminus L, \quad P_1[f]$$

then $[\![P]\!]$ is a result of applying of the corresponding operation to the process P_1 or to the pair of processes (P_1, P_2), i.e.

$$\begin{aligned} &[\![a.P]\!] \stackrel{\text{def}}{=} a.[\![P]\!] \\ &[\![P_1 + P_2]\!] \stackrel{\text{def}}{=} [\![P_1]\!] + [\![P_2]\!] \\ &[\![P_1 \,|\, P_2]\!] \stackrel{\text{def}}{=} [\![P_1]\!] \,|\, [\![P_2]\!] \\ &[\![P \setminus L]\!] \stackrel{\text{def}}{=} [\![P]\!] \setminus L \\ &[\![P\ [f]\ ]\!] \stackrel{\text{def}}{=} [\![P]\!]\ [f] \end{aligned}$$

We now describe a rule that associates processes with process names.

Let $\{A_i = P_i \mid i = 1, \ldots, n\}$ be a RD.

Define a sequence of lists of processes

$$\{(P_1^{(k)}, \ldots, P_n^{(k)}) \mid k \geq 0\} \tag{5.5}$$

as follows:

- $P_1^{(0)} \overset{\text{def}}{=} \mathbf{0}, \ \dots, \ P_n^{(0)} \overset{\text{def}}{=} \mathbf{0}$
- if the processes $P_1^{(k)}$, ..., $P_n^{(k)}$ are already defined, then for each $i = 1, \dots, n$
$$P_i^{(k+1)} \overset{\text{def}}{=} P_i(P_1^{(k)}/A_1, \dots, P_n^{(k)}/A_n)$$

We prove that for each $k \geq 0$ and each $i = 1, \dots, n$

$$P_i^{(k)} \hookrightarrow P_i^{(k+1)} \tag{5.6}$$

The proof will proceed by induction on k.

base of induction:

If $k = 0$, then by definition $P_i^{(0)}$ coincides with the process $\mathbf{0}$, which can be embedded in any process.

inductive step:

Suppose that for each $i = 1, \dots, n \quad P_i^{(k-1)} \hookrightarrow P_i^{(k)}$.

By definition of the processes from the set (5.5), the following equalities hold:

$$\begin{aligned} P_i^{(k)} &= P_i(P_1^{(k-1)}/A_1, \dots, P_n^{(k-1)}/A_n) \\ P_i^{(k+1)} &= P_i(P_1^{(k)}/A_1, \dots, P_n^{(k)}/A_n) \end{aligned}$$

The statement $P_i^{(k)} \hookrightarrow P_i^{(k+1)}$ follows from theorem 28. ■

Define for each $i = 1, \dots, n$ the process $[\![A_i]\!]$ as the limit

$$[\![A_i]\!] \overset{\text{def}}{=} \lim_{k\to\infty} P_i^{(k)}$$

From theorem 30 it follows that for each $i = 1, \dots, n$ the following chain of equalities holds:

$$\begin{aligned} & P_i([\![A_1]\!]/A_1, \dots, [\![A_n]\!]/A_n) = \\ & = P_i((\lim_{k\to\infty} P_1^{(k)})/A_1, \dots, (\lim_{k\to\infty} P_n^{(k)})/A_n) = \\ & = \lim_{k\to\infty} P_i(P_1^{(k)}/A_1, \dots, P_n^{(k)}/A_n) = \\ & = \lim_{k\to\infty} (P_i^{(k+1)}) = [\![A_i]\!] \end{aligned}$$

i.e. the list of processes

$$[\![A_1]\!], \dots, [\![A_n]\!]$$

is a solution of the system of equations, which corresponds to the RD

$$\begin{cases} A_1 = P_1 \\ \dots \\ A_n = P_n \end{cases}$$

(variables of this system of equations are the process names A_1, …, A_n).

5.6 Equivalence of RDs

Suppose that there is given a couple of RDs of the form

$$\begin{cases} A_1^{(1)} = P_1^{(1)} \\ \dots \\ A_n^{(1)} = P_n^{(1)} \end{cases} \quad \text{and} \quad \begin{cases} A_1^{(2)} = P_1^{(2)} \\ \dots \\ A_n^{(2)} = P_n^{(2)} \end{cases} \tag{5.7}$$

For each n-tuple of processes Q_1, …, Q_n the string

$$P_i^{(j)}(Q_1, \dots, Q_n)$$

denotes the following expression on processes (and its value):

$$P_i^{(j)}(Q_1/A_1^{(j)}, \dots, Q_n/A_n^{(j)}) \qquad (i = 1, \dots, n;\ j = 1, 2)$$

Let μ be an equivalence on the set of all processes.

RDs (5.7) are said to be **equivalent** with respect to μ, if for

- each n–tuple of processes Q_1, …, Q_n, and
- each $i = 1, \dots, n$

the following statement holds:

$$\Big(P_i^{(1)}(Q_1, \dots, Q_n)\ ,\ P_i^{(2)}(Q_1, \dots, Q_n) \Big) \in \mu$$

Theorem 31.

Let μ be a congruence on the set of all processes.

For every couple of RDs of the form (5.7), which are equivalent with respect to μ, the processes defined by these RDs, i.e.

$$\{[\![A_i^{(1)}]\!] \mid i = 1, \dots, n\} \quad \text{and} \quad \{[\![A_i^{(2)}]\!] \mid i = 1, \dots, n\}$$

are also equivalent with respect to μ, i.e.

$$\forall\, i = 1, \dots, n \quad \Big([\![A_i^{(1)}]\!]\ ,\ [\![A_i^{(2)}]\!] \Big) \in \mu \quad \blacksquare$$

5.7 Transitions on PE

There is another way of defining of a correspondence between PEs and processes. This method is related to the concept of transitions on the set PE. Every such transition is a triple of the form (P, a, P'), where $P, P' \in PE$, and $a \in Act$. We shall represent a transition (P, a, P') by the diagram

$$P \xrightarrow{a} P' \tag{5.8}$$

We shall define the set of transitions on PE inductively, i.e.

- some transitions will be described explicitly, and
- other transitions will be described in terms of inference rules.

In this section we assume that each process is a value of some process constants.

Explicit transitions are defined as follows.

1. if P is a process constant, then

$$P \xrightarrow{a} P'$$

where P' is a process constant, such that

   - values of P and P' have the form

$$(S, s^0, R) \quad \text{and} \quad (S, s^1, R)$$

   respectively, and

   - R contains the transition $s^0 \xrightarrow{a} s^1$

2. $a.P \xrightarrow{a} P$, for any $a.P \in PE$

Inference rules for constructing of new transitions on PE from existing ones are defined as follows.

1. if $P \xrightarrow{a} P'$, then

   - $P + Q \xrightarrow{a} P'$, and
   - $Q + P \xrightarrow{a} P'$

- $P \,|\, Q \xrightarrow{a} P' \,|\, Q$, and
- $Q \,|\, P \xrightarrow{a} Q \,|\, P'$
- if $L \subseteq Names$, $a \neq \tau$, and $name(a) \notin L$, then

$$P \setminus L \xrightarrow{a} P' \setminus L$$

- for each renaming f

$$P[f] \xrightarrow{f(a)} P'[f]$$

2. if $a \neq \tau$, then from

$$P_1 \xrightarrow{a} P_1' \quad \text{and} \quad P_2 \xrightarrow{\bar{a}} P_2'$$

   it follows that

$$P_1 \,|\, P_2 \xrightarrow{\tau} P_1' \,|\, P_2'$$

3. For each RD (5.1) and each $i \in \{1, \ldots, n\}$

$$\begin{array}{ll} \text{if} & P_i \xrightarrow{a} P' \\ \text{then} & A_i \xrightarrow{a} P' \end{array} \tag{5.9}$$

For each PE $P \in PE$ a process $[\![P]\!]$, which corresponds to this PE, has the form

$$(PE, P, \mathcal{R})$$

where $\mathcal{R}$ is a set of all transitions on PE.

Theorem 32.
For each RD (5.1) and each $i = 1, \ldots, n$ the following statement holds

$$[\![A_i]\!] \sim P_i([\![A_1]\!]/A_1, \ldots, [\![A_n]\!]/A_n)$$

(i.e. the list of processes $[\![A_1]\!], \ldots, [\![A_n]\!]$ is a solution (with respect to $\sim$) of the system of equations which corresponds to RD (5.1). ■

5.8 A method of a proof of equivalence of processes with use of RDs

One of possible methods for proof of an equivalence ($\sim$ or $\overset{+}{\approx}$) between two processes consists of a construction of an appropriate RD such that both of these processes are components with the same numbers of some solutions of a system of equations related to this RD.

The corresponding equivalences are substantiated by theorem 33.

To formulate this theorem, we introduce the following auxiliary notion.

Let μ be a binary relation of the set of all processes, and let there is given an RD of the form (5.1).

A list of processes, defined by the RD, is said to be unique up to μ, if for each pair of lists of processes

$$(Q_1^{(1)}, \ldots, Q_n^{(1)}) \quad \text{and} \quad (Q_1^{(2)}, \ldots, Q_n^{(2)})$$

which satisfies to the condition

$$\begin{array}{l} \forall i = 1, \ldots, n \\ \quad (\ [\![Q_i^{(1)}]\!]\ ,\ P_i(Q_1^{(1)}/A_1, \ldots, Q_n^{(1)}/A_n)\ )\ \in \mu \\ \quad (\ [\![Q_i^{(2)}]\!]\ ,\ P_i(Q_1^{(2)}/A_1, \ldots, Q_n^{(2)}/A_n)\ )\ \in \mu \end{array}$$

the following statement holds:

$$\forall i = 1, \ldots, n \quad \Big(\ [\![Q_i^{(1)}]\!]\ ,\ [\![Q_i^{(2)}]\!]\ \Big)\ \in \mu$$

Theorem 33.

Let there is given a RD of the form (5.1).

1. If each occurrence of each process name A_i in each PE P_j is contained in a subexpression of the form $a.Q$, then a list of processes, which is defined by this RD, is unique up to $\sim$.

2. If

   - each occurrence of each process name A_i in each PE P_j is contained in a subexpression of the form $a.Q$, where $a \neq \tau$, and
   - each occurrence of each process name A_i in each PE P_j is contained only in subexpressions of the forms $a.Q$ and $Q_1 + Q_2$

   then a list of processes, defined by this RD, is unique up to $\overset{+}{\approx}$. ■

5.9 Problems related to RDs

1. Recognition of existence of finite processes that are equivalent (with respect to $\sim$, $\approx$, $\overset{+}{\approx}$) to processes of the form $[\![A]\!]$.

2. Construction of algorithms for finding minimal processes which are equivalent to processes of the form $[\![A]\!]$ in the case when these processes are finite.

3. Recognition of equivalence of processes of the form $[\![A]\!]$
(these processes can be infinite, and methods from chapter 4 are not appropriate for them).

4. Recognition of equivalence of RDs.

5. Finding necessary and sufficient conditions of uniqueness of a list of processes which is defined by a RD (up to $\sim$, $\overset{+}{\approx}$).

Chapter 6

Examples of a proof of properties of processes

6.1 Flow graphs

In this section we describe a notion of a flow graph, which is intended to enhance a visibility and to facilitate an understanding of a relationship between components of complex processes. Each example of a complex process, which is considered in this book, will be accompanied by a flow graph, which corresponds to this process.

Let $P_1, \ldots, P_n$ be a list of processes.

A **structural composition** of the processes P_1, …, P_n is an expression SC over processes, such that

- SC contains only processes from the list P_1, …, P_n, and
- each symbol of an operation, which consists in SC, is a symbol of one of the following operations:
  - parallel composition,
  - restriction,
  - renaming.

Each structural composition SC can be associated with a diagram, which is called a **flow graph (FG)** of SC.

A FG of a structural composition SC is defined by induction on a structure of SC as follows.

1. If SC consists of only a process P_i, then FG of SC is an oval, inside of which it is written an identifier of this process.

   On the border of this oval it is drawn circles, which are called **ports**.

   Each port corresponds to some input or output action $a \in Act(P_i)$, and

   - an identifier of this action is written near of the port, as a label of the port,
   - if a is an input action, then the port is white,
   - if a is an input action then the port is black.

   For every $a \in Act(P_i) \setminus \{\tau\}$ there is a unique port on the oval, such that its label is a.

2. If $SC = SC_1 \,|\, SC_2$, then a FG of SC is obtained by a disjoint union of FGs of SC_1 and SC_2, with drawing of labelled arrows on the disjoint union: for

   - every black port p_1 on one of these FGs, and
   - every white port p_2 on another of these FGs, such that labels of these ports are complementary actions

   it is drawn an arrow from p_1 to p_2 with a label $name(a)$, where a is a label of p_1.

3. If $SC = SC_1 \setminus L$, then a FG of SC is obtained from a FG of SC_1 by a removal of labels of ports, whose names belong to L.

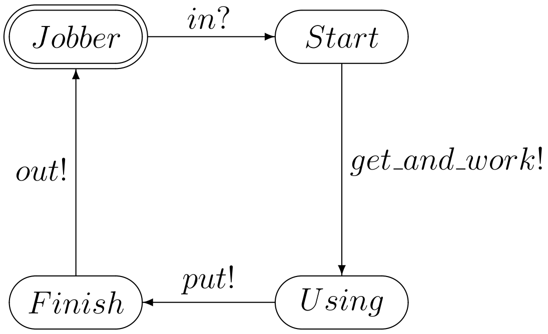
4. If $SC = SC_1\,[f]$, then a FG of SC is obtained from a FG of SC_1 by a corresponfing renaming of labels of ports.

If P is a process which is equal to a value of a structural composition SC, then the notation $FG(P)$ denotes a FG of SC.

6.2 Jobshop

Consider a model of a jobshop, which employs two workers, who use for working one mallet.

A behavior of each worker in the jobshop is described by the following process *Jobber*



where

- the actions *in*? and *out*! are used for interaction of a worker with a client, and denote
  - receiving of a material, and
  - issuance of a finished product

  respectively,
- actions *get_and_work*! and *put*! are used for interaction of a worker with a mallet and denote
  - taking a mallet and working with it, and
  - returning the mallet

  respectively.

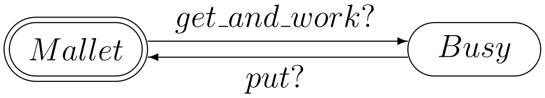
The action *get_and_work*! consists of several elementary actions. We do not detail them and combine them in one action.

According to the definition of the process *Jobber*, a worker works as follows:

- at first he accepts a material
- then he takes the mallet and works

- then he puts the mallet
- then he gives the finished product
- and all these actions are repeated.

A behavior of the mallet we present using the following process *Mallet*:



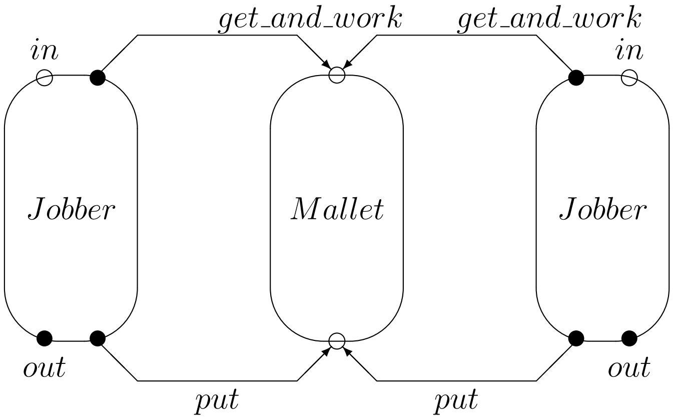
(note that the object "mallet" and the process "Mallet" are different concepts).

A behavior of the jobshop is described by the process *Jobshop*:

$$Jobshop = (Jobber \mid Jobber \mid Mallet) \setminus L$$

where $L = \{get\_and\_work,\ put\}$.

A flow graph of the process *Jobshop* has the following form.

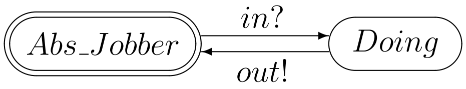


We now introduce the notion of an **abstract worker**, about whom we know that he cyclically

- accepts a material and
- gives finished products

but nothing is known about details of his work.

A behavior of the **abstract worker** we describe by the following process *Abs_Jobber*:



A behavior of an **abstract jobshop** we describe by the following process *Abs_Jobshop*:

$$Abs\_Jobshop = Abs\_Jobber \mid Abs\_Jobber$$

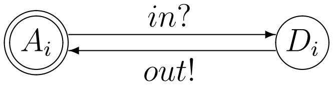
The process *Abs_Jobshop* is used as a **specification** of the jobshop. This process describes a behavior of the jobshop without details of its implementation.

Prove that the process *Jobshop* meets its specification, i.e.

$$Jobshop \overset{+}{\approx} Abs\_Jobshop \tag{6.1}$$

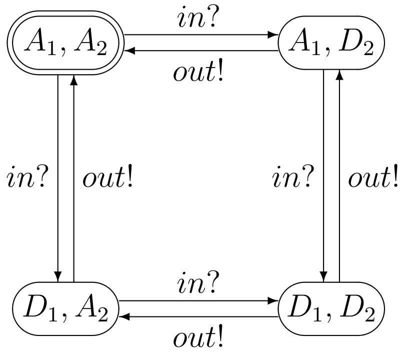
The process *Abs_Jobshop* is a parallel composition of two processes *Abs_Jobber*. In order to avoid conflicts with the notations, we choose different identifiers to refer the states of these processes.

Suppose, for example, that these processes have the form

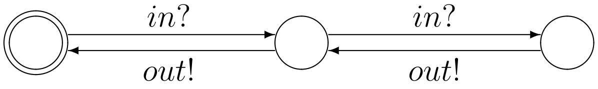


where $i = 1, 2$.

Parallel composition of these processes has the form



Applying to this process the procedure of minimization with respect to observational equivalence, we get the process



(6.2)

The process *Jobshop* has $4\cdot 4\cdot 2 = 32$ states, and we do not present it here because of its bulkiness. After a minimization of this process with respect to observational equivalence, we get a process, which is isomorphic to process (6.2). This means that the following statement holds:

$$Jobshop \ \approx \ Abs\_Jobshop \tag{6.3}$$

Because there is no transitions with a label τ, starting from initial states of processes

Jobshop and *Abs_Jobshop*

then on the reason of (6.3) we conclude that (6.1) holds.

6.3 Dispatcher

Suppose that

- there is some company which consists of several groups: G_1, ..., G_n, and

- there is a special room in the building, where the company does work, such that any group $G_i \quad (i \in \{1, \ldots, n\})$ can use this room to conduct their workshops.

There is a problem of non-conflictual use of the room by the groups G_1, …, G_n. This means that when one of the groups conducts a workshop in the room, other groups should be banned to hold their workshops in this room.

This problem can be solved by use of a special process, which is called a **dispatcher**.

If any group G_i wants to hold a workshop in this room, then G_i should send the dispatcher a request to provide a right to use the room for the workshop.

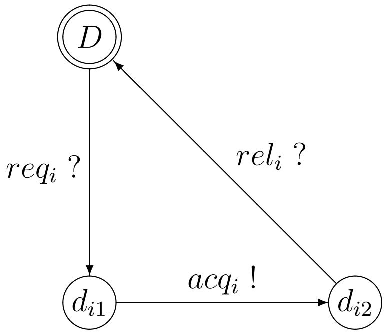
If the dispatcher knows that at this time the room is busy, then he don't allows G_i to use this room.

When the room becomes free, the dispatcher sends G_i a notice that he allows to the group G_i use this room.

After completion the workshop, the group G_i must send the dispatcher a notice that the room is free.

Consider a description of this system in terms of the theory of processes.

A behavior of the dispatcher is described by the process D, a graph representation of which consists of the following subgraphs: for each $i = 1, \ldots, n$ it contains the subgraph



i.e.

$$D \sim \sum_{i=1}^{n} req_i?.\ acq_i!.\ rel_i?.\ D$$

Actions from $Act(D)$ have the following meanings:

- $req_i\,?$ is a receiving of a request from the group G_i

- $acq_i\,!$ is a sending G_i of a notice that G_i may use the room
- $rel_i\,?$ is a receiving a message that G_i released the room.

In the following description of a behavior of each group G_i

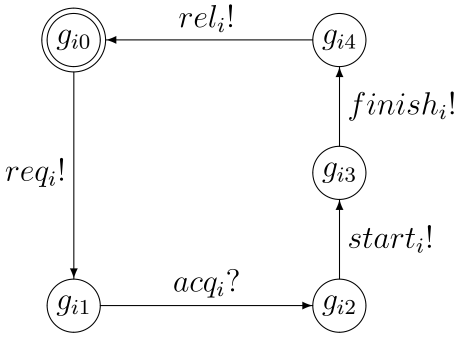
- we shall describe only an interaction of G_i
  - with the dispatcher, and
  - with the room

  and
- will not deal with other functions of G_i.

We shall denote

- a beginning of a workshop in the room by the action $start\,!$, and
- a completion of the meeting by the action of $finish\,!$.

A behavior of the group G_i we describe by a process G_i, which has the following graph representation:



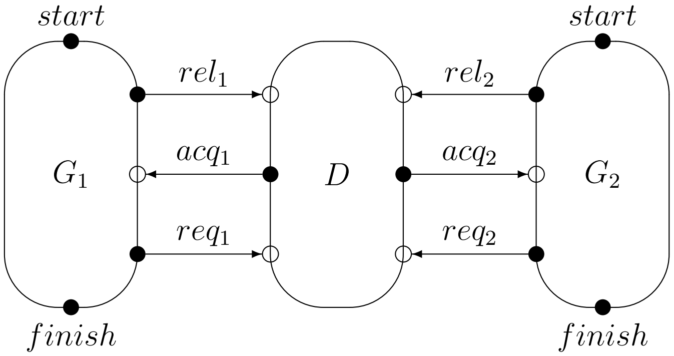
i.e. $G_i \sim req_i!.\ acq_i?.\ start!.\ finish!.\ rel_i!.\ G_i$.

A joint behavior of the dispatcher and the groups can be described as the following process Sys:

$$Sys = (D \,|\, G_1 \,|\, \dots \,|\, G_n) \setminus L$$

where $L = \{req_i, acq_i, rel_i \mid i = 1, \dots, n\}$.

A flow graph of the process Sys for $n = 2$ has the following form



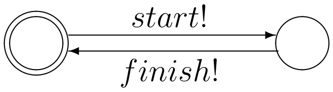
We now show that the processes which represent a behavior of the dispatcher and the groups indeed provide a conflict-free regime of use of the room.

The conflict-free property is that

- after a start of a workshop in the room of any group (i.e. after an execution the action $start!$ by this group), and
- before a completion of this workshop

there is no another group which also may hold a workshop in this room (i.e. which also can execute the action $start!$) until the first group has completed its workshop (i.e. until it has executed the action $finish!$).

Define a process $Spec$ as follows:



i.e. $Spec \sim start!.\, finish!.\, Spec.$

The conflict-free property of the regime of use of the room is equivalent to the following statement:

$$Sys \approx Spec \tag{6.4}$$

To prove this statement, we transform the process Sys, applying several times the expansion theorem:

$$
\begin{array}{l}
Sys \sim \\
\sim \sum\limits_{i=1}^{n} \tau. \left(\begin{array}{l} acq_i!.\ rel_i?.\ D \,|\, G_1 \,|\, \ldots \\ \ldots \,|\, acq_i?.\ start!.\ finish!.\ rel_i!.\ G_i \,|\, \ldots \\ \ldots \,|\, G_n \end{array} \right) \backslash L \sim \\
\sim \sum\limits_{i=1}^{n} \tau.\tau. \left(\begin{array}{l} rel_i?.\ D \,|\, G_1 \,|\, \ldots \\ \ldots \,|\, start!.\ finish!.\ rel_i!.\ G_i \,|\, \ldots \\ \ldots \,|\, G_n \end{array} \right) \backslash L \sim \\
\sim \sum\limits_{i=1}^{n} \tau.\tau.start!. \left(\begin{array}{l} rel_i?.\ D \,|\, G_1 \,|\, \ldots \\ \ldots \,|\, finish!.\ rel_i!.\ G_i \,|\, \ldots \\ \ldots \,|\, G_n \end{array} \right) \backslash L \sim \\
\sim \sum\limits_{i=1}^{n} \tau.\tau.start!.\ finish!. \left(\begin{array}{l} rel_i?.\ D \,|\, G_1 \,|\, \ldots \\ \ldots \,|\, rel_i!.\ G_i \,|\, \ldots \\ \ldots \,|\, G_n \end{array} \right) \backslash L \sim \\
\sim \sum\limits_{i=1}^{n} \tau.\tau.start!.\ finish!.\ \tau. \underbrace{\left(\begin{array}{l} D \,|\, G_1 \,|\, \ldots \\ \ldots \,|\, G_i \,|\, \ldots \\ \ldots \,|\, G_n \end{array} \right) \backslash L}_{Sys} = \\
= \sum\limits_{i=1}^{n} \tau.\tau.start!.\ finish!.\ \tau.Sys
\end{array}
$$

Using the rules

$$P + P \sim P \quad \text{and} \quad \alpha.\tau.P \overset{+}{\approx} \alpha.P$$

we get the statement

$$Sys \overset{+}{\approx} \tau.start!.\ finish!.\ Sys$$

We now consider the equation

$$X = \tau.start!.\ finish!.\ X \tag{6.5}$$

According to theorem 33 from section 5.8, there is a unique (up to $\overset{+}{\approx}$) solution of equation (6.5) .

As shown above, the process Sys is a solution of (6.5) up to $\overset{+}{\approx}$.

The process $\tau.Spec$ is also a solution of (6.5) up to $\overset{+}{\approx}$, because

$$\begin{array}{l} \tau.Spec \sim \tau.start!.\,finish!.\,Spec \overset{+}{\approx} \\ \overset{+}{\approx} \tau.start!.\,finish!.\,(\tau.Spec) \end{array}$$

Consequently, the following statement hold:

$$Sys \overset{+}{\approx} \tau.Spec$$

This statement implies (6.4).

6.4 Scheduler

Suppose that there are n processes

$$P_1, \ldots, P_n \tag{6.6}$$

and for each $i = 1, \ldots, n$ the set $Act(P_i)$ contains two special actions:

- the action $\alpha_i?$, which can be interpreted as a signal

$$P_i \textit{ starts its regular session} \tag{6.7}$$

- the action $\beta_i?$, which can be interpreted as a signal

$$P_i \textit{ completes its regular session} \tag{6.8}$$

We assume that

- all the names
$$\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n \tag{6.9}$$
  are different, and

- $\forall\, i = 1, \ldots, n$ each name from

$$names(Act(P_i)) \setminus \{\alpha_i, \beta_i\}$$

  does not belong to the set (6.9).

Let L be the set (6.9).

For each $i = 1, \ldots, n$ the actions from the set

$$Act(P_i) \setminus \{\alpha_i?, \beta_i?\}$$

are said to be **proper actions** of the process P_i.

An arbitrary trace of each process P_i may contain any quantity of the actions $\alpha_i?$ and $\beta_i?$ in any order.

We would like to create a new process P, in which all the processes P_1, …, P_n would work together, and this joint work should obey certain regime.

The process P must have the form

$$P = (P_1 \,|\, \ldots \,|\, P_n \,|\, Sch) \setminus L$$

where the process Sch

- is called a **scheduler**, and
- is designed for an establishing of a required regime of an execution of the processes P_1, …, P_n.

Non-internal actions, which may be executed by the process Sch, must belong to the set

$$\{\alpha_1!, \ldots, \alpha_n!, \beta_1!, \ldots, \beta_n!\} \tag{6.10}$$

By the definition of the process P, for each $i = 1, \ldots, n$

- the actions $\alpha_i?$ and $\beta_i?$ can be executed by the process $P_i \in (6.6)$ within the process P only simultaneously with an execution of complementary actions by the process Sch, and
- an execution of these actions will be invisible outside the process P.

Informally speaking, each process P_i, which is executed within the process P, may start or complete its regular session if and only if the scheduler Sch allows him to do it.

A regime, which must be respected by the processes P_1, …, P_n, during their execution within the process P, consists of the following two conditions.

1. For each $i = 1, \ldots, n$ an arbitrary trace of the process P_i, which is executed within the process P, should have the form

$$\alpha_i? \;\ldots\; \beta_i? \;\ldots \alpha_i? \;\ldots\; \beta_i? \;\ldots$$

(where the dots represent proper actions of the process P_i), i.e. an execution of the process P_i should be a sequence of sessions of the form

$$\alpha_i? \ \dots \ \beta_i? \ \dots$$

where each session

- starts with an execution of the action $\alpha_i?$
- then several proper actions of P_i are executed,
- after a completion of the session the action $\beta_i?$ is executed, and
- then P_i can execute some proper actions
  (for example, these actions can be related to a preparation to the next session).

2. The processes $P_1, \dots, P_n$ are obliged to start their new sessions in rotation, i.e.

   - at first, only P_1 may start its first session
   - then, P_2 may start its first session
   - ...
   - then, P_n may start its first session
   - then, P_1 may start its second session
   - then, P_2 may start its second session
   - etc.

Note that we do not require that each process P_i may receive a permission to start its k-th session only after the previous process P_{i-1} completes its k-th session. However, we require that each process P_i may receive a permission to start a new session, only if P_i executed the action $\beta_i?$ (which signalizes a completion of a previous session of P_i).

Proper actions of the processes $P_1, \dots, P_n$ can be executed in arbitrary order, and it is allowably an interaction of these processes during their execution within the process P.

The described regime can be formally expressed as the following two conditions on an arbitrary trace

$$tr \in Tr(Sch)$$

In these conditions we shall use the following notation: if

$$tr \in Tr(Sch) \quad \text{and} \quad M \subseteq Act$$

then $tr\,|_{M}$ denotes a sequence of actions, which is derived from tr by a removal of all actions which do not belong to M.

Conditions which describe the above regime have the following form:

$$\begin{array}{l} \forall\, tr \in Tr(Sch),\ \forall\, i = 1, \ldots, n \\ tr\,|_{\{\alpha_i, \beta_i\}} \ = \ (\alpha_i!\ \beta_i!\ \alpha_i!\ \beta_i!\ \alpha_i!\ \beta_i!\ \ldots) \end{array} \tag{6.11}$$

and

$$\begin{array}{l} \forall\, tr \in Tr(Sch) \\ tr\,|_{\{\alpha_1, \ldots, \alpha_n\}} \ = \ (\alpha_1!\ \ldots\ \alpha_n!\ \alpha_1!\ \ldots\ \alpha_n!\ \ldots) \end{array} \tag{6.12}$$

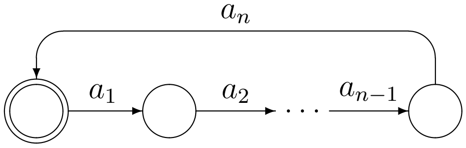
These conditions can be expressed as observational equivalence of certain processes.

To define these processes, we introduce auxiliary notations.

1. Let $a_1 \ldots a_n$ be a sequence of actions from Act. Then the string

   $$(a_1 \ldots a_n)^*$$

   denotes a process which has the following graph representation



2. Let P be a process, and

   $$\{a_1, \ldots, a_k\} \subseteq Act \setminus \{\tau\} \tag{6.13}$$

   be a set of actions.

   The string

   $$hide\ (P, a_1, \ldots, a_k) \tag{6.14}$$

   denotes the process

   $$(\ P\,|\,(\,\overline{a_1}\,)^*\,|\ \ldots\ |\,(\,\overline{a_k}\,)^*\ )\setminus names(\{a_1, \ldots, a_k\})$$

Process (6.14) can be considered as a process, which is obtained from P by a replacement on τ of all labels of transitions of P, which belong to the set (6.13).

Using these notations,

- condition (6.11) can be expressed as follows: for each $i = 1, \dots n$

$$\begin{array}{l} \mathit{hide}\left(\begin{array}{cc} \mathit{Sch}, & \alpha_1!, \dots, \alpha_{i-1}!, \alpha_{i+1}!, \dots, \alpha_n! \\ & \beta_1!, \dots, \beta_{i-1}!, \beta_{i+1}!, \dots, \beta_n! \end{array}\right) \approx \\ \approx (\alpha_i!.\ \beta_i!)^* \end{array} \tag{6.15}$$

  and

- condition (6.12) can be expressed as follows:

$$\mathit{hide}\,(\mathit{Sch}, \beta_1!, \dots, \beta_n!) \approx (\alpha_1!.\ \dots \alpha_n!)^* \tag{6.16}$$

It is easy to see that there are several schedulers that satisfy these conditions. For example, the following schedulers satisfy these conditions:

- $\mathit{Sch} \ = \ (\alpha_1!\ \beta_1!\ \dots\ \alpha_n!\ \beta_n!)^*$
- $\mathit{Sch} \ = \ (\alpha_1!\ \dots\ \alpha_n!\ \beta_1!\ \dots\ \beta_n!)^*$

However, these schedulers impose too large restrictions on an execution of the processes $P_1, \dots, P_n$.

We would like to construct such a scheduler that allows a maximal freedom of a joint execution of the processes $P_1, \dots, P_n$ within the process P.

This means that if at any time

- the process P_i has an intention to execute an action $a \in \{\alpha_i?, \beta_i?\}$, and
- this intention of the process P_i does not contradict to the regime which is described above

then the scheduler should not prohibit P_i to execute this action at the current time, i.e. the action $\overline{a}$ must be among actions, which the scheduler can execute at the current time.

The above informal description of a maximal freedom of an execution of a scheduler can be formally clarified as follows:

- each state s of the scheduler be associated with a pair (i, X), where

  - $i \in \{1, \ldots, n\}$, i is a number of a process, which has the right to start its regular session at the current time
  - $X \subseteq \{1, \ldots, n\} \setminus \{i\}$, X is a set of active processes at the current time
    (a process is said to be active, if it started its regular session, but does not completed it yet)

- an initial state of the scheduler is associated with a pair $(1, \emptyset)$

- a set of transitions of the scheduler consists of

  - transitions of the form

$$s \xrightarrow{\alpha_i!} s'$$

    where

    - s is associated with (i, X)
    - s' is associated with $(next(i), X \cup \{i\})$, where
      $next(i) \overset{\text{def}}{=} \begin{cases} i+1, & \text{if } i < n, \text{ and} \\ 1, & \text{if } i = n \end{cases}$

  - and transitions of the form

$$s \xrightarrow{\beta_j!} s'$$

    where

    - s is associated with (i, X),
    - s' is associated with $(i, X \setminus \{j\})$, where $j \ \in X$

The above description of properties of a required scheduler can be considered as its definition, i.e. we can define a required scheduler as a process Sch_0 with the following components:

- a set of its states is the set of pairs of the form

$$\{(i, X) \in \{1, \ldots, n\} \times \ \mathcal{P}(\{1, \ldots, n\}) \mid i \notin X\}$$

- an initial state and transitions of Sch_0 are defined as it was described above.

The definition of the scheduler Sch_0 has a significant deficiency: a size of the set of states of Sch_0 exponentially depends on the number of processes (6.6), that does not allow quickly modify such scheduler in the case when the set of processes (6.6) is changed.

We can use Sch_0 only as an reference, with which we will compare other schedulers.

To solve the original problem we define another scheduler Sch. We will describe it

- not by explicit description of its states and transitions, but
- by setting of a certain expression, which describes Sch in terms of a composition of several simple processes.

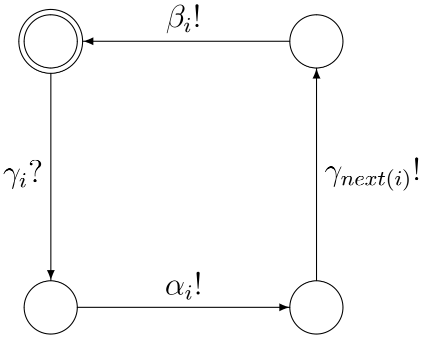
In the description of the scheduler Sch we shall use new names γ_1, $\ldots$, γ_n. Denote the set of these names by the symbol Γ.

Process Sch is defined as follows:

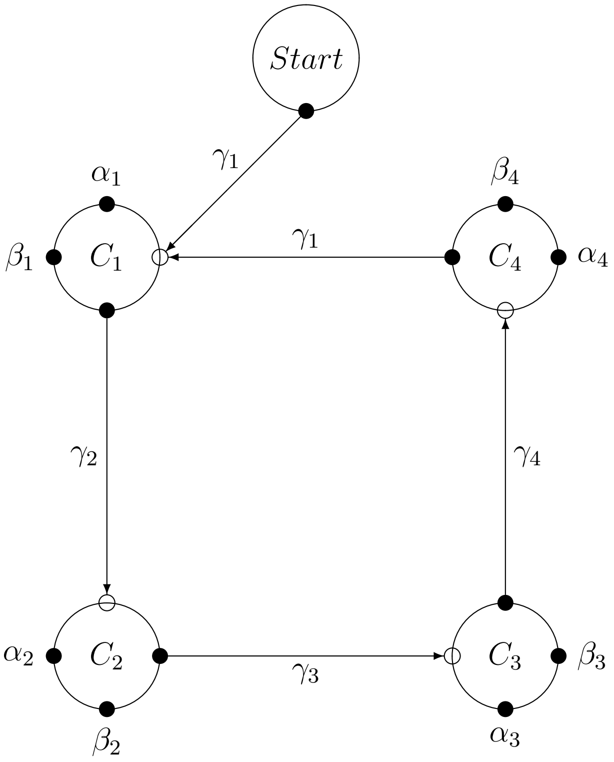
$$Sch \overset{\text{def}}{=} (Start \,|\, C_1 \,|\, \ldots \,|\, C_n) \setminus \Gamma \tag{6.17}$$

where

- $Start \overset{\text{def}}{=} \gamma_1!.\, \mathbf{0}$
- for each $i = 1, \ldots, n$ the process C_i is called a **cycler** and has the form



A flow graph of *Sch* in the case $n = 4$ has the following form:



We give an informal explanation of an execution of the process *Sch*.

The cycler C_i is said to be

- **disabled** if it is in its initial state, and
- **enabled**, if it is not in its initial state.

The process *Start* enables the first cycler C_1 and then "dies".

Each cycler C_i is responsible for an execution of the process P_i. The cycler C_i

- enables the next cycler $C_{next(i)}$ after he gave a permission to the process P_i to start a regular session, and
- becomes disabled after he gave a permission to the process P_i to complete a regular session.

Prove that process (6.17) satisfies condition (6.16) (we omit checking of condition (6.15)).

According to the definition of process (6.14), condition (6.16) has the form

$$(Sch \,|\, (\beta_1?)^* \,|\, \ldots \,|\, (\beta_n?)^*) \setminus B \approx (\alpha_1!. \ \ldots \alpha_n!)^* \tag{6.18}$$

where $B = \{\beta_1, \ldots, \beta_n\}$.

Let Sch' be the left side of (6.18).

Prove that

$$Sch' \overset{+}{\approx} \tau.\alpha_1!. \ \ldots \alpha_n!. \ Sch' \tag{6.19}$$

Hence by the uniqueness property (with respect to $\overset{+}{\approx}$) of a solution of the equation

$$X \ = \ \tau.\alpha_1!. \ \ldots \alpha_n!. \ X$$

we get the statement

$$Sch' \ \overset{+}{\approx} \ (\tau \ \alpha_1! \ \ldots \alpha_n! \ )^*$$

which implies (6.18).

We will convert the left side of the statement (6.19) so as to obtain the right side of this statement. To do this, we will use properties 8, 11 and 12 of operations on processes, which are contained in section 3.7. We recall these properties:

- $P \setminus L = P$, if $L \cap names(Act(P)) = \emptyset$
- $(P_1 \,|\, P_2) \setminus L = (P_1 \setminus L) \,|\, (P_2 \setminus L)$, if
$$L \cap names(Act(P_1) \cap \overline{Act(P_2)}) = \emptyset$$
- $(P \setminus L_1) \setminus L_2 = P \setminus (L_1 \cup L_2) = (P \setminus L_2) \setminus L_1$

Using these properties, it is possible to convert the left side of (6.19) as follows.

$$\begin{array}{l} Sch' = \\ = (Sch \,|\, (\beta_1?)^* \,|\, \ldots \,|\, (\beta_n?)^*) \setminus B = \\ = \left(\begin{array}{l} ((Start \,|\, C_1 \,|\, \ldots \,|\, C_n) \setminus \Gamma) \,| \\ \,|\, (\beta_1?)^* \,|\, \ldots \,|\, (\beta_n?)^* \end{array} \right) \setminus B = \\ = (Start \,|\, C'_1 \,|\, \ldots \,|\, C'_n) \setminus \Gamma \end{array} \tag{6.20}$$

where

$$C'_i = (C_i \,|\, (\beta_i?)^*) \setminus \{\beta_i\}$$

Note that for each $i = 1, \dots, n$ the following statement holds:

$$C_i' \overset{+}{\approx} \gamma_i?.\ \alpha_i!.\ \gamma_{next(i)}!.\ C_i' \tag{6.21}$$

Indeed, by the expansion theorem,

$$\begin{aligned} &C_i' = ((\gamma_i?.\ \alpha_i!.\ \gamma_{next(i)}!.\ \beta_i!.\ C_i) \,|\, (\beta_i?)^*) \setminus \{\beta_i\} \sim \\ &\sim \gamma_i?.\ \alpha_i!.\ \gamma_{next(i)}!.\ \tau.C_i' \overset{+}{\approx} \text{right side of (6.21)} \end{aligned}$$

Using this remark and the expansion theorem, we can continue the chain of equalities (6.20) as follows:

$$\begin{aligned} &(Start \,|\, C_1' \,|\, C_2' \,|\, \dots \,|\, C_n') \setminus \Gamma \overset{+}{\approx} \\ &\overset{+}{\approx} (\underbrace{\gamma_1!.\ \mathbf{0}}_{=Start} \,|\, \underbrace{\gamma_1?.\ \alpha_1!.\ \gamma_2!.\ C_1'}_{\overset{+}{\approx} C_1'} \,|\, C_2' \,|\, \dots \,|\, C_n') \setminus \Gamma \sim \\ &\sim \tau.\ (\mathbf{0} \,|\, \alpha_1!.\ \gamma_2!.\ C_1' \,|\, C_2' \,|\, \dots \,|\, C_n') \setminus \Gamma = \\ &= \tau.\ (\alpha_1!.\ \gamma_2!.\ C_1' \,|\, C_2' \,|\, \dots \,|\, C_n') \setminus \Gamma \sim \\ &\sim \tau.\ \alpha_1!.\ (\gamma_2!.\ C_1' \,|\, C_2' \,|\, \dots \,|\, C_n') \setminus \Gamma \overset{+}{\approx} \\ &\overset{+}{\approx} \tau.\ \alpha_1!.\ (\gamma_2!.\ C_1' \,|\, \underbrace{\gamma_2?.\ \alpha_2!.\ \gamma_3!.\ C_2'}_{\overset{+}{\approx} C_2'} \,|\, \dots \,|\, C_n') \setminus \Gamma \sim \\ &\sim \tau.\ \alpha_1!.\ \tau.\ (C_1' \,|\, \alpha_2!.\ \gamma_3!.\ C_2' \,|\, \dots \,|\, C_n') \setminus \Gamma \sim \dots \sim \\ &\sim \tau.\ \alpha_1!.\ \tau.\ \alpha_2!.\ \dots \tau.\ \alpha_n!.\ (C_1' \,|\, \dots \,|\, \gamma_1!.\ C_n') \setminus \Gamma \overset{+}{\approx} \\ &\overset{+}{\approx} \tau.\ \alpha_1!.\ \dots \alpha_n!.\ (C_1' \,|\, \dots \,|\, \gamma_1!.\ C_n') \setminus \Gamma \overset{+}{\approx} \\ &\overset{+}{\approx} \tau.\ \alpha_1!.\ \dots \alpha_n!.\ (\ \underbrace{\gamma_1?.\ \alpha_1!.\ \gamma_2!.\ C_1'}_{\overset{+}{\approx} C_1'} \,|\, \dots \,|\, \gamma_1!.\ C_n'\ ) \setminus \Gamma \sim \\ &\sim \tau.\ \alpha_1!.\ \dots \alpha_n!.\ \underbrace{\tau.\ (\alpha_1!.\ \gamma_2!.\ C_1' \,|\, \dots \,|\, C_n') \setminus \Gamma} \end{aligned} \tag{6.22}$$

The underlined expression on the last line of the chain coincides with an expression on the fourth line of the chain, which is observationally congruent to Sch'.

We have found that the last expression of the chain (6.22) is observationally congruent to the left side and to the right side of (6.19).

Thus, the statement (6.19) is proven. ■

A reader is provided as an exercise the following problems.

1. To prove

   - condition (6.15), and
   - the statement $Sch \approx Sch_0$,

2. To define and verify a scheduler that manages a set P_1, ..., P_n of **processes with priorities**, in which each process P_i is associated with a certain **priority**, representing a number $p_i \in [0, 1]$, where

   $$\sum_{i=1}^{n} p_i = 1$$

   The scheduler must implement a regime of a joint execution of the processes P_1, ..., P_n with the following properties:

   - for each $i = 1, \ldots, n$ a proportion of a number of sessions which are completed by the process P_i, relative to the total number of sessions which are completed by all processes P_1, ..., P_n, must asymptotically approximate to p_i with an infinite increasing of a time of an execution of of the processes P_1, ..., P_n
   - this scheduler should provide a maximal freedom of an execution of the processes $P_1, \ldots, P_n$.

6.5 Semaphore

Let $P_1, \ldots, P_n$ be a list of processes, and for each $i = 1, \ldots, n$ the process P_i has the following form:

$$P_i \;=\; (\alpha_i?\; a_{i1}\; \ldots\; a_{ik_i}\; \beta_i?)^*$$

where

- α_i? and β_i? are special actions representing signals that

  - P_i started an execution of a regular session, and
  - P_i completed an execution of a regular session

  respectively, and

- $a_{i1}, \ldots, a_{ik_i}$ are proper actions of the process P_i.

We would like to create such a process P, in which all the processes P_1, ..., P_n would work together, and this joint work should obey the following regime:

- if at some time of an execution of the process P any process P_i started its regular session (by an execution of the action $\alpha_i\,?$)
- then this session must be **uninterrupted** i.e. all subsequent action of the process P shall be actions of the process P_i, until P_i complete this session (by an execution of the action $\beta_i\,?$).

This requirement can be expressed in terms of traces: each trace of the process P must have the form

$$\alpha_i?\; a_{i1}\; \ldots\; a_{ik_i}\; \beta_i?\; \alpha_j?\; a_{j1}\; \ldots\; a_{jk_j}\; \beta_i?\; \ldots$$

i.e. each trace tr of the process P must be a concatenation of traces

$$tr_1 \;\cdot\; tr_2 \;\cdot\; tr_3 \;\ldots$$

where each trace tr_i in this concatenation represents a session of any process from the list P_1, ..., P_n.

A required process P we define as follows:

$$P \overset{\text{def}}{=} (\; P_1[f_1] \mid \ldots \mid P_n[f_n] \mid Sem \;) \setminus \{\pi, \varphi\}$$

where

- Sem is a special process designed to establish the required regime of an execution of the processes P_1, ..., P_n, this process
  - is called a **semaphore**, and
  - has the form
$$Sem \;=\; (\;\pi!\;\varphi!\;)^*$$
- $f_i : \alpha_i \mapsto \pi,\; \beta_i \mapsto \varphi$

A **specification** of the process P is represented by the following statement:

$$\begin{array}{rcl} P & \overset{+}{\approx} & \tau.a_{11}.\ldots a_{1k_1}.\,P\ +\ldots+ \\ & + & \tau.a_{n1}.\ldots a_{nk_n}.\,P \end{array} \tag{6.23}$$

A proof that the process P meets this specification, is performed by means of the expansion theorem:

$$\begin{array}{l} P\ =\ (\ P_1[f_1]\ \mid \ldots \mid\ P_n[f_n]\ \mid\ Sem\ ) \setminus \{\pi,\varphi\} \sim \\ \sim \left(\begin{array}{l} \pi?.a_{11}.\ldots.a_{1k_1}.\varphi?.P_1[f_1]\ \mid \ldots \mid \\ \mid\ \pi?.a_{n1}.\ldots.a_{nk_n}\varphi?.P_n[f_n]\ \mid \\ \mid\ \pi!.\ \varphi!.\ Sem \end{array} \right) \setminus \{\pi,\varphi\} \sim \\ \sim\ \tau. \left(\begin{array}{l} a_{11}.\ldots.a_{1k_1}.\varphi?.P_1[f_1]\ \mid \ldots \mid \\ \mid\ \pi?.a_{n1}.\ldots.a_{nk_n}\varphi?.P_n[f_n]\ \mid \\ \mid\ \varphi!.\ Sem \end{array} \right) \setminus \{\pi,\varphi\} + \\ +\ldots+ \\ +\tau. \left(\begin{array}{l} \pi?.a_{11}.\ldots.a_{1k_1}.\varphi?.P_1[f_1]\ \mid \ldots \mid \\ \mid\ a_{n1}.\ldots.a_{nk_n}\varphi?.P_n[f_n]\ \mid \\ \mid\ \varphi!.\ Sem \end{array} \right) \setminus \{\pi,\varphi\} \sim \\ \sim \ldots \sim \\ \sim \tau.a_{11}.\ldots a_{1k_1}.\tau.\ P\ +\ldots+\tau.a_{n1}.\ldots a_{nk_n}.\tau.\ P \overset{+}{\approx} \\ \overset{+}{\approx} \tau.a_{11}.\ldots a_{1k_1}.\ P\ +\ldots+\tau.a_{n1}.\ldots a_{nk_n}.\ P\ \blacksquare \end{array}$$

Finally, pay attention to the following aspect. The prefix "τ." in each summand of the right side of (6.23) means that a choice of a variant of an execution of the process P at the initial time is determined

- not by an environment of the process P, but
- by the process P itself.

If this prefix was absent, then it would mean that a choice of a variant of an execution of the process P at the initial time is determined by an environment of the process P.

Chapter 7

Processes with a message passing

7.1 Actions with a message passing

The concept of a process which was introduced and studied in previous chapters, can be generalized in different ways.

One of such generalizations consists of an addition to actions from *Act* some **parameters** (or **modalities**), i.e. there are considered processes with actions of the form

$$(a, p)$$

where $a \in Act$, and p is a parameter which may have the following meanings:

- a complexity (or a cost) of an execution of the action a
- a priority (or a desirability, or a plausibility) of the action a with respect to other actions
- a time (or an interval of time) at which the action a was executed
- a probability of an execution of the action a
- or anything else.

In this chapter we consider a variant of such generalization, which is related to an addition of **messages** to actions from *Act*. These messages are transmitted together with an execution of the actions.

Recall our informal interpretation of the concept of an execution of an action:

- the action $\alpha\,!$ is executed by sending of an object whose name is α, and
- the action $\alpha\,?$ is executed by receiving of an object whose name is α.

We generalize this interpretation as follows. We shall assume that processes can send or receive not only objects, but also pairs of the form

$$\text{(object, message)}$$

i.e. an action may have the form

$$\alpha\,!\,v \quad \text{and} \quad \alpha\,?\,v \tag{7.1}$$

where $\alpha \in Names$, and v is a **message**, that can be

- a string of symbols,
- a material resource,
- a bill,
- etc.

An execution of the actions $\alpha\,!\,v$ and $\alpha\,?\,v$, consists of sending or receiving the object α with the message v.

Recall that such entities as

- a transferred object, and
- receiving and sending of objects

can have a virtual character (more details see in section 2.3).

For a formal description of processes that can execute actions of the form (7.1), we generalize the concept of a process.

7.2 Auxiliary concepts

7.2.1 Types, variables, values and constants

We assume that there is given a set $Types$ of **types**, and each type $t \in Types$ is associated with a set D_t of **values** of the type t.

Types can be denoted by identifiers. We shall use the following identifiers:

- the type of integers is denoted by `int`
- the type of boolean values (0 and 1) is denoted by `bool`
- the type of messages is denoted by `mes`
- the type of lists of messages is denoted by `list`.

Also, we assume that there are given the following sets.

1. The set Var, whose elements are called **variables**.

   Every variable $x \in Var$

   - is associated with a type $t(x) \in Types$, and
   - can take **values** in the set $D_{t(x)}$, i.e. at different times the variable x can be associated with various elements of the set $D_{t(x)}$.

2. The set Con, whose elements are called **constant**.

   Every constant $c \in Con$ is associated with

   - a type $t(c) \in Types$, and
   - a value $[\![c]\!] \in D_{t(c)}$, which is said to be an **interpretation** of the constant c.

7.2.2 Functional symbols

We assume that there is given a set of **functional symbols (FSs)**, and each FS f is associated with

- a **functional type** $t(f)$, which is a list of the form

  $$(t_1, \ldots, t_n) \to t \tag{7.2}$$

  where $t_1, \ldots, t_n, t \in Types$, and

- a function
$$[\![f]\!] : D_{t_1} \times \ldots \times D_{t_n} \to D_t$$
which is called an **interpretation** of the FS f.

Examples of FSs:
$$+, \quad -, \quad \cdot, \quad \textbf{head}, \quad \textbf{tail}, \quad [\,]$$
where

- the FSs $+$ and $-$ have the functional type
$$(\texttt{int}, \texttt{int}) \to \texttt{int}$$
the functions $[\![+]\!]$ and $[\![-]\!]$ are the corresponding arithmetic operations

- the FS $\cdot$ has the functional type
$$(\texttt{list}, \texttt{list}) \to \texttt{list}$$
the function $[\![\cdot]\!]$ maps each pair of lists (u, v) to their **concatenation** (which is obtained by writing v on the right from u)

- the FS **head** has the functional type
$$\texttt{list} \to \texttt{mes}$$
the function $[\![\textbf{head}]\!]$ maps each nonempty list to its first element
(a value of $[\![\textbf{head}]\!]$ on an empty list can be any)

- the FS **tail** has the functional type
$$\texttt{list} \to \texttt{list}$$
the function $[\![\textbf{tail}]\!]$ maps each nonempty list u to the list which is derived from u by a removing of its first element
(a value of $[\![\textbf{tail}]\!]$ on an empty list can be any)

- the FS $[\,]$ has the functional type
$$\texttt{mes} \to \texttt{list}$$
the function $[\![\,[\,]\,]\!]$ maps each message to the list which consists only of this message

- the FS **length** has the functional type

  $$\texttt{list} \to \texttt{int}$$

  the function $[\![\textbf{length}]\!]$ maps each list to its length
  (a length of a list is a number of messages in this list)

7.2.3 Expressions

Expressions consist of variables, constants, and FSs, and are constructed by a standard way. Each expression e has a type $t(e) \in Types$, which is defined by a structure of this expression.

Rules of constructing of expressions have the following form.

- Each variable or constant is an expression of the type that is associated with this variable or constant.

- If

  - f is a FS of the functional type (7.2), and
  - $e_1, \ldots, e_n$ are expressions of the types $t_1, \ldots, t_n$ respectively

  then the list $f(e_1, \ldots, e_n)$ is an expression of the type t.

Let e be an expression. If each variable x occurred in e is associated with a value $\xi(e)$, then the expression e can be associated with a value $\xi(e)$ which is defined by a standard way:

- if $e = x \in Var$, then $\xi(e) \overset{\text{def}}{=} \xi(x)$
  (the value $\xi(x)$ is assumed to be given)

- if $e = c \in Con$, then $\xi(e) \overset{\text{def}}{=} [\![c]\!]$

- if $e = f(e_1, \ldots, e_n)$, then

  $$\xi(e) \overset{\text{def}}{=} [\![f]\!](\xi(e_1), \ldots, \xi(e_n))$$

Below we shall use the following notations.

- The symbol $\mathcal{E}$ denotes the set of all expressions.

- The symbol $\mathcal{B}$ denotes the set of expressions of the type `bool`.

  Expressions from $\mathcal{B}$ are called **formulas**.

  In constructing of formulas may be used boolean connectives ($\neg, \wedge, \vee$, etc.) interpreted by the standard way.

  The symbol $\top$ denotes a true formula, and the symbol $\bot$ denotes a false formula.

  Formulas of the form $\wedge(b_1, b_2)$, $\vee(b_1, b_2)$, etc. we shall write in a more familiar form $b_1 \wedge b_2$, $b_1 \vee b_2$, etc.

  In some cases, formulas of the form

  $$b_1 \wedge \ldots \wedge b_n \quad \text{and} \quad b_1 \vee \ldots \vee b_n$$

  will be written in the form

  $$\left\{ \begin{array}{c} b_1 \\ \ldots \\ b_n \end{array} \right\} \quad \text{and} \quad \left[\begin{array}{c} b_1 \\ \ldots \\ b_n \end{array} \right]$$

  respectively.

- Expressions of the form $+(e_1, e_2)$, $-(e_1, e_2)$ and $\cdot(e_1, e_2)$ will be written in a more familiar form $e_1 + e_2$, $e_1 - e_2$ and $e_1 \cdot e_2$.

- Expressions of the form $\mathbf{head}(e)$, $\mathbf{tail}(e)$, $[\,](e)$, and $\mathbf{length}(e)$ will be written in the form $\hat{e}$, e', $[e]$ and $|e|$, respectively.

- A constant of the type `list`, such that $[\![c]\!]$ is an empty list, will be denoted by the symbol ε.

7.3 A concept of a process with a message passing

In this section we present a concept of a process with a message passing. This concept is derived from the original concept of a process presented in section 2.4 by the following modification.

- Among the components of a process P there are the following additional components:

  - the component X_P, which is called a **set of variables** of the process P, and
  - the component I_P, which is called an **initial condition** of the process P.

- Transitions are labelled not by actions, but by **operators**.

Before giving a formal definition of a process with a message passing, we shall explain a meaning of the above concepts.

For brevity, in this chapter we shall call processes with message passing simply as **processes**.

7.3.1 A set of variables of a process

We assume that each process P is associated with a set of variables

$$X_P \subseteq Var$$

At any time i of an execution of a process P ($i = 0, 1, 2, \ldots$) each variable $x \in X_P$ is associated with a **value** $\xi_i(x) \in D_{t(x)}$. Values of the variables may be modified during an execution of the process.

An **evaluation** of variables from X_P is a family ξ of values associated with these variables, i.e.

$$\xi = \{\xi(x) \in D_{t(x)} \mid x \in X_P\}$$

The notation $Eval(X_P)$ denotes a set of all evaluations of variables from X_P.

For each time $i \geq 0$ of an execution of a process P the notation ξ_i denotes an evaluation of variables from X_P at this time.

Below we shall assume that for each process P all expressions referring to the process P, contain variables only from the set X_P.

7.3.2 An initial condition

Another new component of a process P is a formula $I_P \in \mathcal{B}$, which is called an **initial condition**. This formula expresses a condition on evaluation ξ_0 of variables from X_P at the initial time of an execution of P: ξ_0 must satisfy the condition

$$\xi_0(I_P) = 1$$

7.3.3 Operators

The main difference between the new definition of a process and the old one is that

- in the old definition a label of each transition is an **action** which is executed by a process of a performance of this transition, and
- in the new definition a label of each transition is an **operator** i.e. a **scheme of an action**, which takes a specific form only at the time of a performance of this transition.

In a definition of an operator we shall use the same set $Names$, which was introduced in section 2.3.

A set of all operators is divided into the following four classes.

1. **Input operators**, which have the form

   $$\alpha\,?\,x \tag{7.3}$$

   where $\alpha \in Names$ and $x \in Var$.

   An action corresponding to the operator (7.3) is executed by

   - an input to a process an object of the form (α, v), where
     - α is the name referred in (7.3), and
     - v is a message

     and
   - a record of the message v in the variable x

   i.e. after an execution of this action a value of the variable x becomes equal to v.

2. **Output operators**, which have the form

   $$\alpha\,!\,e \tag{7.4}$$

   where $\alpha \in Names$ and $e \in \mathcal{E}$.

   An action corresponding to the operator (7.4) is executed by an output from a process an object of the form (α, v), where

- α is the name referred in (7.4), and
- v is a value of the expression e on a current evaluation of variables of the process.

3. **Assignments** (first type of internal operators), which have the form

$$x \ := \ e \tag{7.5}$$

where

- $x \in Var$, and
- $e \in \mathcal{E}$, where $t(e) = t(x)$

An action corresponding to the operator (7.5) is executed by an updating of a value associated with the variable x: after an execution of this operator this value becomes equal to a value of the expression e on a current evaluation of variables of the process.

4. **Conditional operators** (second type of internal operators), which have the form

$$\langle b \rangle$$

where $b \in \mathcal{B}$.

An action corresponding to the operator $\langle b \rangle$ is executed by a calculation of a value of the formula b on a current evaluation of variables of the process, and

- if this value is 0, then an execution of the whole action is impossible, and
- if this value is 1, then the execution is completed.

The set of all operators is denoted by the symbol $\mathcal{O}$.

7.3.4 Definition of a process

A **process** is a 5-tuple P of the form

$$P = (X_P, I_P, S_P, s^0_P, R_P) \tag{7.6}$$

whose components have the following meanings:

1. $X_P \subseteq Var$ is a set of **variables** of the process P
2. $I_P \in \mathcal{B}$ is a formula, called an **initial condition** of the process P
3. S_P is a set of **states** of the process P
4. $s_P^0 \in S_P$ is an **initial state**
5. R_P is a subset of the form

$$R_P \subseteq S_P \times \mathcal{O} \times S_P$$

Elements of R_P are called **transitions**.

If a transition from R_P has the form (s_1, op, s_2), then we denote it as

$$s_1 \xrightarrow{op} s_2$$

and say that

- the state s_1 is a **start** of this transition,
- the state s_2 is an **end** of this transition,
- the operator op is a **label** of this transition.

Also, we assume that for each process P the set X_P contains a special variable at_P, which takes values in S_P.

7.3.5 An execution of a process

Let P be a process of the form (7.6).

An **execution** of a process P is a bypass of the set S_P of its states

- starting from the initial state s_P^0,
- through transitions from R_P, and
- with an execution of operators which are labels of visited transitions.

More detail: at each step $i \geq 0$ of an execution

- the process P is in some state s_i
  $(s_0 = s_P^0)$

- there is defined an evaluation $\xi_i \in Eval(X_P)$
  ($\xi_0(I_P)$ must be equal to 1)

- if there is a transition from R_P starting at s_i, then the process

  - selects a transition starting at s_i, which is labelled by such an operator op_i that can be executed at the current step (i),
    (if there is no such transitions, then the process P suspends until such transition will appear)
  - executes the operator op_i, and then
  - moves to the state s_{i+1} which is the end of the selected transition

- if there is no a transition in R_P starting in s_i, then the process completes its work.

For each $i \geq 0$ an evaluation ξ_{i+1} is determined

- by the evaluation ξ_i, and

- by the operator op_i, which is executed at i-th step of an execution of the process P.

A relationship between ξ_i, ξ_{i+1}, and op_i has the following form:

1. if $op_i = \alpha\,?\,x$, and in the execution of this operator it was inputted a message v, then

$$\begin{array}{ll} \xi_{i+1}(x) = v & \\ \forall y \in X_P \setminus \{x, at_P\} & \xi_{i+1}(y) = \xi_i(y) \end{array}$$

2. if $op_i = \alpha\,!\,e$, then in the execution of this operator it is outputted the message

$$\xi_i(e)$$

   and values of variables from $X_P \setminus \{at_P\}$ are not changed:

$$\forall x \in X_P \setminus \{at_P\} \qquad \xi_{i+1}(x) = \xi_i(x)$$

3. if $op_i =$ $(x := e)$, then

$$\begin{array}{ll} \xi_{i+1}(x) = \xi_i(e) & \\ \forall x \in X_P \setminus \{x, at_P\} & \xi_{i+1}(x) = \xi_i(x) \end{array}$$

4. if $op_i = \langle b \rangle$ and $\xi_i(b) = 1$, then

$$\forall x \in X_P \setminus \{at_P\} \qquad \xi_{i+1}(x) = \xi_i(x)$$

We assume that for each $i \geq 0$ a value of the variable at_P with respect to an evaluation ξ_i is equal to a state $s \in S_P$, in which the process P is located at step i, i.e.

- $\xi_0(at_P) = s_P^0$
- $\xi_1(at_P) = s_1$, where s_1 is an end of first transition
- $\xi_2(at_P) = s_2$, where s_2 is an end of second transition
- etc.

7.4 Representation of processes by flowcharts

In order to increase a visibility, a process can be represented as a flowchart.

A language of flowcharts is originated in programming, where use of this language can greatly facilitate a description and understanding of algorithms and programs.

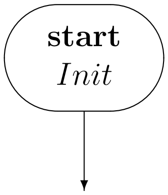
7.4.1 The notion of a flowchart

A **flowchart** is a directed graph, each node n of which

- is associated with an **operator** $op(n)$, and
- is depicted as one of the following geometric figures: a rectangle, an oval, or a circle, inside of which can be contained a label indicating $op(n)$.

An operator $op(n)$ can have one of the following forms.

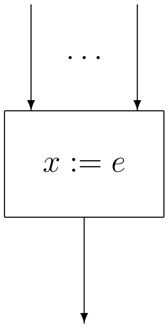
initial operator:



(7.7)

where $Init \in \mathbf{B}$ is a formula, called an **initial condition**.

assignment operator:

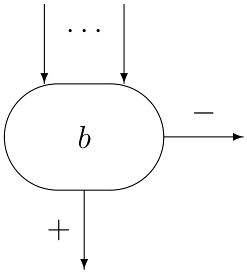


(7.8)

where

- $x \in Var$,
- $e \in \mathcal{E}$, where $t(e) = t(x)$

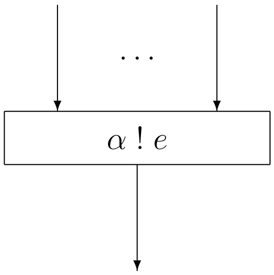
conditional operator:



(7.9)

where $b \in \mathcal{B}$.

sending operator:

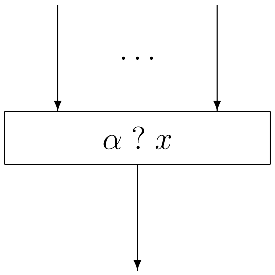


(7.10)

where

- $\alpha \in Names$ is a name (for example, it can be a destination of a message which will be sent), and
- $e \in \mathcal{E}$ is an expression whose value is a message which will be sent.

receiving operator:

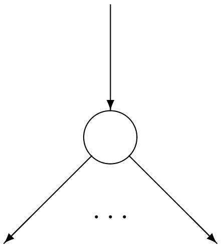


(7.11)

where

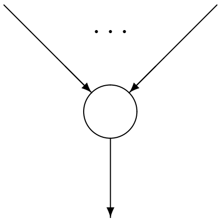
- $\alpha \in Names$ is a name (for example, it can be an expected source of a message which will be received), and
- $x \in Var$ is a variable in which a received message will be recorded.

choice:



(7.12)

join:

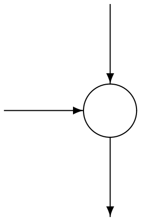


(7.13)

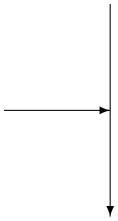
Sometimes

- a circle representing this operator, and
- ends of some edges leading to this circle

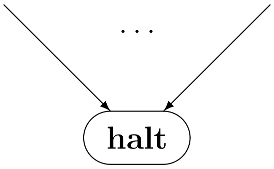
are not pictured. That is, for example, a fragment of a flowchart of the form



can be pictured as follows:



halt:



(7.14)

Flowcharts must meet the following conditions:

- a node of the type (7.7) can be only one
  (this node is called a **start node**)
- there is only one edge outgoing from nodes of the types (7.7), (7.8), (7.10), (7.11), (7.13)
- there are one or two edges outgoing from nodes of the type (7.9), and

  - if there is only one edge outgoing from a node of the type (7.9), then this edge has the label "+", and
  - if there are two edges outgoing from a node of the type (7.9), then
    - one of them has the label "+", and
    - another has the label "−".

- there is only one edge leading to a node of the type (7.12)
- there is no edges outgoing from a node of the type (7.14)

7.4.2 An execution of a flowchart

An **execution** of a flowchart is a sequence of transitions

- from one node to another along edges,
- starting from a start node n_0, and
- with an execution of operators which correspond to visited nodes.

More detail: each step $i \geq 0$ of an execution of a flowchart is associated with some node n_i called a **current node**, and

- if n_i is not of the type (7.14), then after an execution of an operator corresponded to the node n_i it is performed a transition along an edge outgoing from n_i to a node which will be a current node at the next step of an execution
- if n_i is of the type (7.14), then an execution of the flowchart is completed.

Let X be a set of all variables occurred in the flowchart.

At each step i of an execution ($i = 0, 1, \ldots$) each variable $x \in X$ is associated with a value $\xi_i(x)$.

The family $\{\xi_i(x) \mid x \in X\}$

- is denoted by ξ_i, and
- is called an **evaluation** of variables of the flowchart at i–th step of its execution.

The evaluation ξ_0 must meet the initial condition *Init*, i.e. the following statement must be true:

$$\xi_0(Init) = 1$$

An operator $op(n_i)$ associated with a current node n_i is executed as follows.

- If $op(n_i)$ has the type (7.8), then the value $\xi_i(e)$ is recorded in x i.e.

$$\begin{array}{l} \xi_{i+1}(x) \overset{\text{def}}{=} \xi_i(e) \\ \forall\, y \in X \setminus \{x\} \quad \xi_{i+1}(y) \overset{\text{def}}{=} \xi_i(y) \end{array}$$

- If $op(n_i)$ has the type (7.9) then
  - if $\xi_i(b) = 1$, then a transition along an edge outgoing from n_i with a label "+" is performed
  - if $\xi_i(b) = 0$, and there is an edge outgoing from n_i with a label "$-$", then a transition along this edge is performed
  - if $\xi_i(b) = 0$, and there is no an edge outgoing from n_i with a label "$-$", then an execution of $op(n_i)$ is impossbile.

- If $op(n_i)$ has the type (7.10) then an execution of this operator consists of a sending the object

$$(\alpha\,,\,\xi_i(e)) \tag{7.15}$$

  if it is possible.

  If a sending the object (7.15) is impossible, then an execution of $op(n_i)$ is impossbile.

- If $op(n_i)$ has the type (7.11) then an execution of this operator consists of
  - a receiving the object

$$(\alpha\,,\,v) \tag{7.16}$$

    (if it is possible), and
  - a recording of v in the variable x, i.e.

$$\begin{array}{l} \xi_{i+1}(x) \overset{\text{def}}{=} v \\ \forall y \in X \setminus \{y\} \quad \xi_{i+1}(y) \overset{\text{def}}{=} \xi_i(y) \end{array}$$

If a receiving the object (7.16) is impossible, then an execution of $op(n_i)$ is impossbile.

- If a current node n_i is associated with an operator of the type (7.12), then
  - among nodes which are ends of edges outgoing from n_i it is selected a node n labelled by such an operator, which can be executed at the current time, and
  - it is performed a transition to the node n.

  If there are several operators which can be executed at the current time, then a selection of the node n is performed non-deterministically.
- an operator of the type (7.14) completes an execution of the flowchart.

7.4.3 Construction of a process defined by a flowchart

An algorithm of a construction of a process defined by a flowchart has the following form.

1. At every edge of the flowchart it is selected a point.
2. For
   - each node n of the flowchart, which has no the type (7.12) or (7.13), and
   - each pair F_1, F_2 of edges of the flowchart such that F_1 is incoming in n, and F_2 is outgoing from n

   the following actions are performed:

   (a) it is drawn an arrow f from a point on F_1 to a point on F_2

   (b) it is drawn a label $label(f)$ on the arrow f, defined as follows:

      i. if $op(n)$ has the type (7.8), then

$$label(f) \overset{\text{def}}{=} (x := e)$$

ii. if $op(n)$ has the type (7.9), and an edge outgoing from n, has a label "+", then

$$label(f) \overset{\text{def}}{=} \langle b \rangle$$

iii. if $op(n)$ has the type (7.9), and an edge outgoing from n, has a label "−", then

$$label(f) \overset{\text{def}}{=} \langle \neg b \rangle$$

iv. if $op(n)$ has the type (7.10) or (7.11), then $label(f) = op(n)$.

3. For each node n of the type (7.12) and each edge F outgoing from n, the following actions are performed. Let

   - p be a point on an edge incoming to n,
   - p' be a point on F,
   - n' be an end of F, and
   - p'' be a poing on an edge outgoing from n'.

   Then

   - an arrow from p' to p'' is replaced on an arrow from p to p'' with the same label, and
   - the point p' is removed.

4. For each node n of the type (7.13) and each edge F incoming from n, the following actions are performed. Let

   - p be a point on an edge outgoing from n,
   - p' be a point on F,
   - n' be a start of F, and
   - p'' be a poing on an edge incoming to n'.

   Then

   - an arrow from p'' to p' is replaced on an arrow from p'' to p with the same label, and
   - the point p' is removed.

5. States of a constructed process are remaining points.

6. An initial state s_P^0 is defined as follows.

   - If a point which was selected on an edge outgoing from a start node of the flowchart was not removed, then s_P^0 is this point.
   - If this point was removed, then an end of an edge outgoing from a start note of the flowchart is a node n of the type (7.13). In this case, s_P^0 is a point on an edge outgoing from n.

7. Transitions of the process correspond to the pictured arrows: for each such arrow f the process contains a transition

$$s_1 \xrightarrow{label(f)} s_2$$

   where s_1 and s_2 are a start and an end of the arrow f respectively.

8. A set of variables of the process consists of

   - all variables occurred in any operator of the flowchart, and
   - the variable at_P.

9. An initial condition of the process coincides with the initial condition $Init$ of the flowchart.

7.5 An example of a process with a message passing

In this section we consider a process "buffer" as an example of a process with a message passing:

- at first, we define this process as a flowchart, and
- then we transform this flowchart to a standard graph representation of a process.

7.5.1 The concept of a buffer

A **buffer** is a system which has the following properties.

- It is possible to input messages to a buffer.

  A message which is entered to the buffer is stored in the buffer.

  Messages which are stored in a buffer can be extracted from the buffer.

  We assume that a buffer can store not more than a given number of messages. If n is a such number, then we shall denote the buffer as Buffer_n.

- At each time a list of messages

  $$c_1, \ldots, c_k \qquad (0 \leq k \leq n) \tag{7.17}$$

  stored in Buffer_n is called a **content of the buffer**.

  The number k in (7.17) is called a **size** of this content.

  The case $k = 0$ corresponds to the situation when a content of the buffer is empty.

- If at a current time a content of Buffer_n has the form (7.17), and $k < n$, then

  - the buffer can accept any message, and
  - after an execution of the action of an input of a message c a content of the buffer becomes

    $$c_1, \ldots, c_k, c$$

- If at the current time a content of Buffer_n has the form (7.17), and $k > 0$, then

  - it is possible to extract the message c_1 from the buffer, and
  - after an execution of this operation a content of the buffer becomes

    $$c_2, \ldots, c_k$$

Thus, at each time a content of a buffer is a queue of messages, and

- each action of an input of a message to a buffer adds this message to an end of the queue, and

- each action of an output of a message from the buffer
  - extracts a first message of this queue, and
  - removes this message from the queue.

A queue with the above operations is called **a queue of the type FIFO** (First Input - First Output).

7.5.2 Representation of a buffer by a flowchart

In this section we present a formal description of the concept of a buffer by a flowchart.

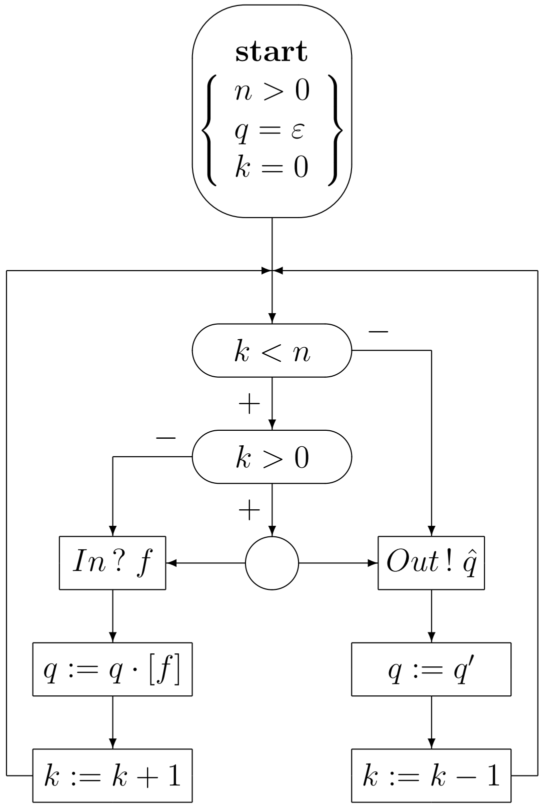
In this flowchart

- the operation of an input of a message to a buffer is represented by an action with the name In, and
- the operation of an output of a message from the buffer is represented by an action with the name Out.

The flowchart has the following variables:

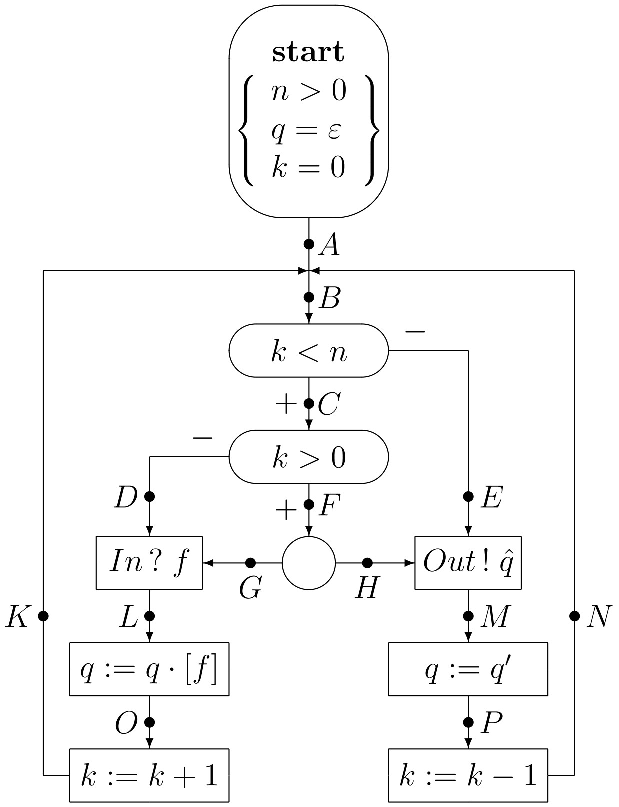
- the variable n of the type `int`, its value does not change, it is equal to the maximal size of a content of the buffer
- the variable k of the type `int`, its value is equal to a size of a content of the buffer at a current time
- the variable f of the type `mes`, this variable will store messages that will come to the buffer
- the variable q of the type `list`, this variable will store a content of the buffer.

A flowchart representing a behavior of a buffer has the following form: (notations used in this flowchart were defined in section 7.2.3)



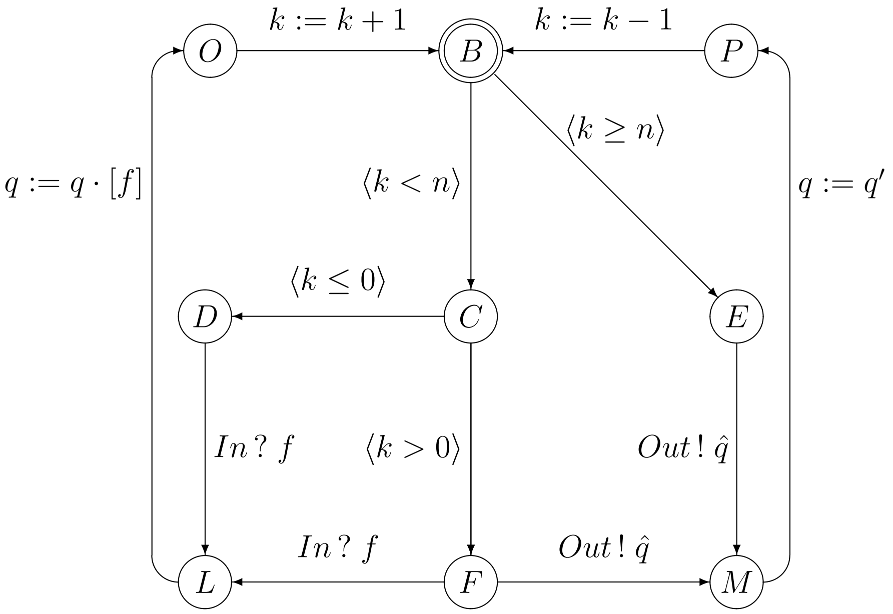
7.5.3 Representation of a buffer as a process

To construct a process $Buffer_n$, which corresponds to the above flowchart, we select points at its edges:



In a construction of a process defined by this flowchart, the points A, G, H, K and N will be removed.

A standard graph representation of the process $Buffer_n$ is the following.



7.6 Operations on processes with a message passing

Operations on processes with a message passing are similar to operations which are considered in chapter 3.

7.6.1 Prefix action

Let P be a process, and *op* be an operator.

The process $op.\,P$ is obtained from P by an adding

- a new state s, which will be an initial state in $op.\,P$,
- a new transition $s \xrightarrow{op} s^0_P$, and
- all variables from *op*.

7.6.2 Alternative composition

Let P_1, P_2 be processes such that $S_{P_1} \cap S_{P_2} = \emptyset$.

Define a process $P_1 + P_2$, which is called an **alternative composition** of P_1 and P_2, as follows.

- sets of its states, transitions, and an initial state are determined by the same way as corresponding components of an alternative composition in chapter 3 (section 3.3)
- $X_{P_1+P_2} \stackrel{\text{def}}{=} X_{P_1} \cup X_{P_2}$
- $I_{P_1+P_2} \stackrel{\text{def}}{=} I_{P_1} \wedge I_{P_2}$

If $S_{P_1} \cap S_{P_2} \neq \emptyset$, then for a construction of the process $P_1 + P_2$ it is necessary

- to replace in S_{P_2} those states that are also in P_1 on new states, and
- modify accordingly other components of P_2.

7.6.3 Parallel composition

Let P_1 and P_2 be processes such that $X_{P_1} \cap X_{P_2} = \emptyset$.

Define a process $P_1 \,|\, P_2$, which is called a **parallel composition** of P_1 and P_2, as follows:

- a set of its states and its initial state are defined by the same way as are defined the corresponding components of the process $P_1 \,|\, P_2$ in chapter 3
- $X_{P_1+P_2} \stackrel{\text{def}}{=} X_{P_1} \cup X_{P_2}$
- $I_{P_1+P_2} \stackrel{\text{def}}{=} I_{P_1} \wedge I_{P_2}$
- the set of transitions of the process $P_1 \,|\, P_2$ is defined as follows:
  - for
    - each transition $s_1 \xrightarrow{op} s_1'$ of the process P_1, and
    - each state s of the process P_2

the process $P_1 \,|\, P_2$ contains the transition

$$(s_1, s) \xrightarrow{op} (s'_1, s)$$

- for
  - each transition $s_2 \xrightarrow{op} s'_2$ of the process P_2, and
  - each state s of the process P_1

  the process $P_1 \,|\, P_2$ contains the transition

$$(s, s_2) \xrightarrow{op} (s, s'_2)$$

- each pair of transitions of the form

$$\begin{array}{lll} s_1 \xrightarrow{op_1} s'_1 & \in R_{P_1} \\ s_2 \xrightarrow{op_2} s'_2 & \in R_{P_2} \end{array}$$

  where
  - one of the operators op_1, op_2 has the form $\alpha\,?\,x$,
  - and another has the form $\alpha\,!\,e$, where $t(x) = t(e)$ (names in both the operators are equal)

  the process $P_1 \,|\, P_2$ contains the transition

$$(s_1, s_2) \xrightarrow{x := e} (s'_1, s'_2)$$

If $X_{P_1} \cap X_{P_2} \neq \emptyset$, then before a construction of the process $P_1 \,|\, P_2$ it is necessary to replace variables which occur in both processes on new variables.

7.6.4 Restriction and renaming

Definition of there operations is the same as definition of corresponding operations in chapter 3.

7.7 Equivalence of processes

7.7.1 The concept of a concretization of a process

Let P be a process.

We shall denote by $Conc(P)$ a process in the original sense of this concept (see section 2.4), which is called a **concretization** of the process P, and has the following components.

1. States of $Conc(P)$ are
   - all evaluations from $Eval(X_P)$, and
   - an additional state s^0, which is an initial state of $Conc(P)$
2. For
   - each transition $s_1 \xrightarrow{op} s_2$ of the process P, and
   - each evaluation $\xi \in Eval(X_P)$, such that

   $$\xi(at_P) = s_1$$

   $Conc(P)$ has a transition

   $$\xi \xrightarrow{a} \xi'$$

   if $\xi'(at_P) = s_2$, and one of the following conditions is satisfied:

   - – $op = \alpha\,?\,x, \quad a = \alpha\,?\,v$, where $v \in D_{t(x)}$
     – $\xi'(x) = v, \quad \forall y \in X_P \setminus \{x, at_P\} \qquad \xi'(y) = \xi(y)$
   - – $op = \alpha\,!\,e, \quad a = \alpha\,!\,\xi(e)$
     – $\forall x \in X_P \setminus \{at_P\} \qquad \xi'(x) = \xi(x)$
   - – $op = \ (x := e), \quad a = \tau$
     – $\xi'(x) = \xi(e), \quad \forall y \in X_P \setminus \{x, at_P\} \qquad \xi'(y) = \xi(y)$
   - – $op = \langle b \rangle, \quad \xi(b) = 1, \quad a = \tau$
     – $\forall x \in X_P \setminus \{at_P\} \qquad \xi'(x) = \xi(x)$
3. For
   - each evaluation $\xi \in Eval(X_P)$, such that

   $$\xi(I_P) = 1$$

   - and each transition of $Conc(P)$ of the form $\xi \xrightarrow{a} \xi'$

   $Conc(P)$ has the transition $s^0 \xrightarrow{a} \xi'$

From the definitions of

- the concept of an execution of a process with a message passing (see section 7.3.5), and
- the concept of an execution of a process in the original sense (see section 2.4)

it follows that there is a one-to-one correspondence between

- the set of all variants of an execution of the process P, and
- the set of all variants of an execution of $Conc(P)$.

A reader is invited to investigate the commutativity property of the mapping $Conc$ with respect to the operations on processes i.e. to check statements of the form

$$Conc(P_1 \mid P_2) = Conc(P_1) \mid Conc(P_2)$$

etc.

7.7.2 Definition of equivalences of processes

We define that every pair (P_1, P_2) of processes with a message passing is in the same equivalence ($\sim$, $\approx$, $\overset{+}{\approx}$, $\ldots$), in which is a pair of concretizations of these processes, i.e.

$$P_1 \sim P_2 \quad \Leftrightarrow \quad Conc(P_1) \sim Conc(P_2), \quad \text{etc.}$$

A reader is invited to

- explore a relationship of the operations on processes with various equivalences ($\approx, \overset{+}{\approx}, \ldots$), i.e. to establish properties, which are similar to the properties presented in sections 3.7, 4.5, 4.8.4, 4.9.5
- formulate and prove necessary and sufficient conditions of equivalence ($\approx, \overset{+}{\approx}, \ldots$) of processes that do not use the concept of a concretization of a process.

7.8 Processes with composite operators

7.8.1 A motivation of the concept of a process with composite operators

A complexity of the problem of an analysis of a process essentially depends on a size of its description (in particular, on a number of its states). Therefore, for a construction of efficient algorithms of an analysis of processes it is required a search of methods to decrease a complexity of a description of analyzed processes. In this section we consider one of such methods.

In this section we generalize the concept of a process to the concept of a process with composite operators. A composite operator is a sequential composition of several operators. Due to the fact that we combine a sequence of operators in a single composite operator, we are able to exclude from a description of a process those states which are located on the intermediate steps of this sequence of operators.

Also in this section we define the concept of a reduction of processes with composite operators in such a way that a reduced process

- has a less complicated description than an original process, and
- is equivalent (in some sense) to an original process.

With use of the above concepts, the problem of an analysis of a process can be solved as follows.

1. First, we transform an original process P to a process P' with composite operators, which is similar to P.
2. Then we reduce P', getting a process P'', whose complexity can be significantly less than a complexity of the original process P.
3. After this, we
   - perform an analysis of P'', and
   - use results of this analysis for drawing a conclusion about properties of the original process P.

7.8.2 A concept of a composite operator

A **composite operator (CO)** is a finite sequence Op of operators

$$Op = (op_1, \dots, op_n) \qquad (n \geq 1) \tag{7.18}$$

which has the following properties.

1. op_1 is a conditional operator.
2. The sequence $(op_2, \dots, op_n)$
   - does not contain conditional operators, and
   - contains no more than one input or output operator.

If Op is a CO of the form (7.18), then we shall denote by

$$cond\,(Op)$$

a formula b such that $op_1 = \langle b \rangle$.

Let Op be a CO.

- Op is said to be an **input CO** (or an **output CO**), if if among operators belonging to Op, there is an input (or an output) operator.
- Op is said to be an **internal CO**, if all operators belonging to Op are internal.
- If Op is an **input CO** (or an **output CO**), then the notation

  $$name\,(Op)$$

  denotes a name occurred in Op.
- If ξ is an evaluation of variables occurred in $cond\,(Op)$, then we say that Op **is open on** ξ, if

  $$\xi(cond\,(Op)) = 1$$

7.8.3 A concept of a process with COs

A concept of a **process with COs** differs from the concept of a process in section 7.3.4 only in the following: labels of transitions of a process with COs are COs.

7.8.4 An execution of a process with COs

An execution of a process with COs

- is defined in much the same as it is defined an execution of a process in section 7.3.5, and
- is also a bypass of a set of its states,
  - starting from an initial state, and
  - with an execution of COs which are labels of visited transitions.

Let $P = (X_P, I_P, S_P, s^0_P, R_P)$ be a process with COs.
At each step $i \geq 0$ of an execution of P

- the process P is in some state s_i ($s_0 = s^0_P$)
- there is defined an evaluation ξ_i of variables from X_P
  ($\xi_0(I_P) = 1, \quad \xi_i(at_P) = s_i$)
- if there is a transition from R_P, starting at s_i, then the process
  - selects a transition starting at s_i, which is labelled by a CO Op_i with the following properties:
    - Op_i is open on ξ_i
    - if among operators occurred in Op_i there is an operator of the form
      $$\alpha\,?\,x \quad \text{or} \quad \alpha\,!\,e$$
      then the process P can in the current time execute an action of the form
      $$\alpha\,?\,v \quad \text{or} \quad \alpha\,!\,v$$
      respectively

    (if there is no such transitions, then the process P suspends until such transition will appear)
  - executes sequentially all operators occurred in Op_i, with a corresponding modification of the current evaluation after an execution of each operator occurred in Op_i, and thereafter
  - turns to the state s_{i+1}, which is the end of the selected transition
- if there is no a transition in R_P starting at s_i, then the process completes its work.

7.8.5 Operations on processes with COs

Definitions of operations on processes with COs almost coincide with corresponding definitions in section 7.6, so we only point out the differences in these definitions.

- In definitions of all operations on processes with COs instead of operators COs are mentioned.
- Definitions of the operation "$|$" differ only in the item, which is related to a description of "diagonal" transitions.

  For processes with COs this item has the following form: for each pair of transitions of the form

  $$\begin{array}{lcl} s_1 \xrightarrow{Op_1} s_1' & \in R_{P_1} \\ s_2 \xrightarrow{Op_2} s_2' & \in R_{P_2} \end{array}$$

  where one of the COs Op_1, Op_2 has the form

  $$(op_1,\ \ldots,\ op_i,\ \alpha\,?\,x,\ op_{i+1},\ \ldots,\ op_n)$$

  and another of the COs has the form

  $$(op'_1,\ \ldots,\ op'_j,\ \alpha\,!\,e,\ op'_{j+1},\ \ldots,\ op'_m)$$

  where

  - $t(x) = t(e)$,
  - the subsequences

    $$(op_{i+1},\ \ldots,\ op_n) \text{ and } (op'_{j+1},\ \ldots,\ op'_m)$$

    may be empty

  the process $P_1 \,|\, P_2$ has the transition

  $$(s_1, s_2) \xrightarrow{Op} (s'_1, s'_2)$$

  where Op has the form

  $$\left(\begin{array}{l} \langle cond\,(Op_1) \wedge cond\,(Op_2)\rangle, \\ op_2,\ \ldots,\ op_i, \\ op'_2,\ \ldots,\ op'_j, \\ (x := e), \\ op_{i+1},\ \ldots,\ op_n, \\ op'_{j+1},\ \ldots,\ op'_m \end{array}\right)$$

7.8.6 Transformation of processes with a message passing to processes with COs

Each process with a message passing can be transformed to a process with COs by a replacement of labels of its transitions: for each transition

$$s_1 \xrightarrow{op} s_2$$

its label *op* is replaced by the CO *Op*, defined as follows.

- If *op* is a conditional operator, then

$$Op \overset{\text{def}}{=} (op)$$

- If *op* is
  - an assignment operator, or
  - an input or output operator

  then $Op \overset{\text{def}}{=} (\langle \top \rangle, op)$
  (remind that $\top$ is a true formula)

For each process with a message passing P we denote the corresponding process with COs by the same symbol P.

7.8.7 Sequential composition of COs

In this section, we introduce the concept of a **sequential composition** of COs: for some pairs (Op_1, Op_2) of COs we define a CO, which is denoted as

$$Op_1 \cdot Op_2 \tag{7.19}$$

and is called a **sequential composition** of the COs Op_1 and Op_2.

A necessary condition of a possibility to define a sequential composition (7.19) is the condition that at least one of the COs Op_1, Op_2 is internal.

Below we shall use the following notations.

1. For

   - each CO $Op = (op_1, \ldots, op_n)$, and

- each assignment operator *op*

the notation $Op \cdot op$ denotes the CO

$$(op_1, \ldots, op_n, op) \tag{7.20}$$

2. For

   - each internal CO $Op = (op_1, \ldots, op_n)$, and
   - each input or output operator *op*

   the notation $Op \cdot op$ denotes the CO (7.20)

3. For

   - each CO $Op = (op_1, \ldots, op_n)$, and
   - each conditional operator $op = \langle b \rangle$

   the notation $Op \cdot op$ denotes an object that

   - either is a CO
   - or is not defined.

   This object is defined recursively as follows.
   If $n = 1$, then

   $$Op \cdot op \stackrel{\text{def}}{=} (\langle cond\,(Op) \wedge b \rangle)$$

   If $n > 1$, then

   - if op_n is an assignment operator of the form $(x := e)$, then

     $$Op \cdot op \stackrel{\text{def}}{=} \underbrace{((op_1, \ldots, op_{n-1}) \cdot op_n(op))}_{(*)} \cdot op_n$$

     where

     – $op_n(op)$ is a conditional operator, which is obtained from *op* by a replacement of all occurrences of the variable x on the expression e
     – if the object $(*)$ is undefined, then $Op \cdot op$ also is undefined

- if op_n is an output operator, then $Op \cdot op$ is the CO
$$((op_1, \ldots, op_{n-1}) \cdot op) \cdot op_n \tag{7.21}$$
- if op_n is an input operator, and has the form $\alpha\,?\,x$, then $Op \cdot op$
  - is undefined, if op depends on x, and
  - is equal to CO (7.21), otherwise.

Now we can formulate a definition of a sequential composition of COs. Let Op_1, Op_2 be COs, and Op_2 has the form

$$Op_2 = (op_1, \ldots, op_n)$$

We shall say that **there is defined a sequential composition** of Op_1 and Op_2, if the following conditions are met:

- at least one of the COs Op_1, Op_2 is internal
- there is no undefined objects in the parentheses in the expression
$$(\ldots((Op_1 \cdot op_1) \cdot op_2) \cdot \ldots) \cdot op_n \tag{7.22}$$

If these conditions are met, then a **sequential composition** Op_1 and Op_2 is a value of expression (7.22). This CO is denoted by

$$Op_1 \cdot Op_2$$

7.8.8 Reduction of processes with COs

Let P be a process with COs.

A **reduction** of P is a sequence

$$P = P_0 \longrightarrow P_1 \longrightarrow \ldots \longrightarrow P_n \tag{7.23}$$

of transformations of this process, each of which is performed according to any of the reduction rules described below. Each of these transformations (except the first) is made on the result of the previous transformation.

A **result** of the reduction (7.23) is a result of the last transformation (i.e. the process P_n).

Reduction rules have the following form.

Rule 1 (sequential composition).

Let s be a state of a process with CO, which is not an initial state, and

- a set of all transitions of this process with the end s has the form

$$s_1 \xrightarrow{Op_1} s, \quad \ldots, s_n \xrightarrow{Op_n} s$$

- a set of all transitions of this process with the start s has the form

$$s \xrightarrow{Op'_1} s'_1, \quad \ldots, s \xrightarrow{Op'_m} s'_m$$

- $s \not\in \{s_1, \ldots, s_n, s'_1, \ldots, s'_m\}$
- for each $i = 1, \ldots, n$ and each $j = 1, \ldots, m$ there is defined the sequential composition

$$Op_i \cdot Op_j$$

Then this process can be transformed to a process

- states of which are states of the original process, with the exception of s
- transitions of which are
  - transitions of the original process, a start or an end of which is not s, and
  - transitions of the form

$$s_i \xrightarrow{Op_i \cdot Op'_j} s'_j$$

    for each $i = 1, \ldots, n$ and each $j = 1, \ldots, m$
- - an initial state of which, an also
  - a set of variables, and
  - an initial condition

  coincide with the corresponding components of the original process.

Rule 2 (gluing).

Let P be a process with CO, which has two transitions with a common start and a common end:

$$s_1 \xrightarrow{Op} s_2, \qquad s_1 \xrightarrow{Op'} s_2 \tag{7.24}$$

and labels of these transitions differ only in first components, i.e. Op and Op' have the form

$$\begin{array}{l} Op = (op_1, op_2, \ldots, op_n) \\ Op' = (op'_1, op_2, \ldots, op_n) \end{array}$$

Rule 2 is a replacement of the pair of transitions (7.24) on a transition

$$s_1 \xrightarrow{Op} s_2$$

where $Op = (\langle cond\,(Op) \vee cond\,(Op')\rangle, op_2, \ldots, op_n)$

Rule 3 (removal of inessential assignments).

Let

- P be a process with CO, and
- $op(P)$ be a set of all operators, occurred in COs of P.

A variable $x \in X_P$ is said to be **inessential**, if

- x does not occur in
  - conditional operators, and
  - output operators

  in $op(P)$,
- if x has an occurrence in a right size of any assignment operator from $op(P)$ of the form $(y := e)$, then the variable y is inessential.

Rule 3 is a removal from all COs of all assignment operators of the form $(x := e)$, where the variable x is inessential.

7.8.9 An example of a reduction

In this section we consider a reduction of the process $Buffer_n$, the graph representation of which is given in section 7.5.3.

Below we use the following agreements.

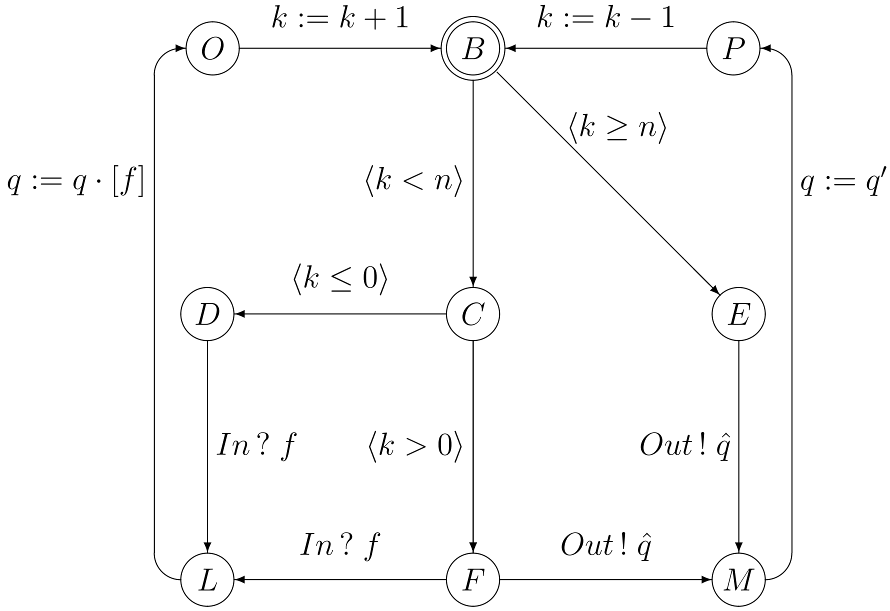
- If Op is a CO such that

$$cond\,(Op) = \top$$

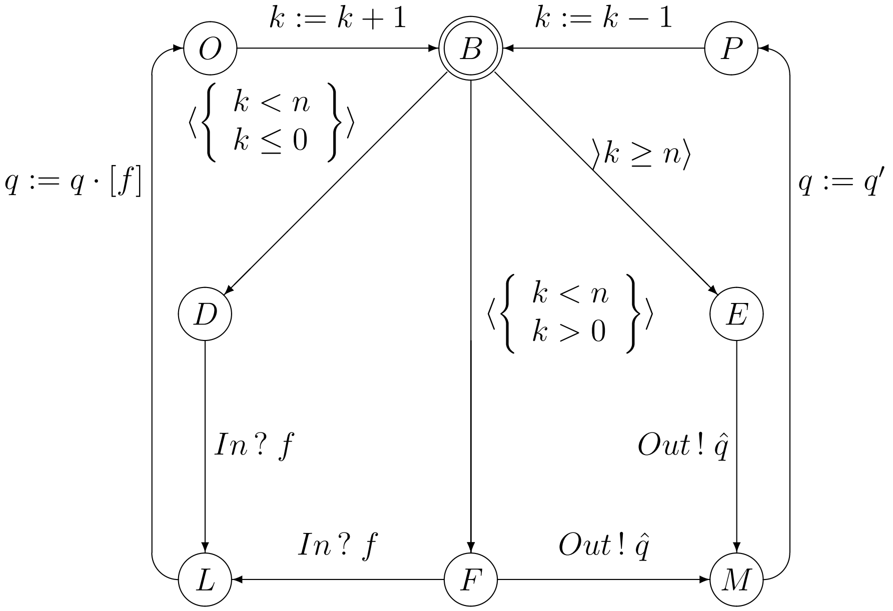
then the first operator in this CO will be omitted.

- Operators in COs can be placed vertically.
- Brackets, which embrace a sequence of operators consisting in a CO, can be omitted.

The original process $Buffer_n$ has the following form:

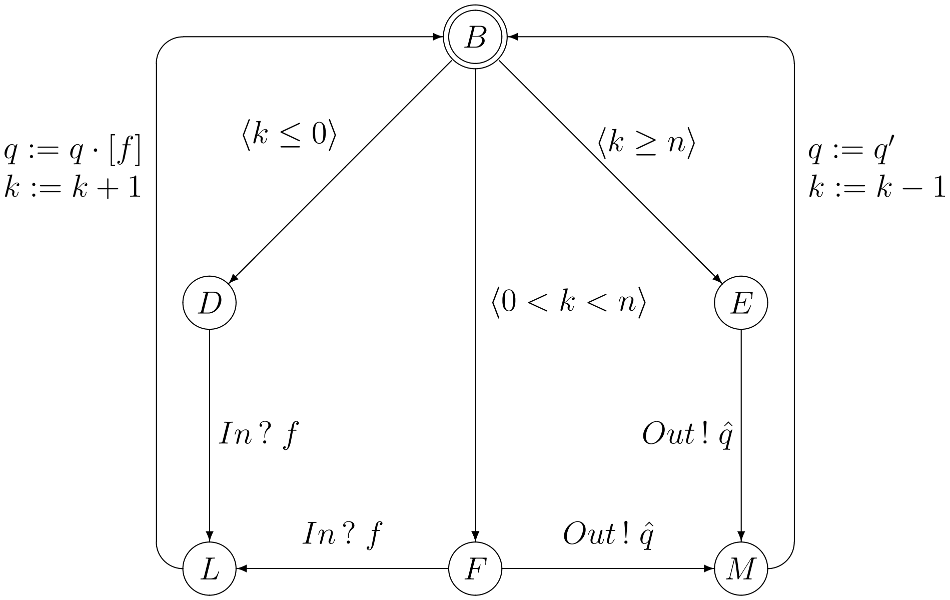


First reduction step is a removing of the state C (we apply rule 1 for $s = C$):

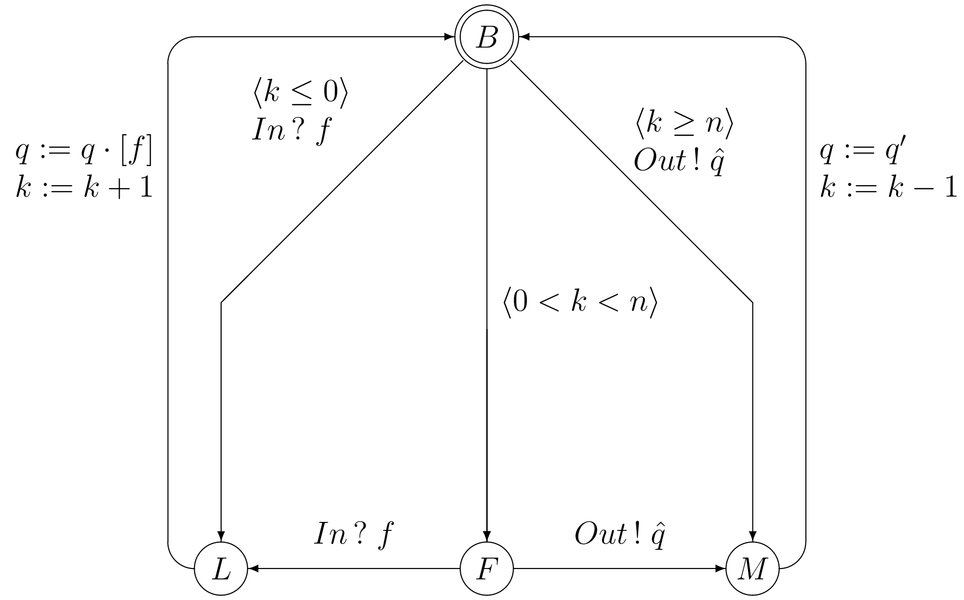

$k := k + 1$
$k := k - 1$
O
B
P
$\langle\left\{\begin{array}{l} k < n \\ k \leq 0 \end{array}\right\}\rangle$
$\rangle k \geq n\rangle$
$q := q \cdot [f]$
$q := q'$
D
E
$\langle\left\{\begin{array}{l} k < n \\ k > 0 \end{array}\right\}\rangle$
$In \,?\, f$
$Out \,!\, \hat{q}$
$In \,?\, f$
$Out \,!\, \hat{q}$
L
F
M

Since $n > 0$, then the formula $(k < n) \wedge (k \leq 0)$ in the label of the transition from B to D can be replaced by the equivalent formula $k \leq 0$.

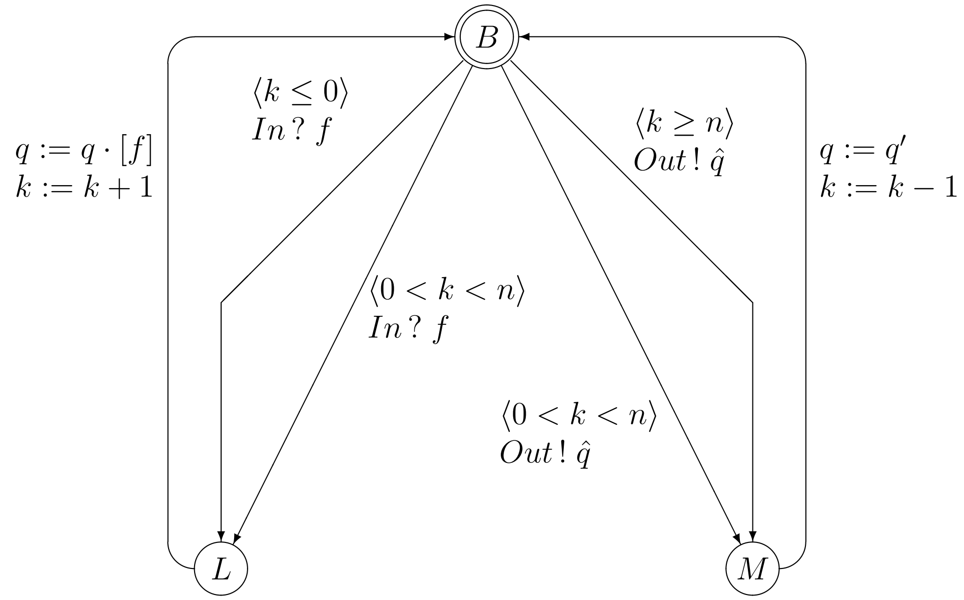
Second and third reduction steps are removing of states O and P:



Fourth and fifth reduction steps are removing of the states D and E:

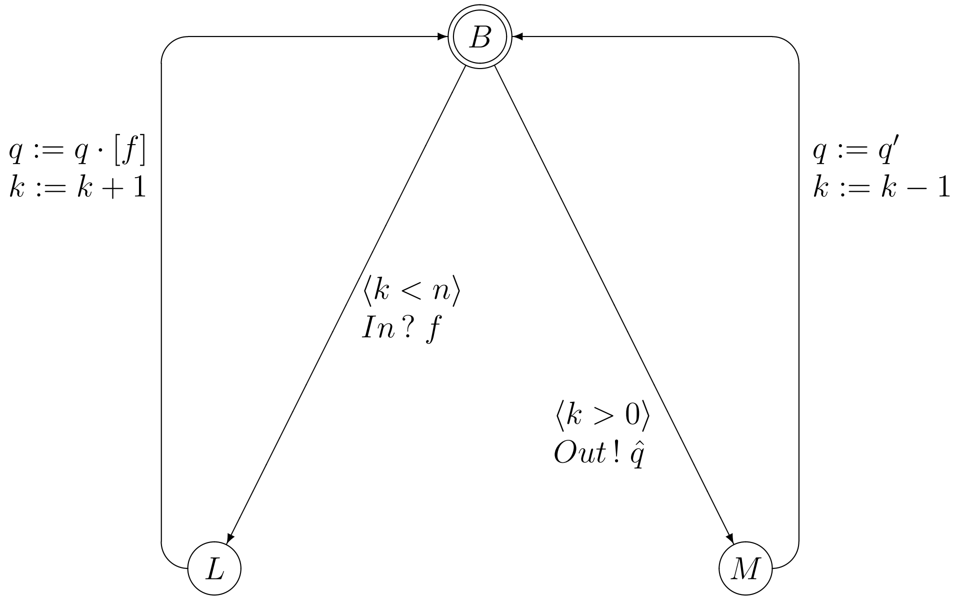


Sixth reduction step is removing of the state F:

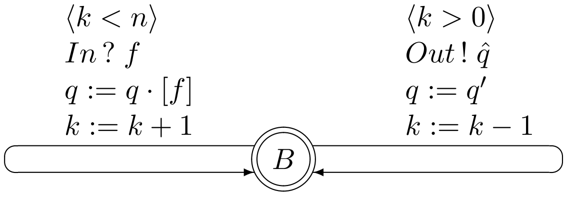


Seventh and eighth reduction steps consist of an application of rule 2 to the transitions from B to L and from B to M. In the resulting process, we replace

- the formula $(0 < k < n) \vee (k \leq 0)$ on the equivalent formula $k < n$, and
- the formula $(0 < k < n) \vee (k \geq n)$ on the equivalent formula $k > 0$.



Ninth and tenth reduction steps are removing of states L and M.



(7.25)

The last process is the result of the reduction of $Buffer_n$.

7.8.10 A concretization of processes with COs

A concept of a concretization of processes with COs is similar to the concept of a concretization of processes with a message passing (see section 7.7.1).

Let P be a process with COs. The notation $Conc(P)$ denotes a process in the original sense of this concept (see section 2.4), which is called a **concretization** of the process P, and has the following components.

1. States of $Conc(P)$ are
   - all evaluations from $Eval(X_P)$, and
   - an additional state s^0, which is an initial state of $Conc(P)$
2. For
   - each transition $s_1 \xrightarrow{Op} s_2$ of the process P, and
   - each evaluation $\xi \in Eval(X_P)$, such that
     - $\xi(at_P) = s_1$, and
     - Op is open on ξ

   $Conc(P)$ has the transition

   $$\xi \xrightarrow{a} \xi'$$

   if $\xi'(at_P) = s_2$, and one of the following cases hold:

   (a) Op is internal, $a = \tau$, and the following statement holds:

   $$\xi \xrightarrow{Op} \xi'$$

   which means the following: if Op has the form

   $$(op_1, \ldots, op_n)$$

   then there is a sequence $\xi_1, \ldots, \xi_n$ of evaluations from $Eval(X_P)$, such that
   - $\forall\, x \in X_P \setminus \{at_P\} \quad \xi(x) = \xi_1(x), \quad \xi'(x) = \xi_n(x)$, and
   - $\forall\, i = 2, \ldots, n$, if op_i has the form $(x := e)$, then

   $$\xi_i(x) = \xi_{i-1}(e), \quad \forall y \in X_P \setminus \{x, at_P\} \quad \xi_i(y) = \xi_{i-1}(y)$$

(b)
- $Op = Op_1 \cdot (\alpha\,?\,x) \cdot Op_2$,
- $a = \alpha\,?\,v$, where $v \in D_{t(x)}$, and
- there are evaluations ξ_1 and ξ_2 from$Eval(X_P)$, such that

$$\xi \xrightarrow{Op_1} \xi_1\,, \qquad \xi_2 \xrightarrow{Op_2} \xi'$$
$$\xi_2(x) = v, \quad \forall y \in X_P \setminus \{x, at_P\} \qquad \xi_2(y) = \xi_1(y)$$

(c)
- $Op = Op_1 \cdot (\alpha\,!\,e) \cdot Op_2$,
- there is an evaluation ξ_1 from $Eval(X_P)$, such that

$$\xi \xrightarrow{Op_1} \xi_1\,, \qquad \xi_1 \xrightarrow{Op_2} \xi'\,, \qquad a = \alpha\,!\,\xi_1(e)$$

3. For

- each evaluation $\xi \in Eval(X_P)$, such that

$$\xi(I_P) = 1$$

- and each transition of $Conc(P)$ of the form $\xi \xrightarrow{a} \xi'$

$Conc(P)$ has the transition $s^0 \xrightarrow{a} \xi'$.

A reader is invited to investigate a relationship between

- a concretization of an arbitrary process with a message passing P, and
- a concretization of a process with COs, which is derived by a reduction of the process P.

7.8.11 Equivalences on processes with COs

Let P_1 and P_2 be processes with COs.

We shall say that P_1 and P_2 are **observationally equivalent** and denote this fact by

$$P_1 \approx P_2$$

if the concretizations $Conc(P_1)$ and $Conc(P_2)$ are observationally equivalent in the original sense of this concept (see section 4.8).

Similarly, the equivalence $\overset{+}{\approx}$ is defined on processes with COs.

Using the concept of a reduction of processes with COs, it is possible to define another equivalence on the set of processes with COs. This equivalence

- is denoted by $\overset{r}{\approx}$, and
- is a minimal congruence on the set of processes with COs, with the following property: if P' is derived from P by any reduction rule, then $P \overset{r}{\approx} P'$

(i.e. $\overset{r}{\approx}$ is the intersection of all congruences on the set of processes with COs, which have the above property).

A reader is invited

- to investigate a relation between
  - operations on processes with COs, and
  - the equivalences $\approx$ and $\overset{+}{\approx}$

  i.e. to establish properties, which are similar to properties represented in sections 3.7, 4.5, 4.8.4, 4.9.5
- to formulate and justify necessary and sufficient conditions of observational equivalence of processes with COs, without use of the concept of a concretization
- explore a relationship between the equivalences $\approx$, $\overset{+}{\approx}$ and $\overset{r}{\approx}$
- find reduction rules such that

$$\overset{r}{\approx} \ \subseteq \overset{+}{\approx}$$

7.8.12 A method of a proof of observational equivalence of processes with COs

One of possible methods of a proof of observational equivalence of processes with COs is based on theorem 34 presented below.

To formulate this theorem, we introduce auxiliary concepts and notations.

1. Let P be a process with COs.

   A **composite transition (CT)** in P is a (possibly empty) sequence CT of transitions of the process P of the form

   $$CT = \quad s_0 \xrightarrow{Op_1} s_1 \xrightarrow{Op_2} \ldots \xrightarrow{Op_n} s_n \qquad (n \geq 0) \tag{7.26}$$

   such that

- among the COs $Op_1, \ldots, Op_n$ there is no more than one input or output CO
- there is defined the sequential composition

  $$(\ldots(Op_1 \cdot Op_2) \cdot \ldots) \cdot Op_n$$

  which will be denoted by the same symbol CT.

If sequence (7.26) is empty, then its sequential composition CT by definitions is the CO $(\langle\top\rangle)$.

The state s_0 is said to be a **start** of CT (7.26), and the state s_n is said to be an **end** of this CT.

The notation $s_0 \xrightarrow{CT} s_n$ is an abridged record of the statement that CT

- is a CT with the start s_0 and the end s_n, and also
- is a CO that corresponds to this CT.

2. Let φ and ψ be formulas.

   The notation $\varphi \leq \psi$ is an abridged record of the statement that the formula $\varphi \to \psi$ is true.

3. Let $Op = (op_1, \ldots, op_n)$ be an internal CO, and φ be a formula.

   The notation $Op(\varphi)$ denotes a formula defined recursively:

   $$Op(\varphi) \overset{\text{def}}{=} \begin{cases} cond\,(Op) \to \varphi, & \text{if } n = 1 \\ (op_1, \ldots, op_{n-1})\,(op_n(\varphi)), & \text{if } n > 1 \end{cases}$$

   where $op_n(\varphi)$ denotes the following formula: if $op_n = (x := e)$, then $op_n(\varphi)$ is obtained from φ by a replacement of each occurrence of the variable x on the expression e.

4. Let φ, ψ be formulas, and Op_1, Op_2 be COs.

We shall say that the following diagram holds

$$\begin{tikzcd}
A \arrow[r, dash, "\varphi"] \arrow[d, "Op_1"'] & B \arrow[d, "Op_2"] \\
C \arrow[r, dash, "\psi"'] & D
\end{tikzcd} \tag{7.27}$$

if one of the following conditions is met.

(a) Op_1 and Op_2 are internal COs, and the following inequality holds:

$$\varphi \leq (Op_1 \cdot Op_2)(\psi)$$

(b) Op_1 and Op_2 can be represented as sequential compositions

$$Op_1 = Op_3 \cdot (\alpha \,?\, x) \cdot Op_4$$
$$Op_2 = Op_5 \cdot (\alpha \,?\, y) \cdot Op_6$$

where Op_3, Op_4, Op_5, Op_6 are internal COs, and the following inequality holds

$$\varphi \leq (Op_1' \cdot Op_2')(\psi)$$

where

- $Op_1' = Op_3 \cdot (x := z) \cdot Op_4$
- $Op_2' = Op_5 \cdot (y := z) \cdot Op_6$
- z is a new variable (i.e. z does not occur in φ, ψ, Op_1, Op_2)

(c) Op_1 and Op_2 can be represented as sequential compositions

$$Op_1 = Op_3 \cdot (\alpha \,!\, e_1) \cdot Op_4$$
$$Op_2 = Op_5 \cdot (\alpha \,!\, e_2) \cdot Op_6$$

where Op_3, Op_4, Op_5, Op_6 are internal COs, and the following inequality holds:

$$\varphi \leq \left\{ \begin{array}{l} (Op_3 \cdot Op_5)(e_1 = e_2) \\ (Op_3 \cdot Op_4 \cdot Op_5 \cdot Op_6)(\psi) \end{array} \right\}$$

Theorem 34.
Let P_1 and P_2 be processes with COs

$$P_i = (X_{P_i}, I_{P_i}, S_{P_i}, s^0_{P_i}, R_{P_i}) \qquad (i = 1, 2)$$

which have no common states and common variables.

Then $P_1 \approx P_2$, if there is a function μ of the form

$$\mu : S_{P_1} \times S_{P_2} \to Fm$$

which has the following properties.

1. $I_{P_1} \wedge I_{P_2} \leq \mu(s^0_{P_1}, s^0_{P_2})$.
2. For
   - each pair $(A_1, A_2) \in S_{P_1} \times S_{P_2}$, and
   - each transition $A_1 \xrightarrow{Op} A'_1$ of the process P_1, such that

   $$cond\,(Op) \;\wedge\; \mu(A_1, A_2) \neq \bot \tag{7.28}$$

   there is a set of CTs of the process P_2 starting from A_2

   $$\{\, A_2 \xrightarrow{CT_i} A^i_2 \mid i \in \Im\} \tag{7.29}$$

   satisfying the following conditions:

   (a) the following inequality holds:

   $$cond\,(Op) \;\wedge\; \mu(A_1, A_2) \;\leq\; \bigvee_{i \in \Im} cond\,(CT_i) \tag{7.30}$$

(b) for each $i \in \Im$ the following diagram holds:

$$\begin{tikzcd}
A_1 \arrow[r, dash, "\mu(A_1, A_2)"] \arrow[d, "Op"'] & A_2 \arrow[d, "CT_i"] \\
A_1' \arrow[r, dash, "\mu(A_1', A_2^i)"'] & A_2^i
\end{tikzcd} \qquad (7.31)$$

3. The property symmetrical to previous: for

   - each pair $(A_1, A_2) \in S_{P_1} \times S_{P_2}$, and
   - each transition $A_2 \xrightarrow{Op} A_2'$ of the process P_2, such that (7.28) holds

   there is a set of CTs of the process P_1 starting from A_1

   $$\{ A_1 \xrightarrow{CT_i} A_1^i \mid i \in \Im\} \qquad (7.32)$$

   satisfying the following conditions:

   (a) inequality (7.30) holds

   (b) for each $i \in \Im$ the following diagram holds:

$$\begin{tikzcd}
A_1 \arrow[r, dash, "\mu(A_1, A_2)"] \arrow[d, "CT_i"'] & A_2 \arrow[d, "Op"] \\
A_1^i \arrow[r, dash, "\mu(A_1^i, A_2')"'] & A_2'
\end{tikzcd} \qquad (7.33)$$

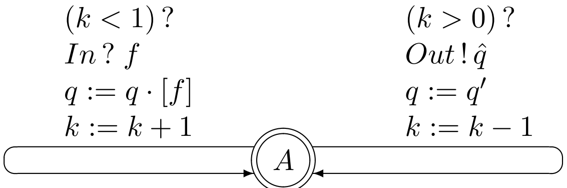
7.8.13 An example of a proof of observational equivalence of processes with COs

As an example of a use of theorem 34 prove that

$$\mathit{Buffer}_1 \approx \mathit{Buf}$$

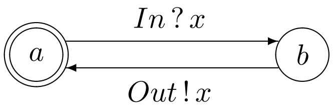
where

- is the considered above process Buffer_n (see (7.25)) for $n = 1$, i.e. a process of the form



  its initial condition is $(k = 0) \wedge (q = \varepsilon)$, and

- Buf is a process of the form



  The initial condition of this process is $\top$.

Define a function $\mu : \{A\} \times \{a, b\} \to Fm$ as follows:

$$\mu(A, a) \overset{\text{def}}{=} (k = 0) \wedge (q = \varepsilon)$$
$$\mu(A, b) \overset{\text{def}}{=} (k = 1) \wedge (q = [x])$$

Check properties 1, 2, and 3 for the function μ.

1. Property 1 in this case is the inequality

$$((k = 0) \wedge (q = \varepsilon)) \wedge \top \leq ((k = 0) \wedge (q = \varepsilon))$$

   which is obviously true.

2. Check property 2.

   - For the pair (A, a) we have to consider a left transition in the process *Buffer*$_1$ (because for the right transition (7.28) does not satisfied).

     As (7.29) we take the set consisting of a single transition from a to b.

     Diagram (7.31) in this case has the form

$$\begin{tikzcd}[column sep=huge, row sep=huge]
A \arrow[r, dash, "(k=0)\wedge(q=\varepsilon)"] \arrow[d, "\begin{array}{l} k<1 \\ In\,?\,f \\ q := q\cdot[f] \\ k := k+1 \end{array}"'] & a \arrow[d, "In\,?\,x"] \\
A \arrow[r, dash, "(k=1)\wedge(q=[x])"'] & b
\end{tikzcd} \tag{7.34}$$

     Using the fact that

$$\forall\, \varphi,\psi,\theta \in Fm \qquad (\varphi \leq \psi \to \theta \quad \Leftrightarrow \quad \varphi \wedge \psi \leq \theta) \tag{7.35}$$

     write the inequality corresponding to this diagram in the form

$$\left\{\begin{array}{c} k=0 \\ q=\varepsilon \\ k<1 \end{array}\right\} \leq \left\{\begin{array}{c} k+1=1 \\ q\cdot[z]=[z] \end{array}\right\} \tag{7.36}$$

     Clearly, this inequality is true.

   - For the pair (A, b) we have to consider only the right transition in the process *Buffer*$_1$ (because the left transition does not satisfied condition (7.28)).

     A (7.29) we take the set consisting of a single transition from b to a.

Diagram (7.31) in this case has the form

$$\begin{tikzcd}[row sep=huge, column sep=huge]
A \arrow[r, dash, "(k=1)\wedge(q=[x])"] \arrow[d, "\begin{array}{l} k>0 \\ Out\,!\,\hat{q} \\ q:=q' \\ k:=k-1 \end{array}"'] & b \arrow[d, "Out\,!\,x"] \\
A \arrow[r, dash, "(k=0)\wedge(q=\varepsilon)"'] & a
\end{tikzcd} \tag{7.37}$$

Using (7.35), write the inequality corresponding to this diagram in the form

$$\left\{\begin{array}{l} k=1 \\ q=[x] \\ k>0 \end{array}\right\} \leq \left\{\begin{array}{l} \hat{q}=x \\ k-1=0 \\ q'=\varepsilon \end{array}\right\} \tag{7.38}$$

Obviously, this inequality is true.

3. Check property 3.

- For the pair (A,a) and for a single transition from a to b as (7.32) we take the set, consisting of a left transition from A to A.

  Diagram (7.33) in this case has the form (7.34). As already established, this diagram is correct.

- For the pair (A,b) and for a single transition from b to a as (7.32) we take the set, consisting of the right transition from A to A.

  Daigram (7.33) in this case has the form (7.37). As already justified, this diagram is correct.

7.8.14 Additional remarks

To improve a usability of theorem 34 you can use the following notions and statements.

Invariants of processes

Let P be a process with CO.

A formula Inv with variables from X_P is said to be an **invariant** of the process P, if it has the following properties.

- $I_P \leq Inv$
- for each transition $s \xrightarrow{Op} s'$ of the process P
  - if Op is internal, then $Inv \leq Op(Inv)$
  - if Op is an input CO of the form $Op_1 \cdot (\alpha\,?\,x) \cdot Op_2$, then

    $$Inv \leq (Op_1 \cdot (x := z) \cdot Op_2)(Inv)$$

    where z is a variable which does not belong to X_P
  - if Op is an output CO of the form $Op_1 \cdot (\alpha\,!\,e) \cdot Op_2$, then

    $$Inv \leq (Op_1 \cdot Op_2)(Inv)$$

Using the concept of an invariant, theorem 34 can be modified as follows.

Theorem 35 .
Let

- P_1 and P_2 be two processes with COs:

  $$P_i = (X_{P_i}, I_{P_i}, S_{P_i}, s^0_{P_i}, R_{P_i}) \qquad (i = 1, 2)$$

  which have no common states and common variables, and
- formulas Inv_1 and Inv_2 are invariants of the processes P_1 and P_2 respectively.

Then $P_1 \approx P_2$, if there is a function μ of the form

$$\mu : S_{P_1} \times S_{P_2} \to Fm$$

with the following properties.

1. $I_{P_1} \wedge I_{P_2} \leq \mu(s^0_{P_1}, s^0_{P_2})$.

2. For

   - each pair $(A_1, A_2) \in S_{P_1} \times S_{P_2}$, and
   - each transition $A_1 \xrightarrow{Op} A_1'$ of the process P_1, such that

$$\left\{ \begin{array}{l} cond\,(Op) \\ \mu(A_1, A_2) \\ Inv_1 \\ Inv_2 \end{array} \right\} \neq \bot \tag{7.39}$$

   there is a set of CTs of the process P_2 with the start A_2

$$\{\, A_2 \xrightarrow{CT_i} A_2^i \mid i \in \Im\} \tag{7.40}$$

   satisfying the following conditions:

   (a) the following inequality holds:

$$\left\{ \begin{array}{l} cond\,(Op) \\ \mu(A_1, A_2) \\ Inv_1 \\ Inv_2 \end{array} \right\} \leq \bigvee_{i \in \Im} cond\,(CT_i) \tag{7.41}$$

   (b) for each $i \in \Im$ the following diagram is correct

$$\begin{tikzcd} A_1 \arrow[r, dash, "\left\{ \begin{array}{l} \mu(A_1, A_2) \\ Inv_1 \\ Inv_2 \end{array} \right\}"] \arrow[d, "Op"'] & A_2 \arrow[d, "CT_i"] \\ A_1' \arrow[r, dash, "\mu(A_1', A_2^i)"'] & A_2^i \end{tikzcd} \tag{7.42}$$

3. The property, which is symmetrical to the previous one: for

- each pair $(A_1, A_2) \in S_{P_1} \times S_{P_2}$, and
- each transition $A_2 \xrightarrow{Op} A_2'$ of the process P_2, such that (7.39) holds,

there is a set of CTs of the process P_1 with the start A_1

$$\{\, A_1 \xrightarrow{CT_i} A_1^i \mid i \in \Im\} \tag{7.43}$$

satisfying the following conditions:

(a) the inequality (7.41) holds

(b) for each $i \in \Im$ the following diagram is correct

$$\begin{tikzcd}
A_1 \arrow[r, dash, "\left\{\begin{array}{l}\mu(A_1, A_2)\\ Inv_1\\ Inv_2\end{array}\right\}"] \arrow[d, "CT_i"'] & A_2 \arrow[d, "Op"] \\
A_1^i \arrow[r, dash, "\mu(A_1^i, A_2')"'] & A_2'
\end{tikzcd} \tag{7.44}$$

Composition of diagrams

Theorem 36 .

Let

- φ, ψ, θ be formulas
- Op_1, Op_2 be internal COs, such that the following diagram is correct

$$\begin{tikzcd}
A \arrow[r, dash, "\varphi"] \arrow[d, "Op_1"'] & B \arrow[d, "Op_2"] \\
C \arrow[r, dash, "\psi"'] & D
\end{tikzcd}$$

- Op'_1, Op'_2 be COs such that the following diagram is correct

$$\begin{tikzcd}
C \arrow[r, dash, "\psi"] \arrow[d, "Op'_1"'] & D \arrow[d, "Op'_2"] \\
E \arrow[r, dash, "\theta"'] & F
\end{tikzcd}$$

- $\{Op_1, Op'_1\}$ and $\{Op_2, Op'_2\}$ have no common variables.

Then the following diagram is correct

$$\begin{tikzcd} A \arrow[r, dash, "\varphi"] \arrow[d, "Op_1 \cdot Op'_1"'] & B \arrow[d, "Op_2 \cdot Op'_2"] \\ E \arrow[r, dash, "\theta"'] & F \end{tikzcd}$$

7.8.15 Another example of a proof of observational equivalence of processes with COs

As an example of a use of theorems from section 7.8.14 prove an observational equivalence of

- the process

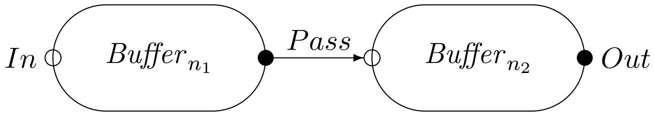
  $$(Buffer_{n_1}[Pass/Out] \mid Buffer_{n_2}[Pass/In]) \setminus \{Pass\} \qquad (7.45)$$

  where $Pass \notin \{In, Out\}$, and

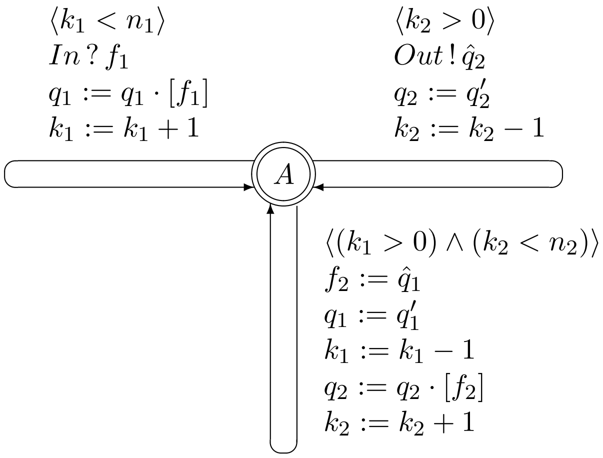
- the process $Buffer_{n_1+n_2}$.

Process (7.45) is a sequential composition of two buffers, size of which is n_1 and n_2 respectively.

A flow graph of this process has the form



According to the definition of operations on processes with COs (see section 7.8.5), a graph representation of the process (7.45) has the form

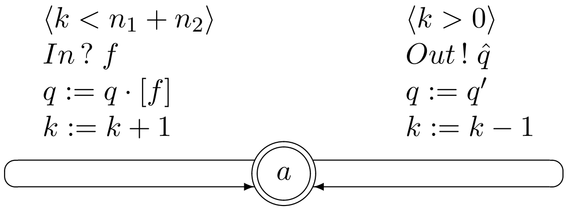


(7.46)

An initial condition of the process (7.46) is the formula

$$\left\{ \begin{array}{l} (n_1 > 0) \ \wedge \ (k_1 = 0) \ \wedge \ (q_1 = \varepsilon) \\ (n_2 > 0) \ \wedge \ (k_2 = 0) \ \wedge \ (q_2 = \varepsilon) \end{array} \right\}$$

A graph representation of the process $Buffer_{n_1+n_2}$ has the form



An initial condition of the process $Buffer_{n_1+n_2}$ is the formula

$$(n_1 + n_2 > 0) \ \wedge \ (k = 0) \ \wedge \ (q = \varepsilon)$$

It is easy to verify that the formula

$$Inv \overset{\text{def}}{=} \left\{ \begin{array}{l} 0 \leq k_1 \leq n_1 \\ |q_1| = k_1 \\ 0 \leq k_2 \leq n_2 \\ |q_2| = k_2 \\ n_1 > 0 \\ n_2 > 0 \end{array} \right\}$$

is an invariant of the process (7.46). This fact follows, in particular, from the statement

$$\left\{\begin{array}{l} |u| > 0 \quad \Rightarrow \quad |u'| = |u| - 1 \\ |u \cdot [a]| = |[a] \cdot u| = |u| + 1 \end{array}\right.$$

which hold for each list u and each message a.

As an invariant of the second process we take the formula $\top$.

Define a function $\mu : \{A\} \times \{a\} \to Fm$ as follows:

$$\mu(A, a) \stackrel{\text{def}}{=} \left\{\begin{array}{c} q = q_2 \cdot q_1 \\ k = k_2 + k_1 \end{array}\right\}$$

Check properties 1, 2, and 3 for the function μ.

1. Property 1 in this case is the inequality

$$\left\{\begin{array}{l} (n_1 > 0) \ \wedge \ (k_1 = 0) \ \wedge \ (q_1 = \varepsilon) \\ (n_2 > 0) \ \wedge \ (k_2 = 0) \ \wedge \ (q_2 = \varepsilon) \\ (n_1 + n_2 > 0) \ \wedge \ (k = 0) \ \wedge \ (q = \varepsilon) \end{array}\right\} \leq \left\{\begin{array}{c} q = q_2 \cdot q_1 \\ k = k_2 + k_1 \end{array}\right\}$$

   which is obviously true.

2. Check property 2.

   - For the left transition of the process (7.46) inequality (7.39) holds. As (7.40) we take the set, the only element of which is the left transition of the process $Buffer_{n_1+n_2}$.
     Inequality (7.41) in this case has the form

$$\left\{\begin{array}{l} k_1 < n_1 \\ q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \end{array}\right\} \leq (k < n_1 + n_2)$$

     that is obviously true.
     Using (7.35), write an inequality corresponding to diagram (7.42) for this case as

$$\left\{\begin{array}{l} q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \\ k_1 < n_1 \\ k < n_1 + n_2 \end{array}\right\} \leq \left\{\begin{array}{c} q \cdot [z] = q_2 \cdot q_1 \cdot [z] \\ k + 1 = k_2 + k_1 + 1 \end{array}\right\} \qquad (7.47)$$

     It is easy to check that the last inequality is true.

- For the middle (internal) transition of the process (7.46) inequality (7.39) holds. As (7.40) we take the set, the only element of which is an empty CT of the process $\mathit{Buffer}_{n_1+n_2}$.

  Inequality (7.41) in this case holds for the trivial reason: its right side is $\top$.

  Using statement (7.35), write an inequality corresponding to diagram (7.42) for this case, in the form

$$\left\{\begin{array}{l} q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \\ k_1 > 0 \\ k_2 < n_2 \end{array}\right\} \leq \left\{\begin{array}{l} q = (q_2 \cdot [\hat{q}_1]) \cdot q_1' \\ k = k_2 + 1 + k_1 - 1 \end{array}\right\} \tag{7.48}$$

  This inequality follows from

  - the associativity property of of a concatenation, and
  - the statement

$$|u| > 0 \quad \Rightarrow \quad u = [\hat{u}] \cdot u'$$

    which holds for each list u.

- For the right transition of the process (7.46) inequality (7.39) holds. A (7.40) we take the set, the only element of which is the right transition of the process $\mathit{Buffer}_{n_1+n_2}$.

  Inequality (7.41) in this case has the form

$$\left\{\begin{array}{l} k_2 > 0 \\ q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \end{array}\right\} \leq (k > 0)$$

  that is obviously true.

  Using the statement (7.35), we write the inequality which corresponds to diagram (7.42) for this case, in the form

$$\left\{\begin{array}{l} q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \\ k_2 > 0 \\ k > 0 \end{array}\right\} \leq \left\{\begin{array}{l} \hat{q}_2 = \hat{q} \\ q' = q_2' \cdot q_1 \\ k - 1 = k_2 - 1 + k_1 \end{array}\right\} \tag{7.49}$$

This inequality follows from the statement

$$|u| > 0 \quad \Rightarrow \quad \left\{ \begin{array}{l} (u \cdot v)\hat{} = \hat{u} \\ (u \cdot v)' = u' \cdot v \end{array} \right\}$$

which holds for each pair of lists u, v.

3. Check property 3.

   - For the left transition of the process $Buffer_{n_1+n_2}$ inequality (7.39) holds. As (7.43) we take the set, consisting of two CTs:
     - the left transition of the process (7.46), and
     - the sequence, which consists of a pair of transitions
       - the first element of which is the middle (internal) transition of the procrss (7.46),
       - and the second is the left transition of the process (7.46)

     Inequality (7.41) in this case has the form

     $$\left\{ \begin{array}{l} k < n_1 + n_2 \\ q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \end{array} \right\} \leq (k_1 < n_1) \;\vee\; \left\{ \begin{array}{l} k_1 > 0 \\ k_2 < n_2 \\ k_1 - 1 < n_1 \end{array} \right\}$$

     This inequality is true, and in the proof of this inequality the conjunctive term $n_1 > 0$ (contained in Inv) is used.

     The inequalities which correspond to diagrams (7.44) for both elements of the set (7.43), follow from (7.47), (7.48) and theorem 36.

   - For the right transition of the process $Buffer_{n_1+n_2}$ inequality (7.39) holds. As (7.43) we take the set, consisting of two CTs:
     - the right transition of the process (7.46), and
     - the sequence which consists of a pair of transitions,
       - the first element of which is the middle (internal) transition of the process (7.46), and
       - the second is the right transition of the process (7.46)

Inequality (7.41) in this case has the form

$$\left\{\begin{array}{l} k > 0 \\ q = q_2 \cdot q_1 \\ k = k_2 + k_1 \\ Inv \end{array}\right\} \leq (k_2 > 0) \ \vee \ \left\{\begin{array}{l} k_1 > 0 \\ k_2 < n_2 \\ k_2 + 1 > 0 \end{array}\right\}$$

This inequality is true, and in the proof of this inequality the conjunctive term $n_2 > 0$ (contained in Inv) is used.

The inequalities corresponding to diagrams (7.44) for both elements of the set (7.43), follow from (7.48), (7.49) and theorem 36.

7.9 Recursive definition of processes

The concept of a **recursive definition** of processes with message passing is similar to the concept of a RD presented in chapter 5.

The concept of a RD is based on the concept of a **process expression (PE)** which is analogous to the corresponding concept in section 5.1, so we only point out differences in the definitions of these concepts.

- In all PEs operators are used (instead of actions).
- Each process name A has a **type** $t(A)$ of the form

  $$t(A) = (t_1, \ldots, t_n) \qquad (n \geq 0)$$

  where $\forall\, i = 1, \ldots, n \quad t_i \in Types$
- Each process name A occurs in each PE only together with a list of expressions of corresponding types, i.e. each occurrence of A in each PE P is contained in a subexpression of P of the form

  $$A(e_1, \ldots, e_n)$$

  where

  - $\forall\, i = 1, \ldots, n \quad e_i \in \mathcal{E}$
  - $(t(e_1), \ldots, t(e_n)) = t(A)$

For each PE P the notation $fv(P)$ denotes a set of **free variables** of P, which consists of all variables from X_P having free occurrences in P.

The concepts of a free occurrence and a bound occurrence of a variable in a PE is similar to the analogous concept in predicate logic. Each free occurrence of a variable x in a PE P becomes bound in the PEs $(\alpha?x).P$ and $(x := e).P$.

A **recursive definition (RD) of processes** is a list of formal equations of the form

$$\left\{\begin{array}{l} A_1(x_{11}, \ldots, x_{1k_1}) = P_1 \\ \ldots \\ A_n(x_{n1}, \ldots, x_{nk_n}) = P_n \end{array}\right. \tag{7.50}$$

where

- $A_1, \ldots, A_n$ are process names,
- for each $i = 1, \ldots, n$ the list $(x_{i1}, \ldots, x_{ik_i})$ in the left side of i–th equality consists of different variables
- $P_1, \ldots, P_n$ are PEs, which satisfy
  - the conditions set out in the definition of a RD in section 5.2, and
  - the following condition:

$$\forall\, i = 1, \ldots, n \quad fv(P_i) = \{x_{i1}, \ldots, x_{ik_i}\}$$

We shall assume that for each process name A there is a unique RD such that A has an occurrence in this RD.

RD (7.50) can be interpreted as a functional program, consisting of functional definitions. For each $i = 1, \ldots, n$ the variables x_{i1}, …, x_{ik_i} can be regarded as formal parameters of the function $A_i(x_{i1}, \ldots, x_{ik_i})$.

A reader is requested to define a correspondence, which associates with each PE of the form $A(x_1, \ldots, x_n)$, where

- A is a process name, and
- $x_1, \ldots, x_n$ is a list of different variables of appropriate types

the process

$$[\![A(x_1, \ldots, x_n)]\!] \tag{7.51}$$

Also a reader is invited to investigate the following problems.

1. Construction of minimal processes which are equivalent ($\approx, \overset{+}{\approx}, \ldots$) to processes of the form (7.51).

2. Recognition of equivalence of processes of the form (7.51).

3. Finding necessary and sufficient conditions of uniqueness of the list of processes defined by a RD.

Chapter 8

Examples of processes with a message passing

8.1 Separation of sets

8.1.1 The problem of separation of sets

Let U, V be a pair of finite disjoint sets, with each element $x \in U \cup V$ is mapped to a number $weight(x)$, called a **weight** of this element.

It is need to convert this pair in a pair of sets U', V', so that

- $|U| = |U'|, \quad |V| = |V'|$
  (for each finite set M the notation $|M|$ denotes a number of elements in M)
- for each $u \in U'$ and each $v \in V'$ the following inequality holds:

$$weight(u) \leq weight(v)$$

Below we shall call the sets U and V as the left set and the right set, respectively.

8.1.2 Distributed algorithm of separation of sets

The problem of separation of sets can be solved by an execution of several sessions of exchange elements between these sets. Each session consists of the following actions:

- find an element mx with a maximum weight in the left set
- find an element mn with minimum weight in the right set
- transfer
  - mx from the left set to the right set, and
  - mn from the right set to the left set.

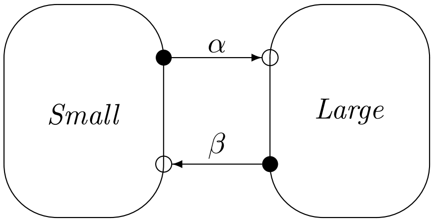
To implement this idea it is proposed a distributed algorithm, defined as a process of the form

$$(Small \mid Large) \setminus \{\alpha, \beta\} \tag{8.1}$$

where

- the process *Small* executes operations associated with the left set, and
- the process *Large* executes operations associated with the right set.

A flow graph corresponding to this process has the form



Below we shall use the following notations:

- for each subset $W \subseteq U \cup V$ the notations

  $$\max(W) \text{ and } \min(W)$$

  denote an element of W with maximum and minimum weight, respectively,
- for
  - any subsets $W_1, W_2 \subseteq U \cup V$, and

- any $u \in U \cup V$

the notations

$$W_1 \leq u, \quad u \leq W_1, \quad W_1 \leq W_2$$

are shorthand expressions

$$\begin{array}{l} \forall x \in W_1 \quad weight(x) \leq weight(u) \\ \forall x \in W_1 \quad weight(u) \leq weight(x) \\ \forall x \in W_1,\ \forall y \in W_2 \quad weight(x) \leq weight(y) \end{array}$$

respectively.

A similar meaning have the expressions

$$\max(W), \quad \min(W), \quad W \leq u, \quad u \leq W, \quad W_1 \leq W_2$$

in which the symbols W, W_i and u denote variables whose values are

- subsets of the set $U \cup V$, and
- elements of the set $U \cup V$

respectively.

8.1.3 The processes *Small* and *Large*

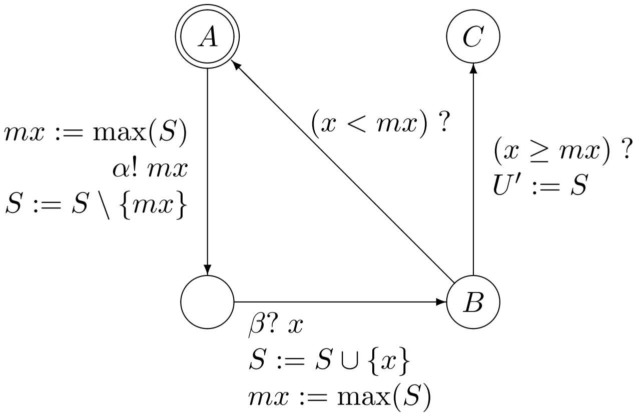
The processes *Small* and *Large*

- can be defined in terms of flowcharts,
- then are transformed to the processes with COs, and
- reduce it.

We will not describe theseflowcharts and their transformations and reductions, we present only the reduced COs.

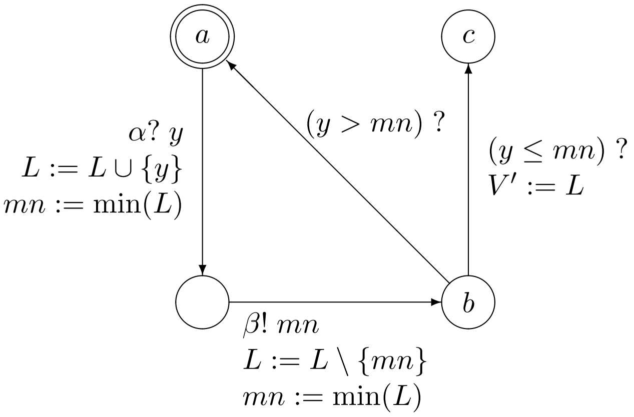
The reduced process *Small* has the following form.

$Init = (S = U)$.



(8.2)

The reduced process *Large* has the following form.
$Init = (L = V)$.



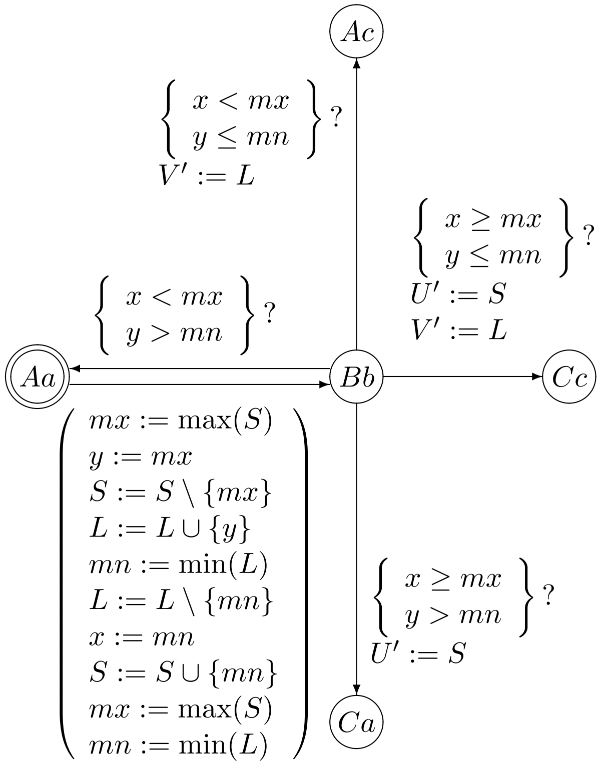
(8.3)

8.1.4 An analysis of the algorithm of separation of sets

The process described by the expression (8.1), is obtained by

- a performing of operations of parallel composition and restrictions on the processes (8.2) and (8.3), in accordance with definition (8.1), and
- a reduction of the resulting process.

The reduced process has the following form:


Ac
$\left\{\begin{array}{l} x < mx \\ y \leq mn \end{array}\right\}?$
$V' := L$
$\left\{\begin{array}{l} x \geq mx \\ y \leq mn \end{array}\right\}?$
$U' := S$
$V' := L$
$\left\{\begin{array}{l} x < mx \\ y > mn \end{array}\right\}?$
Aa
Bb
Cc
$mx := \max(S)$
$y := mx$
$S := S \setminus \{mx\}$
$L := L \cup \{y\}$
$mn := \min(L)$
$L := L \setminus \{mn\}$
$x := mn$
$S := S \cup \{mn\}$
$mx := \max(S)$
$mn := \min(L)$
$\left\{\begin{array}{l} x \geq mx \\ y > mn \end{array}\right\}?$
$U' := S$
Ca

(8.4)

This diagram shows that there are states of the process (8.4) (namely, Ac and Ca) with the following properties:

- there is no transitions starting at these states
  (such states are said to be **terminal**)
- but falling into these states is not a normal completion of the process.

The situation when a process falls in one of such states is called a **deadlock**.

The process (8.1) can indeed fall in one of such states, for example, in the case when

$$U = \{3\} \quad \text{and} \quad V = \{1, 2\}$$

where a weight of each number coincides with its value.

Nevertheless, the process (8.1) has the following properties:

- this process always terminates (i.e., falls into one of the terminal states - Ac, Cc or Ca)

- after a termination of the process, the following statements hold:

$$\left.\begin{array}{l} S \cup L = U \cup V \\ |S| = |U|, \quad |L| = |V| \\ S \leq L \end{array}\right\} \tag{8.5}$$

To justify these properties, we shall use the function

$$f(S, L) \overset{\text{def}}{=} \mid \{(s, l) \in S \times L \mid weight(s) > weight(l)\} \mid$$

Furthermore, for an analyzing of a sequence of assignment operators performed during the transition from Aa to Bb, it is convenient to represent this sequence schematically as a sequence of the following actions:

1. $S \xrightarrow{y:=\max(S)} L$
   (transfer of an element $y := \max(S)$ from S to$L$)
2. $L \xrightarrow{x:=\min(L)} S$
3. $mx := \max(S)$
4. $mn := \min(L)$

It is not so difficult to prove the following statements.

1. If at a current time i

   - the process is in the state Aa, and
   - values S_i, L_i of the variables S and L at this time satisfy the equation
   $$f(S_i, L_i) = 0$$
   i.e. the inequality $S_i \leq L_i$ holds

   then $S_{i+1} = S_i$ and $L_{i+1} = L_i$.

   Furthermore, after an execution of the transition from Aa to Bb values of the variables x, y, mx and mn will satisfy the following statement:

   $$y = x = mx \leq mn$$

   and, thus, a next transition will be the transition from Bb to state Cc, i.e. the process normally completes its work.

   Herewith

- values of the variables U' and V' will be equal to S_i and L_i, respectively,
- and, consequently, values of the variables U' and V' will meet the required conditions

$$|U| = |U'|, \quad |V| = |V'|, \quad U' \leq V'$$

2. If at a current time i

   - the process is in the state Aa, and
   - values S_i, L_i of the variables S and L satisfy the inequality

   $$f(S_i, L_i) > 0$$

   then after an execution of the transition from Aa to Bb (i.e., at the time $i+1$) new values S_{i+1}, L_{i+1} of the variables S and L will satisfy the inequality

   $$f(S_{i+1}, L_{i+1}) < f(S_i, L_i) \tag{8.6}$$

   In addition, the variables x, y, mx, mn at the time $i+1$ will satisfy

   $$\begin{array}{l} y = \max(S_i), \quad x = \min(L_i) \\ mx = \max(S_{i+1}), \quad mn = \min(L_{i+1}) \\ x < y, \quad x \leq mx, \quad mn \leq y \end{array}$$

   It follows that if at the time $i+1$ the process will move from Bb to one of the terminal states (Ac, Cc or Ca), then it is possible

   (a) either if $x = mx$

   (b) or if $y = mn$

   In the case (a) the following statement holds:

   $$S_{i+1} \leq mx = x \leq L_i$$

   whence, using

   $$x < y \quad \text{and} \quad L_{i+1} \subseteq L_i \cup \{y\}$$

   we obtain:

   $$S_{i+1} \leq L_{i+1} \tag{8.7}$$

In the case (b) the following statement holds:

$$S_i \leq y = mn \leq L_{i+1}$$

whence, using

$$x < y \quad \text{and} \quad S_{i+1} \subseteq S_i \cup \{x\}$$

we obtain (8.7).

Thus, if the process is in a terminal state, then $S \leq L$.

Other statements listed in (8.5) are proved directly.

The first and second statements imply that this process can not be endless, because an infinite loop is possible only in the case when

- the process infinitely many times falls into the state Aa, and
- every time when the process is in the state Aa, a value of the function f on current values of the variables S, T is positive.

An impossibility of this situation follows from

- inequality (8.6), and
- the founding property of the set of natural numbers
(there is no an infinite descending chain of natural numbers)

A reader is requested

- to find necessary and sufficient conditions to be met by the shared sets U and V, that there is no a deadlock situation in an execution of process (8.4) (i.e. the process terminates in the state Cc) with these U and V, and
- develop an algorithm for separation of sets that would work without a deadlock on any shared sets U and V.

8.2 Calculation of a square

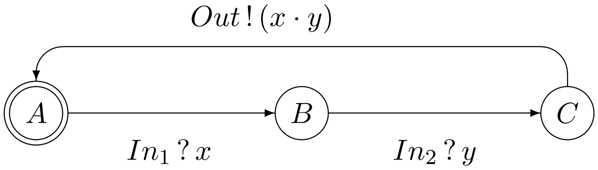
Suppose we have a system "multiplier", which has

- two input ports with the names In_1 and In_2, and
- one output port with the name Out.

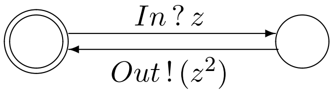
An execution of the multiplier is that it

- receives at its input ports two values, and
- gives their product on the output port.

A behavior of the multiplier is described by the process Mul:



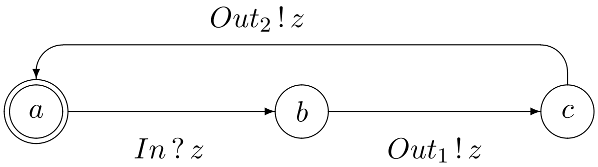
Using this multiplier, we want to build a system "calculator of a square", whose behavior is described by the process $Square\_Spec$:



The desired system we shall build as a composition of

- the auxiliary system "duplicator" having
  - an input port In, and
  - output ports Out_1 and Out_2

  behavior of which is described by the process Dup:



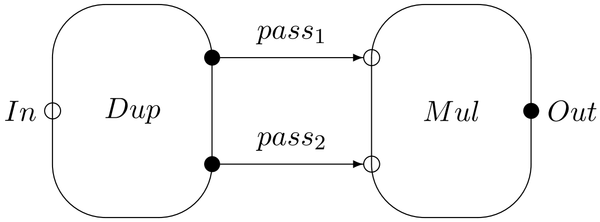
i.e. the duplicator copies its input to two outputs, and

- the multiplier, which receives on its input ports those values that duplicator gives.

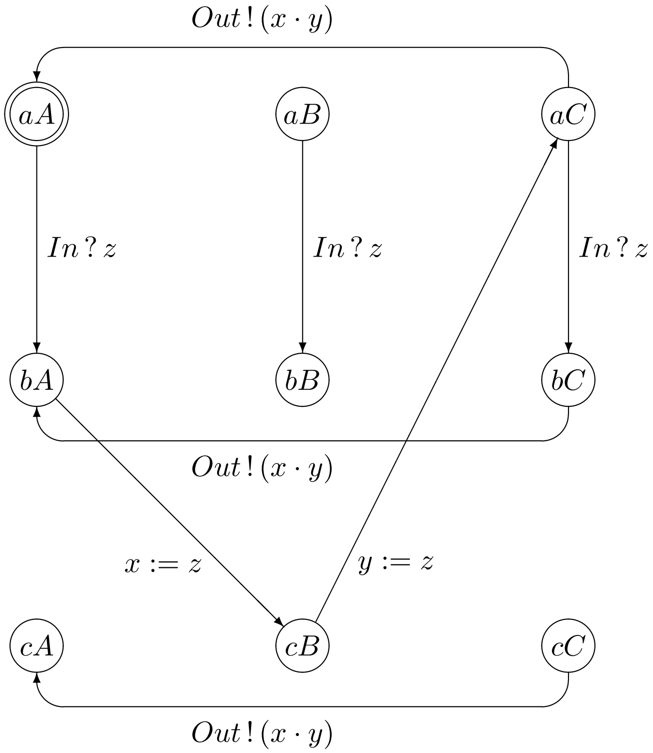
A process *Square*, corresponding to such a composition is determined as follows:

$$\begin{array}{l} Square \overset{\text{def}}{=} \\ \overset{\text{def}}{=} \left(\begin{array}{l} Dup[pass_1/Out_1, pass_2/Out_2] \ \mid \\ \mid Mul[pass_1/In_1, pass_2/In_2] \end{array} \right) \setminus \{pass_1, pass_2\} \end{array}$$

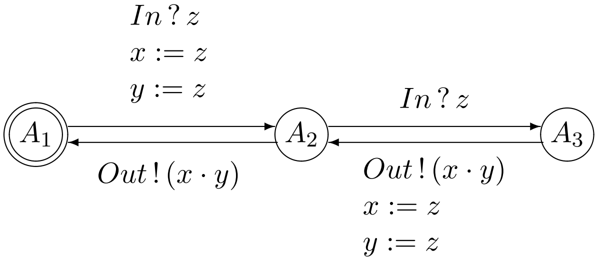
A flow graph of the process *Square* has the form (



However, the process *Square* does not meet the specification *Square_Spec*. This fact is easy to detect by a construction of a graph representation of *Square*, which, by definition operations of parallel composition, restriction and renaming, is the following:



After a reduction of this process we obtain the diagram



(8.8)

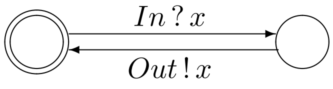
which shows that

- the process *Square* can execute two input actions together (i.e. without an execution of an output action between them), and
- the process *Square_Spec* can not do so.

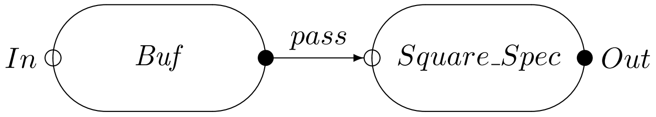
The process *Square* meets another specification:

$$Square\_Spec' \overset{\text{def}}{=} \left(\begin{array}{l} Buf[pass/Out] \ \mid \\ \mid Square\_Spec[pass/In] \end{array} \right) \setminus \{pass\}$$

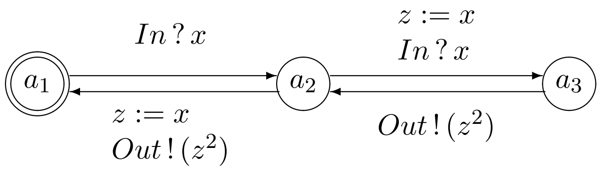
where *Buf* is a buffer which can store one message, whose behavior is represented by the diagram



A flow graph of $Square\_Spec'$ has the form



A reduced process $Square\_Spec'$ has the form

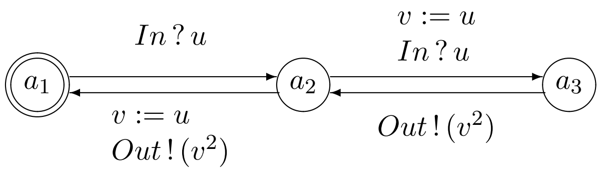


(8.9)

The statement that *Square* meets the specification $Square\_Spec'$ can be formalized as

$$(8.8) \ \approx (8.9) \tag{8.10}$$

We justify (8.10) with use of theorem 34. At first, we rename variables of the process (8.9), i.e. instead of (8.9) we shall consider the process



(8.11)

To prove (8.8) $\approx$ (8.11) with use of theorem 34 we define the function

$$\mu : \{A_1, A_2, A_3\} \times \{a_1, a_2, a_3\} \to Fm$$

as follows:

- $\mu(A_i, a_j) \overset{\text{def}}{=} \bot$, if $i \neq j$
- $\mu(A_1, a_1) \overset{\text{def}}{=} \top$
- $\mu(A_2, a_2) \overset{\text{def}}{=} (x = y = z = u)$
- $\mu(A_3, a_3) \overset{\text{def}}{=} \left\{ \begin{array}{l} x = y = v \\ z = u \end{array} \right\}$

Detailed verification of correctness of corresponding diagrams left to a reader as a simple exercise.

8.3 Petri nets

One of mathematical models to describe a behavior of distributed systems is a **Petri net**.

A **Petri net** is a directed graph, whose set of nodes is divisible in two classes: places (V) and transitions (T). Each edge connects a place with a transition.

Each transition $t \in T$ is associated with two sets of places:

- $in(t) \overset{\text{def}}{=} \{v \in V \mid \text{ there is an edge from } v \text{ to } t\}$
- $out(t) \overset{\text{def}}{=} \{v \in V \mid \text{there is an edge from } t \text{ to } v\}$

A **marking** of a Petri net is a mapping ξ of the form

$$\xi : V \to \{0, 1, 2, \ldots\}$$

An **execution** of a Petri net is a transformation of its marking which occurs as a result of an execution of transitions.

A marking ξ_0 at time 0 is assumed to be given.

If a net has a marking ξ_i at a time i, then any of transition $t \in T$, which satisfies the condition

$$\forall\, v \in in(t) \quad \xi_i(v) > 0$$

can be executed at time i.

If a transition t was executed at time i , then a marking ξ_{i+1} at time $i+1$ is defined as follows:

$$\begin{array}{l}
\forall\, v \in in(t) \quad \xi_{i+1}(v) := \xi(v) - 1 \\
\forall\, v \in out(t) \quad \xi_{i+1}(v) := \xi(v) + 1 \\
\forall\, v \in V \setminus (in(t) \cup out(t)) \quad \xi_{i+1}(v) := \xi(v)
\end{array}$$

Each Petri net $\mathcal{N}$ can be associates with a process $P_{\mathcal{N}}$, which simulates a behavior of this net. Components of the process $P_{\mathcal{N}}$ are as follows.

- - $X_{P_{\mathcal{N}}} \overset{\text{def}}{=} \{x_v \mid v \in V\}$,
  - $I_{P_{\mathcal{N}}} \overset{\text{def}}{=} \bigwedge\limits_{v \in V} (x_v = \xi_0(v))$,
  - $S_{P_{\mathcal{N}}} \overset{\text{def}}{=} \{s^0\}$
- Let t be a transition of the net $\mathcal{N}$, and the sets $in(t)$ and $out(t)$ have the form $\{u_1, \ldots, u_n\}$ and $\{v_1, \ldots, v_m\}$ respectively.

  Then the process $P_{\mathcal{N}}$ has a transition from s^0 to s^0 with the label

$$\left(\begin{array}{l}
(x_{u_1} > 0) \wedge \ldots \wedge (x_{u_n} > 0)\ ? \\
x_{u_1} := x_{u_1} - 1, \ldots, x_{u_n} := x_{u_n} - 1 \\
x_{v_1} := x_{v_1} + 1, \ldots, x_{v_m} := x_{v_m} + 1
\end{array}\right)$$

Chapter 9

Communication protocols

In this chapter we consider an application of the theory of processes to the problem of modeling and verification of communication protocols (which are called below **protocols**).

9.1 The concept of a protocol

A **protocol** is a distributed system which consists of several interacting components, including

- components that perform a formation, sending, receiving and processing of messages
  (such components are called **agents**, and messages sent from one agent to another, a called **frames**)
- components of an environment, through which frames are forwarded
  (such an environment is usually called a **communication channel**).

There are several layers of protocols. In this chapter we consider data link layer protocols.

9.2 Frames

9.2.1 The concept of a frame

Each frame is a string of bits.

Passing through an environment, a frame may be distorted or lost (a distortion of a frame is an inverting some bits of this frame). Therefore, each frame must contain

- not only an information which one agent wishes to transfer to another agent, but
- means allowing to a recipient of the frame to find out whether this frame is distorted during a transmission.

Below we consider some methods of detection of distortions in frames. These methods are divided into two classes:

1. methods which allow
    - not only detect distortions of frames,
    - but also determine distorted bits of a frame and fix them

   (discussed in section 9.2.2), and
2. methods to determine only a fact of a distortion of a frame (discussed in section 9.2.3).

9.2.2 Methods for correcting of distortions in frames

Methods of detection of distortion in frames, which allow

- not only detect the fact of a distortion, but
- determine indexes of distorted bits

are used in such situations, when a probability that each transmitted frame will be distorted in a transmission of this frame, is high. For example, such a situation occurs in wireless communications.

If you know a maximum number of bits of a frame which can be inverted, then for a recognition of inverted bits and their correction methods of **error correction coding** can be used. These methods constitute one of directions of the **coding theory**.

In this section we consider an encoding method with correction of errors in a simplest case, when in a frame no more than one bit can be inverted. This method is called a **Hamming code** to correct one error (there are Hamming codes to fix an arbitrary number of errors).

The idea of this method is that bits of a frame are divided into two classes:

- information bits (which contain an information which a sender of the frame wants to convey to the recipient), and
- control bits (values of which are computed on values of information bits).

Let

- f be a frame of the form $(b_1, \dots, b_n)$
- k is a number of information bits in f
- r is a number of control bits in f
(i.e. $n = k + r$)

Since a sender can place his information in k information bits, then we can assume that an information that a sender sends to a recipient in a frame f, is a string M, which consists of k bits.

A frame which is derived from the string M by addition of control bits, we denote by $\varphi(M)$.

For each frame f denote by $U(f)$ the set of all frames obtained from f by inversion is not more than one bit. Obviously, a number of elements of $U(f)$ is equal to $n + 1$.

The assumption that during a transmission of the frame $\varphi(M)$ not more than one bit of this frame can be inverted, can be reformulated as follows: the recipient can receive instead of $\varphi(M)$ any frame from the set $U(\varphi(M))$.

It is easy to see that the following conditions are equivalent:

1. for each $M \in \{0,1\}^k$ a recipient can uniquely reconstruct M having an arbitrary frame from $U(\varphi(M))$
2. the family
$$\{U(\varphi(M)) \mid M \in \{0,1\}^k\} \tag{9.1}$$
of subsets of $\{0,1\}^n$ consists of disjoint subsets.

Since

- the family (9.1) consists of 2^k subsets, and
- each of these subsets consists of $n + 1$ elements

then a necessary condition of disjointness of subsets from (9.1) is the inequality

$$(n+1) \cdot 2^k \leq 2^n$$

which can be rewritten as

$$(k + r + 1) \leq 2^r \tag{9.2}$$

It is easy to prove that for every fixed $k > 0$ the inequality (9.2) (where r is assumed to be positive) is equivalent to the inequality

$$r_0 \leq r$$

where r_0 depends on k, and is a lower bound on the number of control bits.

It is easy to calculate r_0, when k has the form

$$k = 2^m - m - 1, \qquad \text{where } m \geq 1 \tag{9.3}$$

in this case (9.2) can be rewritten as the inequality

$$2^m - m \leq 2^r - r \tag{9.4}$$

which is equivalent to the inequality $m \leq r$ (because the function $2^x - x$ is monotone for $x \geq 1$).

Thus, in this case the lower bound r_0 of the number of control bits is m.

Below we present a coding method with correction of one error, in which a number r of control bits is equal to the minimum possible value m.

If k has the form (9.3), and $r = r_0 = m$, then $n = 2^m - 1$, i.e. indices of bits of the frame $f = (b_1, \dots, b_n)$ can be identified with m–tuples from $\{0,1\}^m$: each index $i \in \{1, \dots, n\}$ is identified with a binary record of i (which is complemented by zeros to the left, if it is necessary).

By definition, indices of control bits are m–tuples of the form

$$(0 \dots 0\,1\,0 \dots 0) \quad \text{(1 is at } j\text{–th position)} \tag{9.5}$$

where $j = 1, \dots, m$.

For each $j = 1, \dots, m$ a value of the control bit which has the index (9.5) is equal to the sum modulo 2 values of information bits, indices of which contain 1 at j-th position.

When a receiver gets a frame $(b_1, \dots, b_n)$ he checks m equalities

$$\sum_{i_j = 1} b_{i_1 \dots i_m} = 0 \qquad (j = 1, \dots, m) \tag{9.6}$$

(the sum is modulo 2).

The following cases are possible.

- The frame is not distorted.
  In this case, all the equalities (9.6) are correct.

- A control bit which has the index (9.5) is distorted.

  In this case only j–th equality in (9.6) is incorrect.

- An information bit (9.5) is distorted.

  Let an index of this bit contains 1 at the positions j_1, …, j_l.

  In this case among the equalities (9.6) only equalities with numbers j_1, …, j_l are incorrect.

Thus, in all cases, we can

- detect it whether the frame is distorted, and
- calculate an index of a distorted bit, if the frame is distorted.

9.2.3 Methods for detection of distortions in frames

Another class of methods for detection of distortions in frames is related to a detection of only a fact of a distortion.

The problem of a calculation of indexes of distorted bits has high complexity. Therefore, if a probability of a distortion in transmitted frames is low (that occurs when a copper or fibre communication channel is used), then more effective is a resending of distorted frames: if a receiver detects that a received frame is distorted, then he requests a sender to send the frame again.

For a comparison of a complexity of the problems of

- correcting of distortions, and
- detection of distortions (without correcting)

consider the following example. Suppose that no more than one bit of a frame can be distorted. If a size of this frame is 1000, then

- for a *correction* of such distortion it is needed 10 control bits, but
- for a *detection* of such distortion it is enough 1 control bit, whose value is assumed equal to the parity of the number of units in remaining bits of the frame.

One method of coding to detection of distortion is the following:

- a frame is divided into k parts, and
- in each part it is assigned one control bit, whose value is assumed equal to the parity of the number of units in remaining bits of this part.

If bits of the frame are distorted equiprobably and independently, then for each such part of the frame the probability that

- this part is distorted, and
- nevertheless, its parity is correct (i.e., we consider it as undistorted)

is less than 1/2, therefore a probability of undetected distortion is less than 2^{-k}.

Another method of coding to detection of distortions is a **polynomial code** (which is called **Cyclic Redundancy Check, CRC**).

This method is based on a consideration of bit strings as polynomials over the field $\mathbf{Z}_2 = \{0, 1\}$: a bit string of the form

$$(b_k, b_{k-1}, \ldots, b_1, b_0)$$

is regarded as the polynomial

$$b_k \cdot x^k + b_{k-1} \cdot x^{k-1} + \ldots + b_1 \cdot x + b_0$$

Suppose you need to transfer frames of size $m+1$. Each such frame is considered as a polynomial $M(x)$ of a degree $\leq m$.

To encode these frames there are selected

- a number $r < m$, and
- a polynomial $G(x)$ of degree r, which has the form
$$x^r + \ldots + 1$$

The polynomial $G(x)$ is called a **generator polynomial**.

For each frame $M(x)$ its code $T(x)$ is calculated as follows. The polynomial $x^r \cdot M(x)$ is divided on $G(x)$ with a remainder:

$$x^r \cdot M(x) = G(x) \cdot Q(x) + R(x)$$

where $R(x)$ is a remainder (a degree of $R(x)$ is less than r).

A code of the frame $M(x)$ is the polynomial

$$T(x) \overset{\text{def}}{=} G(x) \cdot Q(x)$$

It is easy to see that a size of $T(x)$ is larger than a size of $M(x)$ on r.

Detection of a distortion in a transmission of the frame $T(x)$ is produced by a dividing a received frame $T'(x)$ on $G(x)$: we consider that the frame $T(x)$ was transmitted without a distortion (i.e. a received frame $T'(x)$ coincides with $T(x)$), if $T'(x)$ is divisible on $G(x)$ (i.e. $T'(x)$ has the form $G(x) \cdot Q'(x)$, where $Q'(x)$ is a polynomial).

If the frame $T(x)$ was transmitted without a distortion, then the original frame $M(x)$ can be recovered by a representation of $T(x)$ as a sum

$$T(x) = x^r \cdot M(x) + R(x)$$

where $R(x)$ consists of all monomials in $T(x)$ of a degree $< r$.

If the frame $T(x)$ was transmitted with distortions, then a relation between $T(x)$ and $T'(x)$ can be represented as

$$T'(x) = T(x) + E(x)$$

where $E(x)$ is a polynomial which

- is called a **polynomial of distortions**, and
- corresponds to a string of bits each component of which is equal to
  - 1 if the corresponding bit of the frame $T(x)$ has been distorted, and
  - 0, otherwise.

Thus

- if $T(x)$ has been distorted in a single bit, then $E(x) = x^i$
- if $T(x)$ has been distorted in two bits, then $E(x) = x^i + x^j$,
- etc.

From the definitions of $T'(x)$ and $E(x)$ it follows that $T'(x)$ is divisible on $G(x)$ if and only if $E(x)$ is divisible on $G(x)$.

Therefore, a distortion corresponding to the polynomial $E(x)$, can be detected if and only if $E(x)$ is not divisible on $G(x)$.

Let us consider the question of what kinds of distortions can be detected using this method.

1. A single-bit distortion can be detected always, because the polynomial $E(x) = x^i$ is not divisible on $G(x)$.
2. A double-byte distortion can not be detected in the case when the corresponding polynomial

$$E(x) = x^i + x^j = x^j \cdot (x^{i-j} + 1) \quad (i > j)$$

is divisible on $G(x)$:

$$\exists\, Q(x): \quad x^j \cdot (x^{i-j} + 1) = G(x) \cdot Q(x) \tag{9.7}$$

On the reason of a uniqueness of factorization of polynomials over a field the statement (9.7) implies the statement

$$\exists\, Q_1(x): \quad x^{i-j} + 1 = G(x) \cdot Q_1(x) \tag{9.8}$$

The following fact holds: if

$$G(x) = x^{15} + x^{14} + 1 \tag{9.9}$$

then for each $k = 1, \ldots, 32768$ the polynomial $x^k + 1$ is not divisible on $G(x)$.

Therefore the generator polynomial (9.9) can detect a double-byte distortion in frames of a size ≤ 32768.

3. Consider the polynomial of distortions $E(x)$ as a product of the form

$$E(x) = x^j \cdot (x^{k-1} + \ldots + 1) \tag{9.10}$$

The number k in (9.10) is called a **size of a packet of errors**. k is equal to the size of a substring of a string of distortions (which corresponds to $E(x)$), which is bounded from left and right by the bits “1”.

Let $E_1(x)$ be the second factor in (9.10).

On the reason of a uniqueness of factorization of polynomials over a field we get that

- a distortion corresponding to the polynomial (9.10) is not detected if and only if
- $E_1(x)$ is divisible on $G(x)$.

Consider separately the following cases.

(a) $k \leq r$, i.e. $k - 1 < r$.

In this case $E_1(x)$ is not divisible on $G(x)$, because a degree of $E_1(x)$ is less than a degree of $G(x)$.

Thus, in this case we can detect any distortion.

(b) $k = r + 1$.

In this case the polynomial $E_1(x)$ is divisible on $G(x)$ if and only if $E_1(x) = G(x)$.

The probability of such coincidence is equal to $2^{-(r-1)}$.

Thus, a probability that such distortion will not be detected is equal to $2^{-(r-1)}$.

(c) $k > r + 1$.

It can proved that in this case a probability that such distortion will not be detected is less that $< 2^{-r}$.

4. If

   - an odd number of bits is distorted, i.e. $E(x)$ has an odd number of monomials, and
   - $G(x) = (x + 1) \cdot G_1(x)$

   then such a distortion can be detected, because if for some polynomial $Q(x)$

   $$E(x) = G(x) \cdot Q(x)$$

   then, in particular

   $$E(1) = G(1) \cdot Q(1) \tag{9.11}$$

   that is wrong, since

   - the left side of (9.11) is equal to 1, and
   - the right side of (9.11) is equal to 0.

In the standard IEEE 802 the following generator polynomial $G(x)$ is used:

$$\begin{aligned} G(x) = \ & x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + \\ & + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1 \end{aligned}$$

This polynomial can detect a distortion, in which

- a size of a packet of errors is no more than 32, or
- it is distorted an odd number of bits.

9.3 Protocols of one-way transmission

9.3.1 A simplest protocol of one-way transmission

The protocol consists of the following agents:

- the **sender**
- the **timer** which is used by the sender
- the **receiver**
- the **channel**

The purpose of the protocol is a delivery of frames from the sender to the receiver via the channel. The channel is assumed to be unreliable, it can distort and lose transmitted frames.

The protocol works as follows.

1. The sender receives a message (which is called a **packet**) from an agent which is not included in the protocol. This agent is called a **sender's network agent (SNA)**.

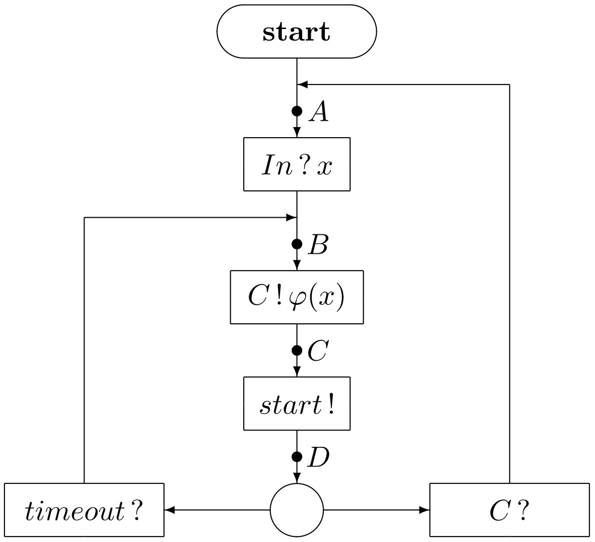
   The purpose of the sender is a cyclic execution of the following sequence of actions:

   - get a packet from the SNA
   - build a frame, which is obtained by an applying of a encoding function φ to the packet,
   - send this frame to the channel and switch-on the timer
   - if it comes the signal *timeout* from the timer, which means that
     - the waiting time of a confirmation of the sent frame has ended, and
     - apparently this frame is not reached by the receiver

     then send the frame again
   - if it comes the confirmation signal from the receiver, then
     - this means that the current frame is successfully accepted by the receiver, and
     - the sender can
       - get the next packet from the SNA,

- ∗ build a frame from this packet,
- ∗ etc.

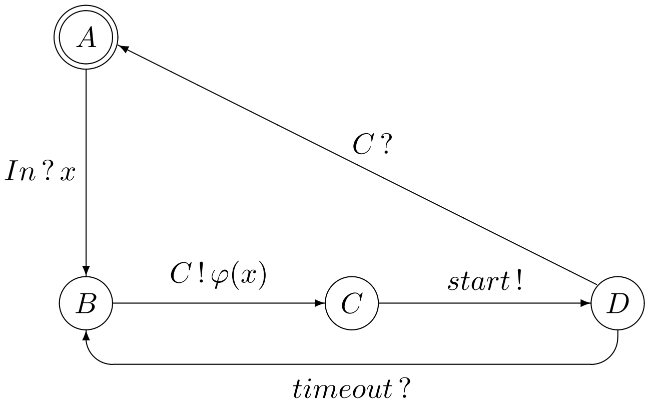
A flowchart representing this behavior has the following form:



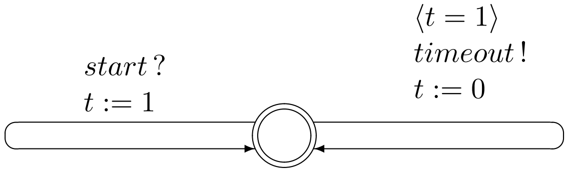
Operators belonging to this flowchart have the following meanings.

- $In\,?\,x$ is a receiving a packet from the SNA, and record this packet to the variable x
- $C\,!\,\varphi(x)$ is a sending the frame $\varphi(x)$ to the channel
- $start\,!$ is a switching-on of the timer
- $timeout\,?$ is a receiving of a signal "timeout" from the timer
- $C\,?$ is a receiving a confirmation signal from the channel.

The process represented by this flowchart, is denoted by *Sender* and has the following form:



The behavior of the timer is represented by the process *Timer* having the form



(9.12)

An initial condition of *Timer* is $t = 0$.

In this model we do not detail a magnitude of an interval between

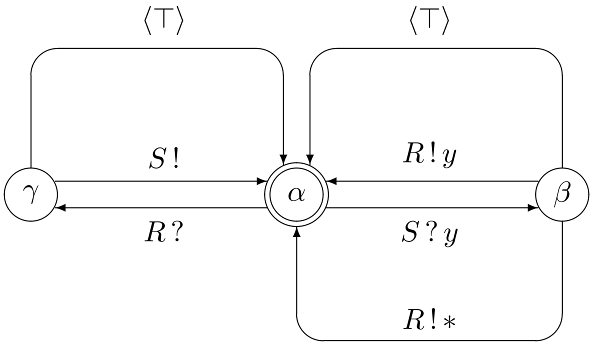
- a switching-on of the timer (the action *start* ?), and
- a switching-off of the timer (the action *timeout* !).

2. A **channel** at each time can contain no more than one frame or signal.

   It can execute the following actions:

   - receiving a frame from the sender, and
     - sending this frame to the receiver, or
     - sending a distorted frame to the receiver, or
     - loss of the frame
   - receivng a confirmation signal from the receiver, and
     - sending this signal to the sender, or
     - loss of the signal.

The behavior of the channel is described by the following process:



(9.13)

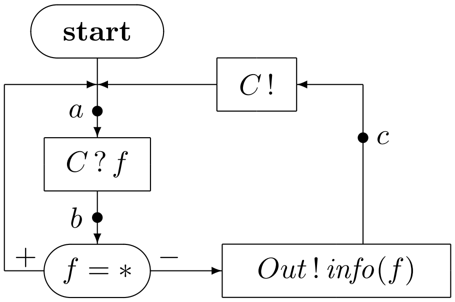
In this process, we use the following abstraction: the symbol '$*$' means a "distorted frame". We do not specify exactly, how frames can be distorted in the channel.

Each frame which has been received by the channel

- either is transferred from the channel to the receiver
- or is transformed to the abstract value '$*$', and this value is transferred from the channel to receiver
- or disappears, which is expressed by the transition of the process (9.13) with the label $\langle\top\rangle$

3. The **receiver** executes the following actions:
   - receiving a frame from the channel
   - checking of a distortion of the frame
   - if the frame is not distorted, then
     - extracting a packet from the frame
     - sending this packet to a process called a **receiver's network agent (RNA)**
       (this process is not included in the protocol)
     - sending a confirmation signal to the sender through the channel
   - if the frame is distorted, then the receiver ignores it (assuming that the sender will be tired to wait a confirmation signal, and will send the frame again)

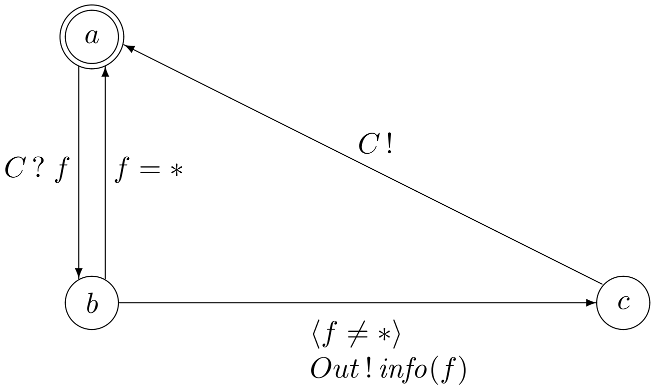
A flowchart representing the above behavior has the following form:



Operators belonging to this flowchart have the following meanings.

- $C\,?\,f$ is a receiving of a frame from the channel, and a record it to the variable f
- $(f = *)$ is a checking of a distortion of the frame f
- $Out\,!\,info(f)$ is a sending of the packet $info(f)$, extracted from the frame f, to the RNA
- $C\,!$ is a sending of the confirmation signal

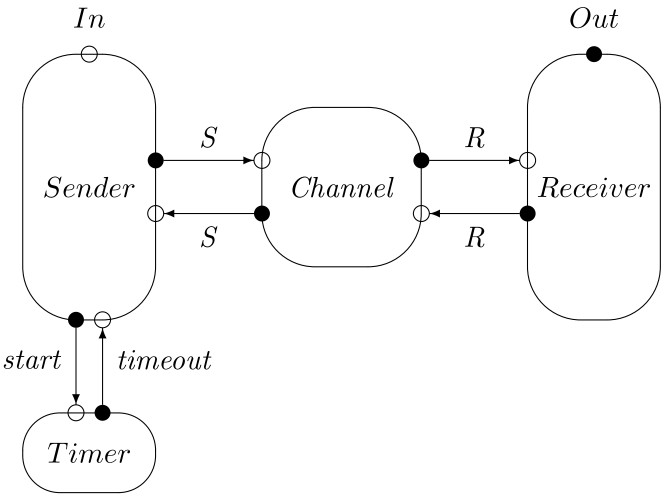
The process represented by this flowchart, is denoted as *Receiver* and has the following form:



The process *Protocol*, corresponding to the whole system, is defined as a parallel composition (with restriction and renaming) of the above processes:

$$Protocol \overset{\text{def}}{=} \left(\begin{array}{l} Sender\,[S/C]\,| \\ Timer\,| \\ Channel\,| \\ Receiver\,[R/C] \end{array} \right) \setminus \{S, R, start, timeout\} \qquad (9.14)$$

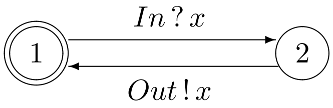
A flow graph of the process *Protocol* has the form



(9.15)

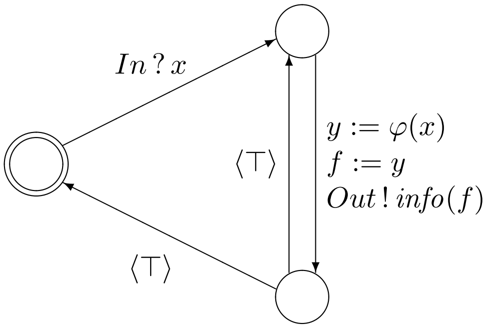
In order to be able to analyze the correctness of this protocol is necessary to determine a specification which he must meet.

If we want to specify only properties of external actions executed by the protocol (i.e., actions of the form $In\,?\,v$ and $Out\,!\,v$), then the specification can be as follows: the behavior of this protocol coincides with the behavior of the buffer of the size 1, i.e. the process *Protocol* is observationally equivalent to the process *Buf*, which has the form

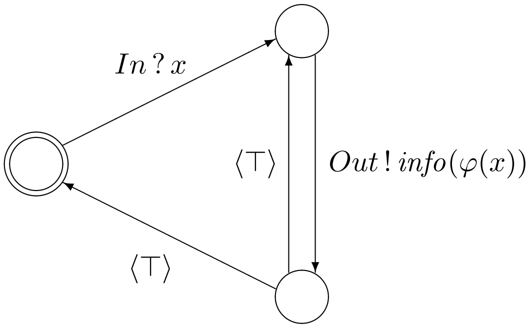


(9.16)

After a reduction of the graph representation of the process *Protocol* we get the diagram



which is observationally equivalent to the diagram

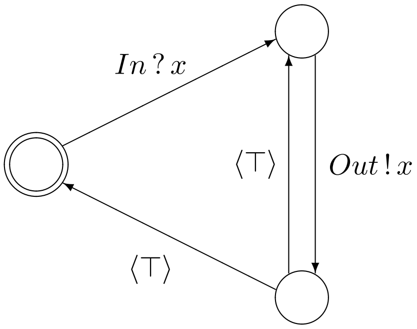


(9.17)

We assume that the function *info* of extracting of packets from frames is inverse to φ, i.e. for each packet x

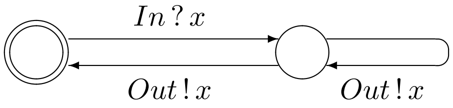
$$info(\varphi(x)) = x$$

therefore the diagram (9.17) can be redrawn as follows:



(9.18)

The process (9.18) can be reduced, resulting in the process



(9.19)

After a comparing of the processes (9.19) and (9.16) we conclude that these processes can not be equivalent in any acceptable way. For example,

- the process (9.16) after receiving the packet x can only
  - send this packet to the RNA, and
  - move to the state of waiting of another packet
- while the process (9.19) after receiving the packet x can send this packet to the RNA several times.

Such retransmission can occur, for example, in the following version of an execution of the protocol.

- First frame which is sent by the sender, reaches the receiver successfully.
- The receiver
  - sends the packet, extracted from this frame, to the RNA, and
  - sends a confirmation to the sender through the channel.
- This confirmation is lost in the channel.
- The sender does not received a confirmation, and sends this frame again, and this frame again goes well.
- The receiver perceives this frame as a new one. He
  - sends the packet, extracted from this frame, to the RNA, and
  - sends the confirmation signal to the sender through the channel.
- This confirmation again is lost in the channel.
- etc.

This situation may arise because in this protocol there is no a mechanism through which the receiver can distinguish:

- is a received frame a new one, or
- this frame was transmitted before.

In section 9.3.2 we consider a protocol which has such mechanism. For this protocol it is possible to prove formally its compliance with the specification (9.16).

9.3.2 One-way alternating bit protocol

The protocol described in this section is called the **one-way alternating bit protocol**, or, in an abbreviated notation, **ABP**.

The protocol ABP is designed to solve the same problem as the protocol in section 9.3.1: delivery of frames from the sender to the receiver via an unreliable channel (which can distort and lose transmitted frames).

The protocol ABP

- consists of the same agents as the protocol in section 9.3.1 (namely: the sender, the timer, the receiver, and the channel), and
- has the same flow graph.

A mechanism by which the receiver can distinguish new frames from retransmitted ones, is implemented in this protocol as follows: among the variables of the sender and the receiver there are boolean variables s and r, respectively, values which have the following meanings:

- a value of s is equal to a parity of an index of a current frame, which is trying to be sent by the sender, and
- a value of r is equal to a parity of an index of a frame, which is expected by the receiver.

At the initial time values of s and r are equal to 0 (the first frame has an index 0).

As in the protocol in section 9.3.1, the abstract value "$*$" is used in this protocol, this value denotes a distorted frame.

The protocol works as follows.

1. The **sender** gets a packet from the SNA, and
    - records this packet to the variable x,
    - builds the frame, which is obtained by an applying of a coding function φ to the pair (x, s),

- sends the frame to the channel,
- starts the timer, and then
- expects a confirmation of the frame which has been sent.

If

- the sender gets from the times the signal *timeout*, and
- he does not received yet an acknowledgment from the receiver

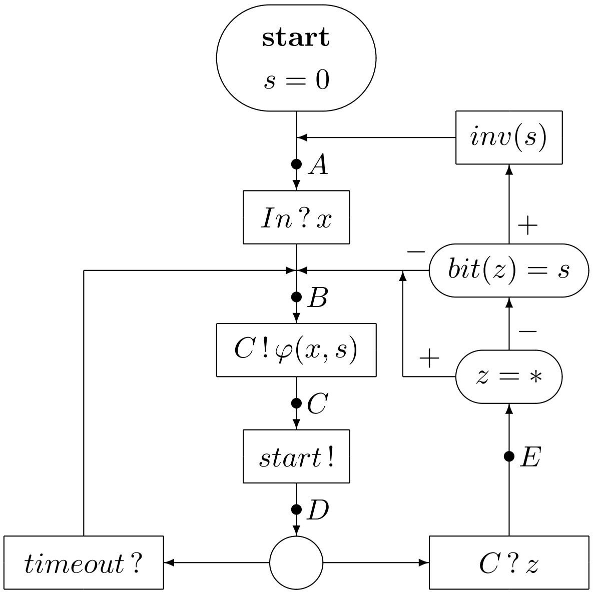
then the sender retransmits this frame.

If the sender receives from the channel an undistorted frame, which contains a boolean value, then the sender analyzes this value: if it coincides with the current value of s, then the sender

- inverts the value of the variable s (using the function $Inv(x) = 1 - x$), and
- starts a new cycle of his work.

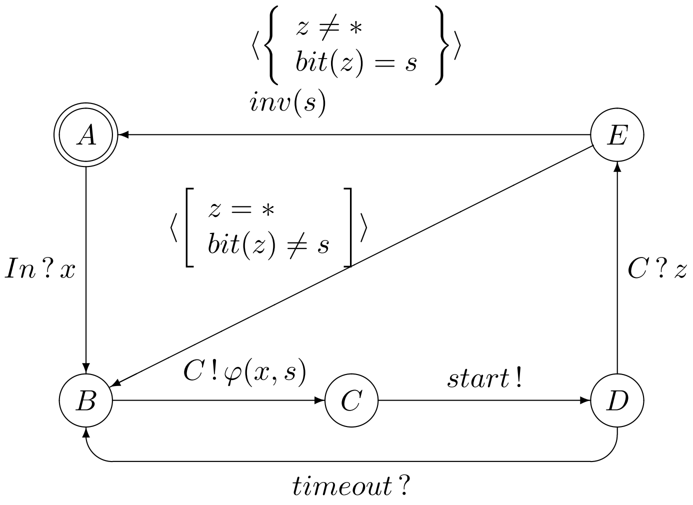
Otherwise, he sends the frame again.

The flowchart representing this behavior has the following form:



The process, which corresponds to this flowchart, is denoted by *Sender*, and has the following form:

$Init = (s = 0)$.

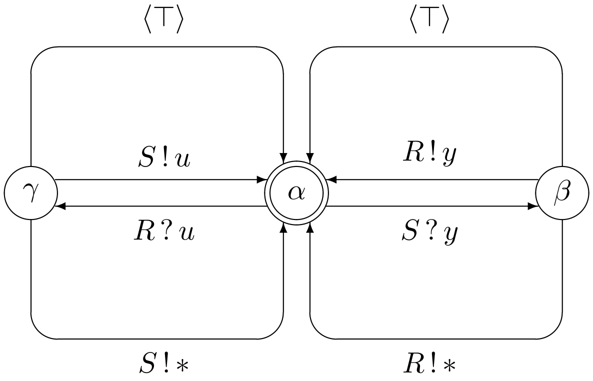

$\langle \left\{ \begin{array}{l} z \neq * \\ bit(z) = s \end{array} \right\} \rangle$
$inv(s)$
A
E
$\langle \left[\begin{array}{l} z = * \\ bit(z) \neq s \end{array} \right] \rangle$
$In\,?\,x$
$C\,?\,z$
$C\,!\,\varphi(x,s)$
$start\,!$
B
C
D
$timeout\,?$

2. The **channel** can contain no more than one frame.

   It can execute the following actions:

   - receive a frame from the sender, and
     - either send this frame to the receiver,
     - or send a distorted frame to the receiver,
     - or lose the frame
   - receive a confirmation frame from the receiver, and
     - either send this frame to the sender,
     - or send the distorted frame to the sender,
     - or lose the frame.

   The behavior of the channel is represented by the following process:



(9.20)

3. The **receiver** upon receiving of a frame from the channel

   - checks whether the frame is distorted,
   - and if the frame is not distorted, then the receiver extracts from the frame a packet and a boolean value using functions *info* and *bit*, with the following properties:

   $$\mathit{info}(\varphi(x,b)) = x, \quad \mathit{bit}(\varphi(x,b)) = b$$

   The receiver checks whether the boolean value extracted from the frame coincides with the expected value, which is contained in the variable r, and

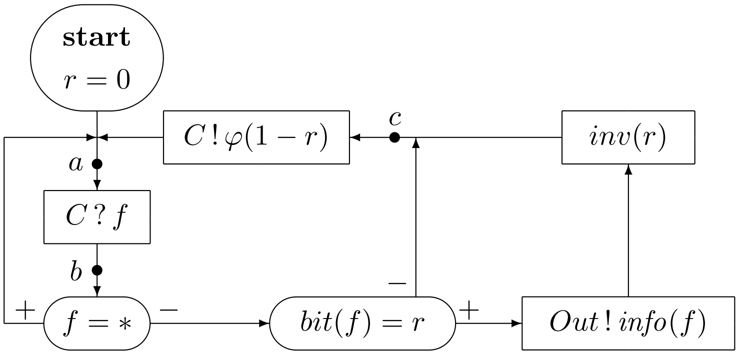
   (a) if the checking gave a positive result, then the receiver

      - transmits the packet extracted from this frame to the RNA
      - inverts the value of r, and
      - sends the confirmation frame to the sender through the channel.

   (b) if the checking gave a negative result, then the receiver sends a confirmation frame with an incorrect boolean value (which will cause the sender to send its current frame again).

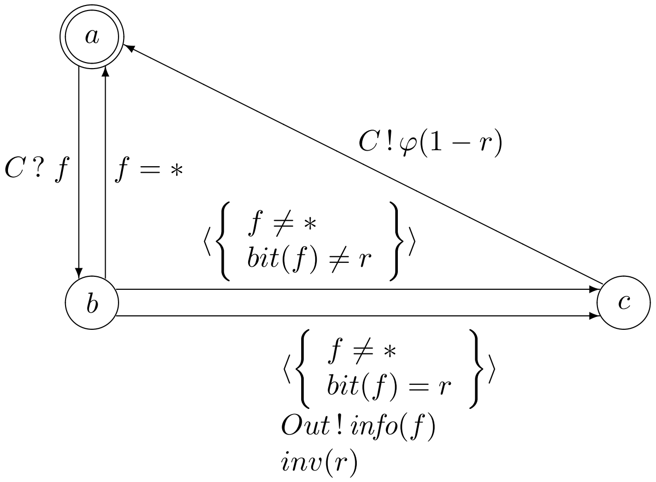
   If the frame is distorted, then the receiver ignores this frame (assuming that the sender will send this frame again on the reason of receiving of the signal *timeout* from the timer).

   The flowchart representing the above behavior has the following form:



The process represented by this flowchart, is denoted by *Receiver* and has the following form:

$Init = (r = 0)$

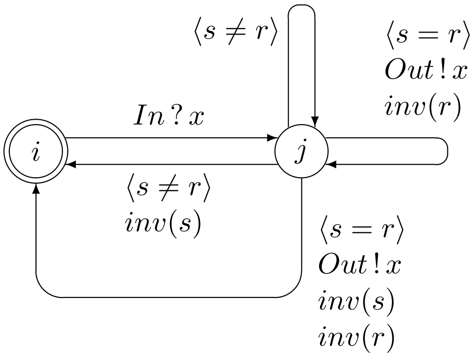


a
$C\,!\,\varphi(1-r)$
$C\,?\,f$
$f = *$
$\langle \left\{ \begin{array}{l} f \neq * \\ bit(f) \neq r \end{array} \right\} \rangle$
b
c
$\langle \left\{ \begin{array}{l} f \neq * \\ bit(f) = r \end{array} \right\} \rangle$
$Out\,!\,info(f)$
$inv(r)$

The process *Protocol*, which corresponds to the whole protocol ABP, is defined in the same manner as in section 9.3.1, by the expression (9.14). The flow graph of this process has the form (9.15).

The specification of the protocol ABP also has the same form as in section 9.3.1, i.e. is defined as the process (9.16).

The reduced process *Protocol* has the form



(9.21)

The statement

$$(9.16) \approx (9.21)$$

can be proven, for example, with use of theorem 34, defining the function μ of the form

$$\mu : \{1,2\} \times \{i,j\} \to Fm$$

as follows:

$$\begin{cases} \mu(1,i) \overset{\text{def}}{=} (s = r) \\ \mu(2,i) \overset{\text{def}}{=} \bot \\ \mu(1,j) \overset{\text{def}}{=} (s \neq r) \\ \mu(2,j) \overset{\text{def}}{=} (s = r) \end{cases}$$

9.4 Two-way alternating bit protocol

The above protocols implement a data transmission (i.e. a transmission of frames with packets from a NA) only in one direction.

In most situations, a data transmission must be implemented in both directions, i.e. each agent, which communicates with a channel, must act as a sender and as a receiver simultaneously.

Protocols which implement a data transmission in both directions, are called **duplex** protocols, or protocols of two-way transmission.

In protocols of two-way transmission a sending of confirmations can be combined with a sending of data frames (i.e. frames which contain packets from a NA): if an agent B has successfully received a data frame f from an agent A, then he may send a confirmation of receipt of the frame f not separately, but as part of his data frame.

In this section we consider the simplest correct protocol of two-way transmission.

This protocol

- is a generalization of ABP (which is considered in section 9.3.2), and
- is denoted as ABP-2.

ABP-2 also involves two agents, but behavior of each agent is described by the same process, which combines the processes *Sender* and *Receiver* from ABP.

Each frame f, which is sent by any of these agents, contains

- a packet x, and
- two boolean values: s and r, where
  - s has the same meaning as in ABP: this is a boolean value associated with the packet x, and
  - r is a boolean value associated with a packet in the last received undistorted frame.

To build a frame, the encoding function φ is used.

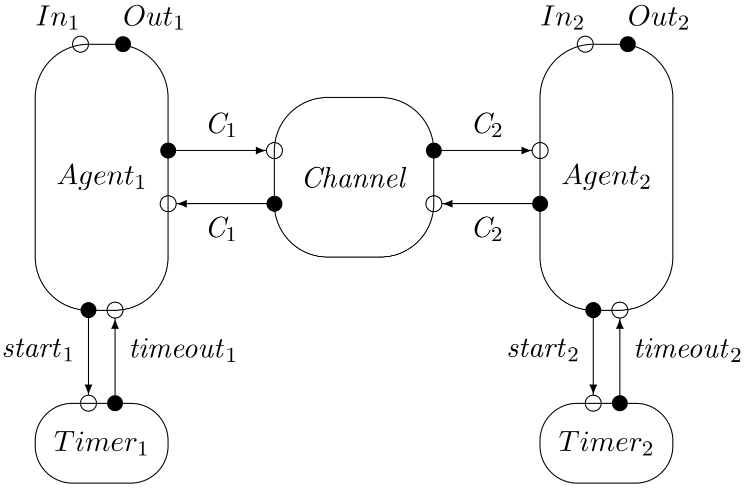
To extract a packet and boolean values s and r from a frame the functions *info*, *seq* and *ack* are used. These functions have the following properties:

$$\begin{array}{rcl} \mathit{info}(\varphi(x,s,r)) & = & x \\ \mathit{seq}(\varphi(x,s,r)) & = & s \\ \mathit{ack}(\varphi(x,s,r)) & = & r \end{array}$$

Also, agents use the inverting function *inv* to invert values of the boolean variables.

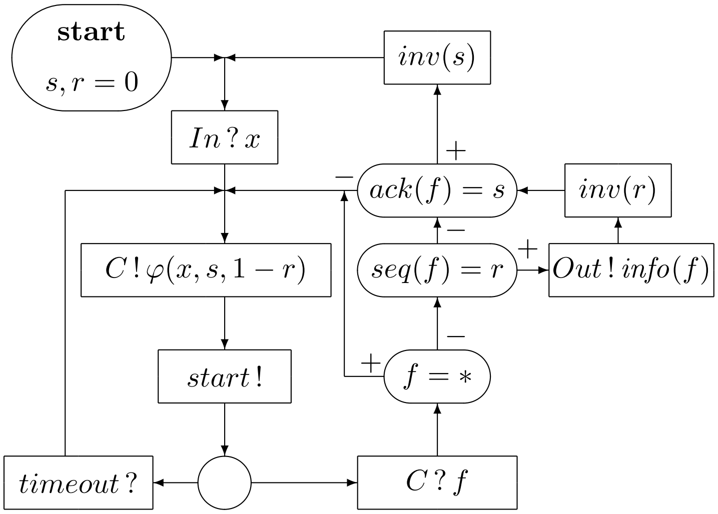
Each sending/receiving agent is associated with a timer. A behavior of the timer is described by the process *Timer*, which is represented by the diagram (9.12).

A flow graph of the protocol is as follows:



(9.22)

The process describing the behavior of sending/receiving agents, is represented by the following flowchart:



This flowchart shows that the agent sends a frame with its next packet only after receiving a confirmation of receiving of its current packet.

The flowchart describing the behavior of a specific agent (i.e. $Agent_1$ or $Agent_2$), is obtained from this flowchart by assigning the corresponding index (1 or 2) to the variables and names, included in this flowchart.

The behavior of the channel is described by the process

$$(9.20)\,[\,C_1/S,\, C_2/R\,]$$

The reader is requested

- to define the process *Spec*, which is a specification of this protocol, and
- to prove that this protocol meets the specification *Spec*.

9.5 Two-way sliding window protocols

ABP-2 is practically acceptable only when a duration of a frame transmission through the channel is negligible.

If a duration of a frame transmission through the channel is large, then it is better to use a **conveyor transmission**, in which the sender may send several frames in a row, without waiting their confirmation.

Below we consider two protocols of two-way conveyor transmission, called **sliding window protocols (SWPs)**.

These protocols are extensions of ABP-2. They

- also involve two sending/receiving agents, and behavior of each of these agent is described by the same process, combining functions of a sender and a receiver
- an analog of a boolean value associated with each frame is an element of the set
$$\mathbf{Z}_n = \{0, \ldots, n-1\}$$
where n is a fixed integer of the form 2^k.

An element of the set $\mathbf{Z}_n$, associated with a frame, is called a **number** of this frame.

9.5.1 The sliding window protocol using go back n

The first SWP is called **SWP using go back** n.

The process which describes a behavior of a sending/receiving agent of this protocol, has the array $x[n]$ among its variables. Components of this array may contain packets which are sent, but not yet confirmed.

A set of components of the array x, which contain such packets at the current time, is called a **window**.

Three variables of the process are related to the window:

- b (a lower bound of the window)
- s (an upper bound of the window), and
- w (a number of packets in the window).

Values of the variables b, s and w belong to the set $\mathbf{Z}_n$.

At the initial time

- the window is empty, and
- values of the variables b, s and w are equal to 0.

Adding a new packet to the window is performed by execution of the following actions:

- this packet is written in the component $x[s]$, and it is assumed that the number s is associated with this packet
- upper bound of the window s increases by 1 modulo n, i.e. new value of s is assumed to be
  - $s + 1$, if $s < n - 1$, and
  - 0, if $s = n - 1$,

  and
- w (the number of packets in the window) is increased by 1.

Removing a packet from the window is performed by execution of the following operations:

- b (the lower bound of the window) is increased by 1 modulo n, and
- w (the number of packets in the window) is decreased by 1

i.e. it is removed a packet whose number is equal to the lower bound of the window.

To simplify an understanding of the operations with a window you can use the following figurative analogy:

- the set of components of the array x can be regarded as a ring (i.e. after the component $x[n - 1]$ is the component $x[0]$)
- at each time the window is a connected subset of this ring,
- during the execution of the process this window is moved on this ring in the same direction.

If the window size reaches its maximum value $(n-1)$, then the agent does not accept new packets from his NA until the window size is not reduced.

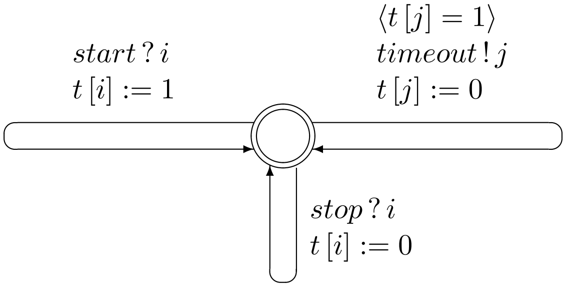
An ability to receive a new packet is defined by the boolean variable *enable*:

- if the value is 1, then the agent can receive new packets from his NA, and
- if 0, then he can not do receive new packets.

If the agent receives an acknowledgment of a packet whose number is equal to the lower bound of the window, then this packet is removed from the window.

Each component $x[i]$ of the array x is associated with a timer, which determines a duration of waiting of confirmation from another agent of a receiving of the packet contained in the component $x[i]$. The combination of these timers is considered as one process $Timers$, which has an array of $t\,[n]$ of boolean variables. This process is defined as follows:

$Init = (t = (0, \ldots, 0))$



(9.23)

The right arrow in this diagram is the abbreviation for a set of n transitions with labels

$$\begin{array}{lcl} \langle t\,[0] = 1\rangle & & \langle t\,[n-1] = 1\rangle \\ timeout\,!\,0 & \ldots & timeout\,!\,(n-1) \\ t\,[0] := 0 & & t\,[n-1] := 0 \end{array}$$

Note that in this process there is the operator $stop\,?\,i$, an execution of which prematurely terminates a corresponding timer.

The protocol has the following features

- If a sending/receiving agent has received a signal *timeout* from any timer, then the agent sends again all packets from his window.
- If an agent has received a confirmation of a packet, then all previous packets in the window are considered also as confirmed (even if their confirmations were not received).

Each frame f, which is sent by any of the sending/receiving agents of this protocol, contains

- a packet x,
- a number s, which is associated with the packet x
  (by definition, s is also associated with the frame f)
- a number r, which is a number associated with a last received undistorted frame.

To build a frame, the encoding function φ is used.

To extract the components from the frames, the functions *info*, *seq* and *ack*, are used. These functions have the following properties:

$$\begin{array}{rcl} info(\varphi(x,s,r)) & = & x \\ seq(\varphi(x,s,r)) & = & s \\ ack(\varphi(x,s,r)) & = & r \end{array}$$

The description of the process, representing the behavior of an agent of the protocol, we give in a flowchart form, which easily can be transformed to a flowchart.

In this description we use the following notations.

- The symbols $\underset{n}{+}$ and $\underset{n}{-}$ denote addition and subtraction modulo n.
- The symbol r denotes a variable with has values at $\mathbf{Z}_n$.

  A value of r is equal to a number of an expected frame.

  The agent sends to his NA a packet, extracted from such a frame f, whose number $seq(f)$ coincides with a value of the variable r.

  If a frame f is such that $seq(f) \neq r$, then

  - the packet $info(f)$ in this frame is ignored, and
  - it is taken into account only the component $ack(f)$.
- The notation *send* is the abbreviation of the following group of operators:

$$send = \left\{ \begin{array}{l} C\,!\,\varphi(x[s], s, r \underset{n}{-} 1) \\ start\,!\,s \\ s := s \underset{n}{+} 1 \end{array} \right\}$$

- The notation

$$between(a,b,c)$$

  is the abbreviation of the formula

$$\Big(\ a \leq b < c\ \Big) \vee \Big(\ c < a \leq b\ \Big) \vee \Big(\ b < c < a\ \Big) \qquad (9.24)$$

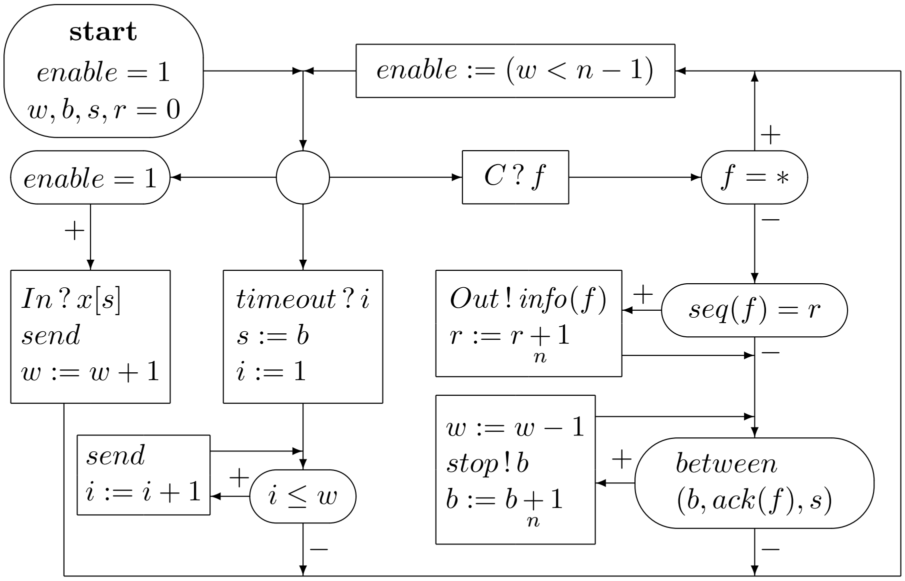
- The expression $(w < n - 1)$ in the operator

$$enable := (w < n - 1)$$

  has a value

  - 1, if the inequality $w < n - 1$ holds, and
  - 0, otherwise.

The process representing the behavior of a sending/receiveng agent of this protocos is the following:



The reader is requested

- to define a process "channel" for this protocol
  (channel contains an ordered sequence of frames, which may distort and disappear)
- to define a specification *Spec* of this protocol, and
- to prove that the protocol meets the specification *Spec*.

In conclusion, we note that this protocol is ineffective if a number of distortions in the frame transmission is large.

9.5.2 The sliding window protocol using selective repeat

The second SWP differs from the previous one in the following: an agent of this protocol has two windows.

1. First window has the same function, as a window of the first SWP (this window is called a **sending window**).

   The maximum size of the sending window is $m \overset{\text{def}}{=} n/2$, where n has the same status as described in section 9.5.1 (in particular, frame numbers are elements of $\mathbf{Z}_n$).

2. Second window (called a **receiving window**) is designed to accommodate packets received from another agent, which can not yet be transferred to a NA, because some packets with smaller numbers have not received yet.

   A size of the receiving window is $m = n/2$.

Each frame f, which is sent by a sending/receiving agent of this protocol, has 4 components:

1. k is a type of the frame,
   this component can have one of the following three values:

   - *data* (data frame)
   - *ack* (frame containing only a confirmation)
   - *nak* (frame containing a request for retransmission)
     ("nak" is an abbreviation of "negative acknowledgment")

2. x is a packet

3. s is a number associated with the frame

4. r is a number associated with the last received undistorted packet.

If a type of a frame is *ack* or *nak*, then second and third components of this frame are fictitious.

To build a frame, the encoding function φ is used.

To extract the components from the frames, the functions *kind*, *info*, *seq* and *ack* are used. These functions have the following properties:

$$\begin{array}{rcl}
\mathit{kind}(\varphi(k,x,s,r)) & = & k\\
\mathit{info}(\varphi(k,x,s,r)) & = & x\\
\mathit{seq}(\varphi(k,x,s,r)) & = & s\\
\mathit{ack}(\varphi(k,x,s,r)) & = & r
\end{array}$$

The process describing the behavior of a sending/receiveng agent has the following variables.

1. Arrays $x[m]$ and $y[m]$, designed to accommodate the sending window and the receiving window, respectively.

2. Variables $enable, b, s, w$, having

   - the same sets of values, and
   - the same meaning

   as they have in the previous protocol.

3. Variables r, u, values of which

   - belong to $\mathbf{Z}_n$, and
   - are equal to lower and upper bounds respectively of the receiving window.

   If these is a packet in the receiving window, a number of which is equal to the lower boundary receiving window (i.e. r), then the agent

   - transmits this packet to his NA, and
   - increases by 1 (modulo n) values of r and u.

4. Boolean array
   $$arrived[m]$$
   whose components have the following meaning: $arrived[i] = 1$ if and only if an i–th component of the receiving window contains a packet which is not yet transmitted to the NA.

5. Boolean variable no_nak, which is used with the following purpose.

   If the agent receives

   - a distorted frame, or
   - a frame, which has a number different from the lower boundary of the receiving window (i.e. r)

   then he sends to his colleague a request for retransmission of a frame whose number is r.

   This request is called a **Negative Acknowledgement (NAK)**.

The boolean variable *no_nak* is used to avoid multiple requests for a retransmission of the same frame: This variable is set to 1, if NAK for a frame with the number r has not yet been sent.

When a sending/receiveng agent gets an undistorted frame f of the type *data*, it performs the following actions.

- If the number $seq(f)$ falls into the receiving window, i.e. the following statement holds:
$$between(r, seq(f), u)$$
where the predicate symbol *between* has the same meaning as in the previous protocol (see (9.24)), then the agent
  - extracts a packet from this frame, and
  - puts the packet in its receiving window.
- If the condition from the previous item does not satisfied (i.e. the number $seq(f)$ of the frame f does not fall into the receiving window) then
  - a packet in this frame is ignored, and
  - only the component $ack(f)$ of this frame is taken into account.

The following timers are used by the sending/receiving agent.

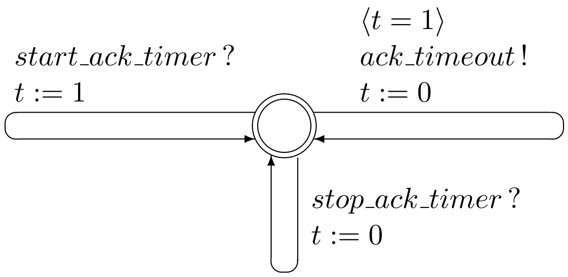
1. An array of m timers, whose behavior is described by the process $Timers$ (see (9.23), with the replacement of n on m).

   Each timer from this array is intended to alert the sending/receiving agent that

   - a waiting of a confirmation of a packet from the sending window with the corresponding number is over, and
   - it is necessary to send a frame with this packet again

2. Additional timer, whose behavior is described by the following process:

   $Init = (t = 0)$



This timer is used with the following purpose.

A sending by an agent of confirmations of frames received from another agent can be done as follows: the confirmation is sent

(a) as a part of a data frame, or

(b) as a special frame of the type *ack*.

When the agent should send a confirmation **conf**, he

- starts the auxiliary timer (i.e. executes the action *start_ack_timer* !),
- if the agent has received a new packet from his NA before a receiving of the signal *timeout* from the auxiliary timer, then the agent
  - builds a frame of the type *data*, with consists of
    - ∗ this packet, and
    - ∗ the confirmation **conf** as the component *ack*
  - sends this frame to the colleague
- if after an expiration of the auxiliary timer (i.e., after receiving the signal *ack_timeout*) the agent has not yet received a new packet from his NA, then he sends the confirmation **conf** by a separate frame of the type *ack*.

The description of the process, representing the behavior of an agent of the protocol, we give in a flowchart form, which easily can be transformed to a flowchart.

In this description we use the following notations and agreements.

1. If i is an integer, then the notation $i\%m$ denotes a remainder of the division of i on m.

2. If

   - *mass* is a name of an array of m components (i.e. x, y, *arrived*, etc.) and
   - i is an integer

   then the notation $mass[i]$ denotes the element $mass[i\%m]$.

3. A notation of the form *send*(*kind*, i) is the abbreviation of the following group of operators:

$$send(kind, i) = \left\{ \begin{array}{l} C\,!\,\varphi(kind, x[i], i, r \underset{n}{-} 1) \\ \textbf{if}\ \ (kind = nak)\ \ \textbf{then}\ \ no\_nak := 0 \\ \textbf{if}\ \ (kind = data)\ \ \textbf{then}\ \ start\,!\,(i\%m) \\ stop\_ack\_timer\,! \end{array} \right\}$$

4. The notation *between*(a, b, c) has the same meaning as in the previous protocol.

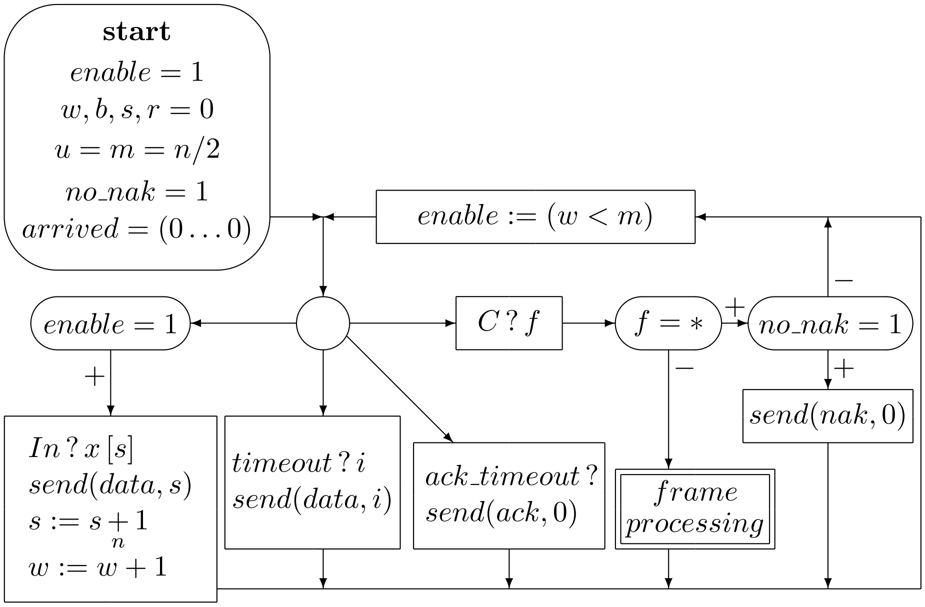
5. If any oval contains several formulas, then we assume that these formulas are connected by the conjunction ($\wedge$).

6. In order to save a space, some expressions of the form

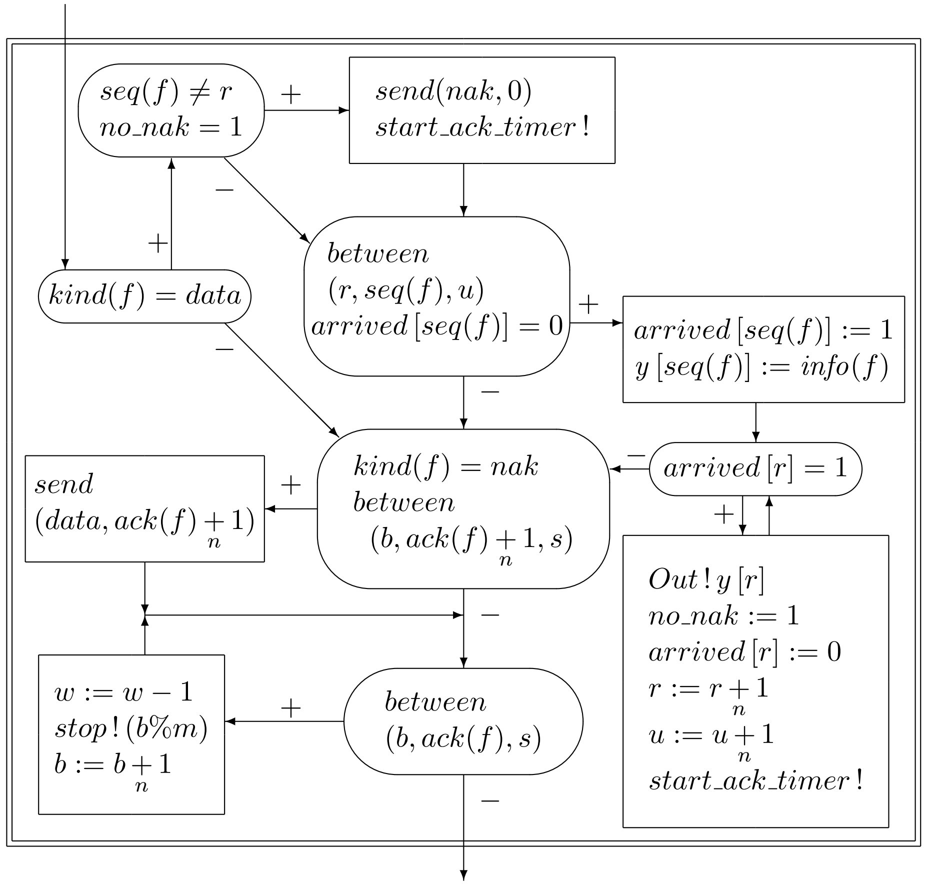
$$f(e_1, \ldots, e_n)$$

are written in two lines (f in the first line, and the list $(e_1, \ldots, e_n)$ in the second line)

The process which represents a behavior of an agent of this protocol, has the following form:



The fragment *frame processing* in this diagram has the following form.



The reader is requested

- to define a process “channel” for this protocol (channel contains an ordered sequence of frames, which may distort and disappear)
- to define a specification *Spec* of this protocol, and
- to prove that the protocol meets the specification *Spec*.

Chapter 10

History and overview of the current state of the art

Theory of processes combines several research areas, each of which reflects a certain approach to modeling and analysis of processes. Below we consider the largest of these directions.

10.1 Robin Milner

The largest contribution to the theory of processes was made by outstanding English mathematician and computer scientist **Robin Milner** (see [1] - [5]). He was born 13 January 1934 near Plymouth, in the family of military officer, and died 20 March 2010 in Cambridge.

Since 1995 Robin Milner worked as a professor of computer science at University of Cambridge (`http://www.cam.ac.uk`). From January 1996 to October 1999 Milner served as a head of Computer Lab at University of Cambridge.

In 1971-1973, Milner worked in the Laboratory of Artificial Intelligence at Stanford University. From 1973 to 1995 he worked at Computer Science Department of University of Edinburgh (Scotland), where in 1986 he founded the Laboratory for Foundation of Computer Science.

From 1971 until 1980, when he worked at Stanford and then in Edinburgh, he made a research in the area of automated reasoning. Together with colleagues he developed a Logic for Computable Functions (LCF), which

- is a generalization of D. Scott's approach to the concept of computability, and
- is designed for an automation of formal reasoning.

This work formed the basis for applied systems developed under the leadership of Milner.

In 1975-1990 Milner led the team which developed the Standard ML (ML is an abbreviation of "Meta-language"). ML is a widely used in industry and education Programming Language. A semantics of this language has been fully formalized. In the language Standard ML it was first implemented an algorithm for inference of polymorphic types. The main advantages of Standard ML are

- an opportunity of operating with logic proofs, and
- means of an automation of a construction of logical proofs.

Around 1980 Milner developed his main scientific contribution - a Calculus of Communicating Systems (CCS, see section 10.2). CCS is one of the first algebraic calculi for an analysis of parallel processes.

In late 1980, together with two colleagues he developed a π-calculus, which is the main model of the behavior of mobile interactive systems.

In 1988, Milner was elected a Fellow of the Royal Society. In 1991 he was awarded by A. M. Turing Award – the highest award in the area of Computer Science.

The main objective of his scientific activity Milner himself defined as a building of a theory unifying the concept of a computation with the concept of an interaction.

10.2 A Calculus of Communicating Systems (CCS)

A Calculus of Communicating Systems (CCS) was first published in 1980 in Milner's book [89]. The standard textbook on CCS is [92].

In [89] presented the results of Milner's research during the period from 1973 to 1980.

The main Milner's works on models of parallel processes made at this period:

- papers [84], [85], where Milner explores the denotational semantics of parallel processes
- papers [83], [88], where in particular, it is introduced the concept of a flow graph with synchronized ports
- [86], [87], in these papers the modern CCS was appeared.

The model of interaction of parallel processes, which is used in CCS,

- is based on the concept of a message passing, and
- was taken from the work of Hoare [71].

In the paper [66]

- a strong and observational equivalences are studied, and
- it is introduced the logic of Hennessy-Milner.

The concepts introduced in CCS were developed in other approaches, the most important of them are

- the π-calculus ([53], [97], [94]), and
- structural operational semantics (SOS), this approach was established by G. Plotkin, and published in the paper [104].

More detail historical information about CCS can be found in [105].

10.3 Theory of communicating sequential processes (CSP)

Theory of Communicating Sequential Processes (CSP) was developed by English mathematician and computer scientist Tony Hoare (C.A.R. Hoare) (b. 1934). This theory arose in 1976 and was published in [71]. A more complete summary of CSP is contained in the book [73].

In the CSP it is investigated a model of communication of parallel processes, based on the concept of a message passing. It is considered a synchronous interaction between processes.

One of the key concepts of CSP is the concept of a guarded command, which is borrowed from Dijkstra's work [52].

In [72] it is considered a model of CSP, based on the theory of traces. The main disadvantage of this model is the lack of methods for studying of the deadlock property. This disadvantage is eliminated in the other model CSP (failure model), introduced in [46].

10.4 Algebra of communicating processes (ACP)

Jan Bergstra and Jan Willem Klop in 1982 introduced in [37] the term “process algebra” for the first order theory with equality, in which the object variables take values in the set of processes. Then they have developed approaches led to the creation of a new direction in the theory of processes - the Algebra of Communicating Processes (ACP), which is contained in the papers [39], [40], [34].

The main object of study in the ACP logical theories, function symbols of which correspond to operations on processes ($a.$, $+$, etc).

In [19] a comparative analysis of different points of view on the concept of a process algebra can be found.

10.5 Process Algebras

The term **process algebra (PA)**, introduced by Bergstra and Klop, is used now in two meanings.

- In the first meaning, the term refers to an arbitrary theory of first order with equality, the domain of interpretation of which is a set of processes.
- In the second meaning, the term denotes a large class of directions, each of which is an algebraic theory, which describes properties of processes.

  In this meaning, the term is used, for example, in the title of the book "Handbook of Process Algebra" [42].

Below we list the most important directions related to PA in both meanings of this term.

1. Handbook of PA [42].
2. Summary of the main results in the PA: [19].
3. Historical overviews: [27], [28], [15].
4. Different approaches related to the concept of an equivalence of processes: [101], [59], [57], [58], [56].
5. PA with the semantics of partial orders: [44].
6. PA with recursion: [91], [47].
7. SOS-model for the PA: [21], [38].
8. Algebraic methods of verification: [63].
9. PA with data (actions and processes are parameterized by elements of the data set)

- PA with data μ-CRL
- [62] (there is a software tool for verification on the base of presented approach).
- PSF [79] (there is a software tool).
- Language of formal specifications LOTOS [45].

10. PA with time (actions and processes are parameterized by times)

    - PA with time based on CCS: [114], [99].
    - PA with time based on CSP: [107]. Textbook: [109].
    - PA with time on the base of ACP: [29].
    - Integration of discrete and dense time relative and absolute time: [32].
    - Theory ATP: [100].
    - Account of time in a bisimulation: [33].
    - Software tool UPPAAL [74]
    - Software tool KRONOS [116] (timed automata).
    - μ-CRL with time: [111] (equational reasonings).

11. Probabilistic PA (actions and processes are parameterized by probabilities).

    These PAs are intended for combined systems research, which simultaneously produced verification, and performance analysis.

    - Pioneering work: [64].
    - Probabilistic PA, based on CSP: [76]
    - Probabilistic PA, based on CCS: [69]
    - Probabilistic PA, based on ACP: [31].
    - PA TIPP (and the associated software tool): [60].
    - PA EMPA: [43].
    - In the works [21] and [23] it is considered simultaneous use of conventional and probabilistic alternative composition of processes.
    - In the paper [51] the concept of an approximation of probabilistic processes is considered.

12. Software related to PAs

- Concurrency Workbench [98] (PAs similar to CCS).
- CWB-NC [117].
- CADP [54].
- CSP: FDR `http://www.fsel.com/`

10.6 Mobile Processes

Mobile processes describe a behavior of distributed systems, which may change

- a configuration of connections between their components, and
- structure of these components

during their functioning.

Main sources:

1. the π-calculus (Milner and others):
   - the old handbook: [53],
   - standard reference: [97],
   - textbooks: [94], [8], [10], [9]
   - page on Wikipedia: [14]
   - implementation of the π-calculus on a distributed computer system: [115].
   - application of the π-calculus to modeling and verification of security protocols: [12].
2. The ambient calculus: [48].
3. Action calculus (Milner): [93]
4. Bigraphs: [95], [96].
5. Review of the literature on mobile processes: [11].
6. Software tool: Mobility Workbench [112].
7. Site `www.cs.auc.dk/mobility`

Other sources:

- R. Milner's lecture "Computing in Space" [6], which he gave at the opening of the building named by B.Gates built for the Computer Lab of Cambridge University, May 1, 2002.

  In the lecture the concepts of an "ambient" and a "bigraph" are introduced.

- R. Milner's lecture "Turing, Computing and Communication" [7].

10.7 Hybrid Systems

A hybrid system is a system, in which

- values of some variables change discretely, and
- values of other variables are changed continuously.

Modeling of a behavior of such systems is produced by using of differential and algebraic equations.

The main approaches:

- Hybrid Process Algebras: [41], [49], [113].
- Hybrid automata: [22] [77].

For simulation and verification of hybrid systems it is developed a software tool HyTech [68].

10.8 Other mathematical theories and software tools, associated with a modeling and an analysis of processes

1. Page in Wikipedia on the theory of processes [13].
2. Theory of Petri nets [103].
3. Theory of partial orders [80].
4. Temporal logic and model checking [106], [118].
5. Theory of traces [108].
6. Calculus of invariants [24].

7. Metric approach (which studies the concept of a distance between processes): [35], [36].

8. SCCS [90].

9. CIRCAL [82].

10. MEIJE [25].

11. Process algebra of Hennessy [65].

12. Models of processes with infinite sets of states: [119], [120], [121], [122].

13. Synchronous interacting machines: [123], [124], [125].

14. Asynchronous interacting extended machines: [126] - [130].

15. Formal languages SDL [131], Estelle [132], LOTOS [133].

16. The formalism of Statecharts, introduced by D. Harel [134], [135] and used in the design of the language UML.

17. A model of communicating extended timed automata CETA [136] - [140].

18. A Calculus of Broadcasting Systems [17], [18].

10.9 Business Processes

1. BPEL (Business process execution language) [141].

2. BPML (Business Process Modeling Language) [16], [142].

3. The article "Does Better Math Lead to Better Business Processes?" [143].

4. The web-page "π-calculus and Business Process Management" [144].

5. The paper "Workflow is just a π-process", Howard Smith and Peter Fingar, October 2003 [145].

6. "Third wave" in the modeling of business processes: [146], [147].

7. The paper "Composition of executable business process models by combining business rules and process flows" [148].

8. Web services choreography description language [149].

Bibliography

[1] Web-page of R. Milner

`http://www.cl.cam.ac.uk/~rm135/`

[2] Web-page of R. Milner in the Wikipedia.

`http://en.wikipedia.org/wiki/Robin_Milner`

[3] An interview of R. Milner.

`http://www.dcs.qmul.ac.uk/~martinb/interviews/milner/`

[4] `http://www.fairdene.com/picalculus/robinmilner.html`

[5] `http://www.cs.unibo.it/gorrieri/icalp97/Lauree_milner.html`

[6] **R. Milner:** Computing in Space. *May, 2002.*

`http://www.fairdene.com/picalculus/milner-computing-in-space.pdf`

[7] **R. Milner:** Turing, Computing and Communication. *King's College, October 1997.*

`http://www.fairdene.com/picalculus/milner-infomatics.pdf`

[8] The π-calculus, a tutorial.

`http://www.fairdene.com/picalculus/pi-c-tutorial.pdf`

[9] **J. Parrow:** An introduction to the π-calculus. *[42], p. 479-543.*

[10] **D. Sangiorgi and D. Walker:** The π-calculus: A Theory of Mobile Processes. *ISBN 0521781779.*

`http://us.cambridge.org/titles/`
`catalogue.asp?isbn=0521781779`

[11] **S. Dal Zilio:** Mobile Processes: a Commented Bibliography.

`http://www.fairdene.com/picalculus/`
`mobile-processes-bibliography.pdf`

[12] **M. Abadi and A. D. Gordon:** A calculus for cryptographic protocols: The Spi calculus. *Journal of Information and Computation, 143:1-70, 1999.*

[13] The site "Process calculus".

`http://en.wikipedia.org/wiki/Process_calculus`

[14] The site about the π-calculus.

`http://en.wikipedia.org/wiki/Pi-calculus`

[15] **J.C.M. Baeten:** A brief history of process algebra, *Rapport CSR 04-02, Vakgroep Informatica, Technische Universiteit Eindhoven, 2004*

`http://www.win.tue.nl/fm/0402history.pdf`

[16] Business Process Modeling Language

`http://en.wikipedia.org/wiki/BPML`

[17] `http://en.wikipedia.org/wiki/`
`Calculus_of_Broadcasting_Systems`

[18] **K. V. S. Prasad:** A Calculus of Broadcasting Systems, *Science of Computer Programming, 25, 1995.*

[19] **L. Aceto:** Some of my favorite results in classic process algebra. *Technical Report NS-03-2, BRICS, 2003.*

[20] **L. Aceto, Z.T. Ésik, W.J. Fokkink, and A. Ingólfsdóttir (editors):** Process Algebra: Open Problems and Future Directions. *BRICS Notes Series NS-03-3, 2003.*

[21] **L. Aceto, W.J. Fokkink, and C. Verhoef:** Structural operational semantics. *In [42], pp. 197–292, 2001.*

[22] **R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine:** The algorithmic analysis of hybrid systems. *Theoretical Computer Science, 138:3–34, 1995.*

[23] **S. Andova:** Probabilistic Process Algebra. *PhD thesis, Technische Universiteit Eindhoven, 2002.*

[24] **K.R. Apt, N. Francez, and W.P. de Roever:** A proof system for communicating sequential processes. *TOPLAS, 2:359–385, 1980.*

[25] **D. Austry and G. Boudol:** Algébre de processus et synchronisation. *Theoretical Computer Science, 30:91–131, 1984.*

[26] **J.C.M. Baeten:** The total order assumption. *In S. Purushothaman and A. Zwarico, editors, Proceedings First North American Process Algebra Workshop, Workshops in Computing, pages 231–240. Springer Verlag, 1993.*

[27] **J.C.M. Baeten:** Over 30 years of process algebra: Past, present and future. *In L. Aceto, Z. T. Ésik, W.J. Fokkink, and A. Ingólfsdóttir, editors, Process Algebra: Open Problems and Future Directions, volume NS-03-3 of BRICS Notes Series, pages 7–12, 2003.*

[28] `http://www.win.tue.nl/fm/pubbaeten.html`

[29] **J.C.M. Baeten and J.A. Bergstra:** Real time process algebra. *Formal Aspects of Computing, 3(2):142–188, 1991.*

[30] **J.C.M. Baeten, J.A. Bergstra, C.A.R. Hoare, R. Milner, J. Parrow, and R. de Simone:** The variety of process algebra. *Deliverable ESPRIT Basic Research Action 3006, CONCUR, 1991.*

[31] **J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka:** Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation, 121(2):234–255, 1995.*

[32] **J.C.M. Baeten and C.A. Middelburg:** Process Algebra with Timing. *EATCS Monographs. Springer Verlag, 2002.*

[33] **J.C.M. Baeten, C.A. Middelburg, and M.A. Reniers:** A new equivalence for processes with timing. *Technical Report CSR 02-10, Eindhoven University of Technology, Computer Science Department, 2002.*

[34] **J.C.M. Baeten and W.P. Weijland:** Process Algebra. *Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.*

[35] **J.W. de Bakker and J.I. Zucker:** Denotational semantics of concurrency. *In Proceedings 14th Symposium on Theory of Computing, pages 153–158. ACM, 1982.*

[36] **J.W. de Bakker and J.I. Zucker:** Processes and the denotational semantics of concurrency. *Information and Control, 54:70–120, 1982.*

[37] **J.A. Bergstra and J.W. Klop:** Fixed point semantics in process algebra. *Technical Report IW 208, Mathematical Centre, Amsterdam, 1982.*

[38] **J.A. Bergstra and J.W. Klop:** The algebra of recursively defined processes and the algebra of regular processes. *In J. Paredaens, editor, Proceedings 11th ICALP, number 172 in LNCS, pages 82–95. Springer Verlag, 1984.*

[39] **J.A. Bergstra and J.W. Klop:** Process algebra for synchronous communication. *Information and Control, 60(1/3):109–137, 1984.*

[40] **J.A. Bergstra and J.W. Klop:** A convergence theorem in process algebra. *In J.W. de Bakker and J.J.M.M. Rutten, editors, Ten Years of Concurrency Semantics, pages 164–195. World Scientific, 1992.*

[41] **J.A. Bergstra and C.A. Middelburg:** Process algebra semantics for hybrid systems. *Technical Report CS-R 03/06, Technische Universiteit Eindhoven, Dept. of Comp. Sci., 2003.*

[42] **J.A. Bergstra, A. Ponse, and S.A. Smolka, editors:** Handbook of Process Algebra. *North-Holland, Amsterdam, 2001.*

[43] **M. Bernardo and R. Gorrieri:** A tutorial on EMPA: A theory of concurrent processes with non-determinism, priorities, probabilities and time. *Theoretical Computer Science, 202:1–54, 1998.*

[44] **E. Best, R. Devillers, and M. Koutny:** A unified model for nets and process algebras. *In [42], pp. 945–1045, 2001.*

[45] **E. Brinksma (editor):** Information Processing Systems, Open Systems Interconnection, LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, *volume IS-8807 of International Standard. ISO, Geneva, 1989.*

[46] **S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe:** A theory of communicating sequential processes. *Journal of the ACM, 31(3):560–599, 1984.*

[47] **O. Burkart, D. Caucal, F. Moller, and B. Steffen:** Verification on infinite structures. *In [42], pp. 545–623, 2001.*

[48] **L. Cardelli and A.D. Gordon:** Mobile ambients. *Theoretical Computer Science, 240:177–213, 2000.*

[49] **P.J.L. Cuijpers and M.A. Reniers:** Hybrid process algebra. *Technical Report CS-R 03/07, Technische Universiteit Eindhoven, Dept. of Comp. Sci., 2003.*

[50] **P.R. D'Argenio:** Algebras and Automata for Timed and Stochastic Systems. *PhD thesis, University of Twente, 1999.*

[51] **J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden:** Metrics for labeled Markov systems. *In J.C.M. Baeten and S. Mauw, editors, Proceedings CONCUR'99, number 1664 in Lecture Notes in Computer Science, pages 258–273. Springer Verlag, 1999.*

[52] **E.W. Dijkstra:** Guarded commands, nondeterminacy, and formal derivation of programs. *Communications of the ACM, 18(8):453– 457, 1975.*

[53] **U. Engberg and M. Nielsen:** A calculus of communicating systems with label passing. *Technical Report DAIMI PB-208, Aarhus University, 1986.*

[54] **J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu:** CADP (CAESAR/ALDEBARAN development package): A protocol validation and verification toolbox. *In R. Alur and T.A. Henzinger, editors, Proceedings CAV '96, number 1102 in Lecture Notes in Computer Science, pages 437–440. Springer Verlag, 1996.*

[55] **R.W. Floyd:** Assigning meanings to programs. *In J.T. Schwartz, editor, Proceedings Symposium in Applied Mathematics, Mathematical Aspects of Computer Science, pages 19–32. AMS, 1967.*

[56] **R.J. van Glabbeek:** The linear time – branching time spectrum II; the semantics of sequential systems with silent moves. *In E. Best, editor, Proceedings CONCUR '93, number 715 in Lecture Notes in Computer Science, pages 66–81. Springer Verlag, 1993.*

[57] **R.J. van Glabbeek:** What is branching time semantics and why to use it? *In M. Nielsen, editor, The Concurrency Column, pages 190–198. Bulletin of the EATCS 53, 1994.*

[58] **R.J. van Glabbeek:** The linear time – branching time spectrum I. The semantics of concrete, sequential processes. *In [42], pp. 3–100, 2001.*

[59] **R.J. van Glabbeek and W.P. Weijland:** Branching time and abstraction in bisimulation semantics. *Journal of the ACM, 43:555–600, 1996.*

[60] **N. Götz, U. Herzog, and M. Rettelbach:** Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. *In L. Donatiello and R. Nelson, editors, Performance Evaluation of Computer and Communication Systems, number 729 in LNCS, pages 121–146. Springer, 1993.*

[61] **J.F. Groote:** Process Algebra and Structured Operational Semantics. *PhD thesis, University of Amsterdam, 1991.*

[62] **J.F. Groote and B. Lisser:** Computer assisted manipulation of algebraic process specifications. *Technical Report SEN-R0117, CWI, Amsterdam, 2001.*

[63] **J.F. Groote and M.A. Reniers:** Algebraic process verification. *In [42], pp. 1151–1208, 2001.*

[64] **H. Hansson:** Time and Probability in Formal Design of Distributed Systems. *PhD thesis, University of Uppsala, 1991.*

[65] **M. Hennessy:** Algebraic Theory of Processes. *MIT Press, 1988.*

[66] **M. Hennessy and R. Milner:** On observing nondeterminism and concurrency. *In J.W. de Bakker and J. van Leeuwen, editors, Proceedings 7th ICALP, number 85 in Lecture Notes in Computer Science, pages 299– 309. Springer Verlag, 1980.*

[67] **M. Hennessy and G.D. Plotkin:** Full abstraction for a simple parallel programming language. *In J. Becvar, editor, Proceedings MFCS, number 74 in LNCS, pages 108– 120. Springer Verlag, 1979.*

[68] **T.A. Henzinger, P. Ho, and H. Wong-Toi:** Hy-Tech: The next generation. *In Proceedings RTSS, pages 56–65. IEEE, 1995.*

[69] **J. Hillston:** A Compositional Approach to Performance Modelling. *PhD thesis, Cambridge University Press, 1996.*

[70] **C.A.R. Hoare:** An axiomatic basis for computer programming. *Communications of the ACM, 12:576–580, 1969.*

[71] **C.A.R. Hoare:** Communicating sequential processes. *Communications of the ACM, 21(8):666–677, 1978.*

[72] **C.A.R. Hoare:** A model for communicating sequential processes. *In R.M. McKeag and A.M. Macnaghten, editors, On the Construction of Programs, pages 229–254. Cambridge University Press, 1980.*

[73] **C.A.R. Hoare:** Communicating Sequential Processes. *Prentice Hall, 1985.*

[74] **K.G. Larsen, P. Pettersson, and Wang Yi:** Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer, 1, 1997.*

[75] **P. Linz:** An Introduction to Formal Languages and Automata. *Jones and Bartlett, 2001.*

[76] **G. Lowe:** Probabilities and Priorities in Timed CSP. *PhD thesis, University of Oxford, 1993.*

[77] **N. Lynch, R. Segala, F. Vaandrager, and H.B. Weinberg:** Hybrid I/O automata. *In T. Henzinger, R. Alur, and E. Sontag, editors, Hybrid Systems III, number 1066 in Lecture Notes in Computer Science. Springer Verlag, 1995.*

[78] **S. MacLane and G. Birkhoff:** Algebra. *MacMillan, 1967.*

[79] **S. Mauw:** PSF: a Process Specification Formalism. *PhD thesis, University of Amsterdam, 1991.*

`http://carol.science.uva.nl/~psf/`

[80] **A. Mazurkiewicz:** Concurrent program schemes and their interpretations. *Technical Report DAIMI PB-78, Aarhus University, 1977.*

[81] **J. McCarthy:** A basis for a mathematical theory of computation. *In P. Braffort and D. Hirshberg, editors, Computer Programming and Formal Systems, pages 33–70. North-Holland, Amsterdam, 1963.*

[82] **G.J. Milne:** CIRCAL: A calculus for circuit description. *Integration, 1:121–160, 1983.*

[83] **G.J. Milne and R. Milner:** Concurrent processes and their syntax. *Journal of the ACM, 26(2):302–321, 1979.*

[84] **R. Milner:** An approach to the semantics of parallel programs. *In Proceedings Convegno di informatica Teoretica, pages 285– 301, Pisa, 1973. Instituto di Elaborazione della Informazione.*

[85] **R. Milner:** Processes: A mathematical model of computing agents. *In H.E. Rose and J.C. Shepherdson, editors, Proceedings Logic Colloquium, number 80 in Studies in Logic and the Foundations of Mathematics, pages 157–174. North-Holland, 1975.*

[86] **R. Milner:** Algebras for communicating systems. *In Proc. AFCET/SMF joint colloquium in Applied Mathematics, Paris, 1978.*

[87] **R. Milner:** Synthesis of communicating behaviour. *In J. Winkowski, editor, Proc. 7th MFCS, number 64 in LNCS, pages 71–83, Zakopane, 1978. Springer Verlag.*

[88] **R. Milner:** Flowgraphs and flow algebras. *Journal of the ACM, 26(4):794–818, 1979.*

[89] **R. Milner:** A Calculus of Communicating Systems. *Number 92 in Lecture Notes in Computer Science. Springer Verlag, 1980.*

[90] **R. Milner:** Calculi for synchrony and asynchrony. *Theoretical Computer Science, 25:267–310, 1983.*

[91] **R. Milner:** A complete inference system for a class of regular behaviours. *Journal of Computer System Science, 28:439–466, 1984.*

[92] **R. Milner:** Communication and Concurrency. *Prentice Hall, 1989.*

[93] **R. Milner:** Calculi for interaction. *Acta Informatica, 33:707–737, 1996.*

[94] **R. Milner:** Communicating and Mobile Systems: the π-Calculus. *Cambridge University Press, ISBN 052164320, 1999.*

`http://www.cup.org/titles/`
`catalogue.asp?isbn=0521658691`

[95] **R. Milner:** Bigraphical reactive systems. *In K.G. Larsen and M. Nielsen, editors, Proceedings CONCUR '01, number 2154 in LNCS, pages 16–35. Springer Verlag, 2001.*

[96] **O. Jensen and R. Milner** Bigraphs and Mobile Processes. *Technical report, 570, Computer Laboratory, University of Cambridge, 2003.*

`http://citeseer.ist.psu.edu/`
`jensen03bigraphs.html`

`http://citeseer.ist.psu.edu/668823.html`

[97] **R. Milner, J. lParrow, and D. Walker:** A calculus of mobile processes. *Information and Computation, 100:1–77, 1992.*

[98] **F. Moller and P. Stevens:** Edinburgh Concurrency Workbench user manual (version 7.1).

`http://www.dcs.ed.ac.uk/home/cwb/`

[99] **F. Moller and C. Tofts:** A temporal calculus of communicating systems. *In J.C.M. Baeten and J.W. Klop, editors, Proceedings CONCUR'90, number 458 in LNCS, pages 401–415. Springer Verlag, 1990.*

[100] **X. Nicollin and J. Sifakis:** The algebra of timed processes ATP: Theory and application. *Information and Computation, 114:131– 178, 1994.*

[101] **D.M.R. Park:** Concurrency and automata on infinite sequences. *In P. Deussen, editor, Proceedings 5th GI Conference, number 104 in LNCS, pages 167–183. Springer Verlag, 1981.*

[102] **C.A. Petri:** Kommunikation mit Automaten. *PhD thesis, Institut fuer Instrumentelle Mathematik, Bonn, 1962.*

[103] **C.A. Petri:** Introduction to general net theory. *In W. Brauer, editor, Proc. Advanced Course on General Net Theory, Processes and Systems, number 84 in LNCS, pages 1–20. Springer Verlag, 1980.*

[104] **G.D. Plotkin:** A structural approach to operational semantics. *Technical Report DAIMI FN-19, Aarhus University, 1981.*

[105] **G.D. Plotkin:** The origins of structural operational semantics. *Journal of Logic and Algebraic Programming, Special Issue on Structural Operational Semantics, 2004.*

[106] **A. Pnueli:** The temporal logic of programs. *In Proceedings 19th Symposium on Foundations of Computer Science, pages 46–57. IEEE, 1977.*

[107] **G.M. Reed and A.W. Roscoe:** A timed model for communicating sequential processes. *Theoretical Computer Science, 58:249–261, 1988.*

[108] **M. Rem:** Partially ordered computations, with applications to VLSI design. *In J.W. de Bakker and J. van Leeuwen, editors, Foundations of Computer Science IV, volume 159 of Mathematical Centre Tracts, pages 1–44. Mathematical Centre, Amsterdam, 1983.*

[109] **S.A. Schneider:** Concurrent and Real-Time Systems (the CSP Approach). *Worldwide Series in Computer Science. Wiley, 2000.*

[110] **D.S. Scott and C. Strachey:** Towards a mathematical semantics for computer languages. *In J. Fox, editor, Proceedings Symposium Computers and Automata, pages 19–46. Polytechnic Institute of Brooklyn Press, 1971.*

[111] **Y.S. Usenko:** Linearization in μCRL. *PhD thesis, Technische Universiteit Eindhoven, 2002.*

[112] **B. Victor:** A Verification Tool for the Polyadic π-Calculus. *Licentiate thesis, Department of Computer Systems, Uppsala University, Sweden, May 1994. Report DoCS 94/50.*

[113] **T.A.C. Willemse:** Semantics and Verification in Process Algebras with Data and Timing. *PhD thesis, Technische Universiteit Eindhoven, 2003.*

[114] **Wang Yi:** Real-time behaviour of asynchronous agents. *In J.C.M. Baeten and J.W. Klop, editors, Proceedings CONCUR'90, number 458 in LNCS, pages 502– 520. Springer Verlag, 1990.*

[115] **L. Wischik:** New directions in implementing the π-calculus. *University of Bologna, August 2002.*

`http://www.fairdene.com/picalculus/`
`implementing-pi-c.pdf`

[116] **S. Yovine:** Kronos: A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer, 1:123–133, 1997.*

[117] **D. Zhang, R. Cleaveland, and E. Stark:** The integrated CWB-NC/PIOAtool for functional verification and performance analysis of concurrent systems. *In H. Garavel and J. Hatcliff, editors, Proceedings TACAS'03, number 2619 in Lecture Notes in Computer Science, pages 431–436. Springer-Verlag, 2003.*

[118] **E. Clarke, O. Grumberg, D. Peled:** Model checking. *MIT Press, 2001.*

[119] **J.Esparza:** Decidability of model-checking for infinite-state concurrent systems, *Acta Informatica, 34:85-107, 1997.*

[120] **P.A.Abdulla, A.Annichini, S.Bensalem, A.Bouajjani, P.Habermehl, Y.Lakhnech:** Verification of Infinite-State Systems by Combining Abstraction and Reachability Analysis, *Lecture Notes in Computer Science 1633, pages 146-159, Springer-Verlag, 1999.*

[121] **K.L.McMillan**: Verification of Infinite State Systems by Compositional Model Checking, *Conference on Correct Hardware Design and Verification Methods, pages 219-234, 1999.*

[122] **O.Burkart, D.Caucal, F.Moller, and B.Steffen:** Verification on infinite structures, *In J. Bergstra, A. Ponse and S. Smolka, editors, Handbook of Process Algebra, chapter 9, pages 545-623, Elsevier Science, 2001.*

[123] **D. Lee and M. Yannakakis.** Principles and Methods of Testing Finite State Machines - a Survey. The Proceedings of the IEEE, 84(8), pp 1090-1123, 1996.

[124] **G. Holzmann.** Design and Validation of Computer Protocols. Prentice-Hall, Englewood Cliffs, N.J., first edition, 1991.

[125] **G. Holzmann.** The SPIN Model Checker - Primer and Reference Manual. Addison-Wesley, 2003.

[126] **S. Huang, D. Lee, and M. Staskauskas.** Validation-Based Test Sequence Generation for Networks of Extended Finite State Machines. In Proceedings of FORTE/PSTV, October 1996.

[127] **J. J. Li and M. Segal.** Abstracting Security Specifications in Building Survivable Systems. In Proceedings of 22-th National Information Systems Security Conference, October 1999, Arlington, Virginia, USA.

[128] **Y.- J. Byun, B. A. Sanders, and C.-S. Keum.** Design Patterns of Communicating Extended Finite State Machines in SDL . In Proceedings of 8-th Conference on Pattern Languages of Programs (PLoP'2001), September 2001, Monticello, Illinois, USA.

[129] **J. J. Li and W. E. Wong.** Automatic Test Generation from Communicating Extended Finite State Machine (CEFSM)-Based Models. In Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'02), pp. 181-185, 2002.

[130] **S. Chatterjee.** EAI Testing Automation Strategy. In Proceedings of 4-th QAI Annual International Software Testing Conference in India, Pune, India, February 2004

[131] ITU Telecommunication Standardization Sector (ITU-T), Recommendation Z.100, CCITT Specification and Description Language (SDL), Geneva 1994.

[132] Information Processing Systems - Open Systems Interconnection: Estelle, A Formal Description Technique Based on Extended State Transition Model, ISO International Standard 9074, June 1989.

[133] ISOIEC. LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard 8807, 1988.

[134] **D. Harel.** A Visual Formalism for Complex Systems. Science of Computer Programming, 8:231-274, 1987.

[135] **D. Harel and A. Naamad.** The STATEMATE semantics of statecharts. ACM Transactions on Software Engineering and Methodology (Also available as technical report of Weizmann Institute of Science, CS95-31), 5(4):293-333, Oct 1996.

[136] **M. Bozga, J. C. Fernandez, L. Ghirvu, S. Graf, J. P. Krimm, and L. Mounier.** IF: An intermediate representation and validation environment for timed asynchronous systems. In Proceedings of Symposium on Formal Methods 99, Toulouse, number 1708 in LNCS. Springer Verlag, September 1999.

[137] **M. Bozga, S. Graf, and L. Mounier.** IF-2.0: A validation environment for componentbased real-time systems. In Proceedings of Conference on Computer Aided Verification, CAV'02, Copenhagen, LNCS. Springer Verlag, June 2002.

[138] **M. Bozga, D. Lesens, and L. Mounier.** Model-Checking Ariane-5 Flight Program. In Proceedings of FMICS'01, Paris, France, pages 211-227. INRIA, 2001.

[139] **M. Bozga, S. Graf, and L. Mounier.** Automated validation of distributed software using the IF environment. In 2001 IEEE International Symposium on Network Computing and Applications (NCA 2001). IEEE, October 2001.

[140] **M. Bozga and Y. Lakhnech.** IF-2.0 common language operational semantics. Technical report, 2002. Deliverable of the IST Advance project, available from the authors.

[141] `http://www-128.ibm.com/developerworks/`
`library/specification/ws-bpel/`

[142] `http://www.bpml.org`

[143] `http://www.wfmc.org/standards/docs/`
`better_maths_better_processes.pdf`

[144] `http://www.fairdene.com/picalculus/`

[145] `http://www.bpmi.org/bpmi-library/`
`2B6EA45491.workflow-is-just-a-pi-process.pdf`

[146] `http://www.fairdene.com/picalculus/`
`bpm3-apx-theory.pdf`

[147] http://www.bpm3.com

[148] http://portal.acm.org/citation.cfm?id=1223649

[149] http://www.w3.org/TR/ws-cdl-10/