A New Approach for Specification and Verification Of Distributed Agents

by

Andrew M. Mironov
Virendra C. Bhavsar

TR98-120a, June 2000

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
E-mail: fcs@unb.ca
www: http://www.cs.unb.ca

Contents

1 Introduction

The problem of *specification* and *formal verification* of distributed communicating systems (DCS) is one of most important problems in theoretical computer science. DCSs arise in a wide range of applications, for example in distributed information processing, telecommunications, control of complex systems like aircrafts and nuclear reactors. The problems of human safety necessitate formal proofs that DCSs which control functioning of such systems are absolutely trustworthy. The mathematical formalization of the problem of checking correctness of a DCS is known as the problem of specification and verification. **A specification** of a DCS is a formal representation of desired properties of the DCS. The problem of **verification** of a DCS consists of construction of a formal proof that the DCS meets its specification.

A solution of the problem of specification and verification of a DCS depends on the choice of a formal model of the DCS and a specification language.

There are several algebraic and logical approaches to representation, specification and verification of DCSs. The most popular ones are CCS and π–calculus ([17], [18], [19]), the approaches based on graph rewriting and partial ordering semantics ([4], [5], [6], [20], [9], [10], [21], [22]), CSP ([11]), UNITY ([2]), Input-Output Automata ([13], [14]), temporal logic approaches ([3], [7], [12], [16]). These approaches use various techniques for the solution of verification problem. One of the techniques is to convert the verification problem to the problem of search of a deduction of a formula or to the problem of proving of observable congruence between a pair of process expressions. A major difficulty of this techniques is the large complexity of deduction search procedure with respect to the size of description of a given DCS and its specification.

In the approaches which are founded on temporal logic, the most popular technique of verification consists of construction of a state transition diagram for a distributed program and reduction of the verification problem to the problem of model checking. A major difficulty of this technique is the large complexity of state transition diagrams.

In the theory of input-output (I/O) automata the verification technique is founded on the construction of invariants, i.e. conditions which are true in all reachable states of an I/O automaton. The disadvantage of this approach is again higher complexity of an I/O automaton corresponding to a DCS which is by definition a cartesian product of I/O automata corresponding to sequential subsystems of the DCS.

In the present report we introduce a new formal model for distributed communicating systems, with the specification language and the corresponding verification technique, which eliminates some disadvantages of the above verification techniques.

The proposed verification technique is founded on the modular approach, that consists of two stages: (a) verification of all subsystems of the DCS separately, and (b) proving that the conjunction of specifications of these subsystems and some special formulas implies specification of the whole system.

The main concepts of this model are *sequential* and *distributed agents*. A sequential agent is a formal model of sequential subsystems of distributed communicating systems. A sequential agent is represented in a form similar to a flowchart of a sequential program.

A distributed agent is a formal model of a DCS on the whole. Distributed agent is a set of sequential agents with a set of communication channels connecting these sequential agents.

Our approach to the problem of verification is a generalization of Floyd's inductive assertion method (see [8], [15], ch.3).

The proposed approach to specification and verification has some advantages in comparison with other approaches. Since we do not employ the operation of cartesian product commonly used in construction of state transition graphs, the complexity of verification of a DCS is considerably reduced. The proposed verification technique allows the interactive implementation the verification process. The use of fixpoint constructions in the specification language results in a simple and precise description of the behavior of a DCS.

The report is organized as follows. In Section 2 we introduce the concepts of types, data expressions, and queues. The concepts of sequential agents and distributed agents are defined in Sections 3 and 4. The specification language for expressing properties of distributed agents is given in Section 5. In Section 6 we give an approach to verification of distributed agents. The example of alternating bit protocol is used in Section 7 to illustrate the concepts introduced in the report. Finally, concluding remarks are given along with directions of future research.

2 Types, queues and data expressions

In this section we define the concepts which are necessary for formal description and specification of sequential and distributed agents.

2.1 Types

Assume that a set $\mathcal{T}$ is given, the elements of which are called **types**.

The concept of a type in our model is a generalization of the concept of a data type in programming languages. For example, the following entities can be considered as types: a natural number, an integer number, a real number, a boolean value, a list, a set and a string.

Let M be a set, every element m of which is associated with some type $type(m) \in \mathcal{T}$. In this case for every $\tau \in \mathcal{T}$ the symbol M_τ denotes the set

$$\{m \in M \mid type(m) = \tau\}.$$

For every $\tau \in \mathcal{T}$ and every $m \in M_\tau$ we say that m **is of the type** τ.

2.2 Lists

For every set M a **list** of elements of the set M is any finite (possible empty) string, components of which are elements of M. The empty list is denoted by the symbol Λ. The symbol M^* denotes the set of all possible lists of elements of M.

Let M be a set, every element m of which is associated with some type $type(m) \in \mathcal{T}$. In this case for every list $N \in M^*$ the symbol $type(N)$ denotes the list

- $(type(m_1), \ldots, type(m_n))$, if $N = (m_1, \ldots, m_n)$, where $n \geq 1$,
- Λ, if $N = \Lambda$.

2.3 Data values

We assume that a set $\mathcal{D}$ is given, every element d of which is associated with some type $type(d)$. The elements of the set $\mathcal{D}$ are called **data values**.

For every type $\tau \in \mathcal{T}$ the symbol ω denotes an element which does not belong to $\mathcal{D}_\tau$ (this symbol is the same for all types).

We assume that

- for every type τ the set $\mathcal{T}$ contains a type which is denoted by the symbol $\hat{\tau}$, and $\mathcal{D}_{\hat{\tau}} \stackrel{\text{def}}{=} \mathcal{D}_\tau \sqcup \{\omega\}$,
- for every list $T \in \mathcal{T}^*$ the set $\mathcal{T}$ contains a type which is equal to the list T, and the set $\mathcal{D}_T$ is equal
  - to the cartesian product
    $$\mathcal{D}_{\tau_1} \times \ldots \times \mathcal{D}_{\tau_n},$$
    if $T = (\tau_1, \ldots, \tau_n)$, where $n \geq 1$,
  - to the one-element set $\{1\}$, if $T = \Lambda$.

2.4 Queues

We assume that for every type τ the set $\mathcal{T}$ contains a type which is denoted by the symbol $\bar{\tau}$, and the set $\mathcal{D}_{\bar{\tau}}$ consists of all infinite strings, elements of which belong to the set $\mathcal{D}_{\hat{\tau}}$. Elements of the set $\mathcal{D}_{\bar{\tau}}$ are called **queues**. Types of the form $\bar{\tau}$ are called **queue types**.

For every $\tau \in \mathcal{T}$ and every queue $Q = (d_1, d_2, \ldots) \in \mathcal{D}_{\hat{\tau}}$

- the symbol **head**(Q) denotes the first element of Q:
  $$\mathbf{head}(Q) \stackrel{\text{def}}{=} d_1,$$
- the symbol **tail**(Q) denotes Q with its first component removed:
  $$\mathbf{tail}(Q) \stackrel{\text{def}}{=} (d_2, d_3, \ldots),$$
- for every $k \geq 1$ the symbol $Q[k]$ denotes the k–th element of Q,
- for every $d \in \mathcal{D}_\tau$ the symbol $d \cdot Q$ denotes the queue which is the concatenation of d and Q:
  $$d \cdot Q \stackrel{\text{def}}{=} (d, d_1, d_2, \ldots).$$

2.5 Functional types and functional symbols

We assume that for every string of the form

$$(T \to \tau)$$

where $T \in \mathcal{T}^*$ and $\tau \in \mathcal{T}$, the set $\mathcal{T}$ contains a type, which is equal to this string. The types of this form are called **functional types**.

We assume that a set $\mathcal{F}$ is given. Elements of $\mathcal{F}$ are called **functional symbols**. Every functional symbol f is associated with

- a functional type $type(f)$,
- a mapping (denoted by the same symbol f) of the form

  $$f : \mathcal{D}_T \to \mathcal{D}_\tau$$

  where T and τ are such that $type(f) = (T \to \tau)$.

For every functional type $(T \to \tau)$ and every $f \in \mathcal{F}$ such that $type(f) = (T \to \tau)$

- the symbol $dom\_type(f)$ denotes the list T, and
- the symbol $im\_type(f)$ denotes the type τ.

If $T(f)$ has the form $(\Lambda \to \tau)$, i.e. the mapping f has the form $f : \{1\} \to \mathcal{D}_\tau$, then the symbol f denotes also the element $f(1)$ of the set $\mathcal{D}_\tau$.

2.6 Special types and functional symbols

The following assumptions are used.

- The set $\mathcal{T}$ contains the type *bool*, such that $\mathcal{D}_{bool}$ is the two-element set $\{\top, \bot\}$,
- The set $\mathcal{F}$ contains the following functional symbols:
  - $\top, \bot :\quad T(\top) \stackrel{\text{def}}{=} T(\bot) \stackrel{\text{def}}{=} (\Lambda \to bool)$,
  - $\neg :\quad T(\neg) \stackrel{\text{def}}{=} (bool \to bool)$,

– $\wedge, \vee, \to, \leftrightarrow$:
$T(\wedge) \stackrel{\text{def}}{=} T(\vee) \stackrel{\text{def}}{=} T(\to) \stackrel{\text{def}}{=} T(\leftrightarrow) \stackrel{\text{def}}{=}$
$\stackrel{\text{def}}{=} ((bool,\ bool) \to bool)$.

- The mappings associated with the functional symbols $\top, \bot, \neg, \wedge, \vee, \to$ $, \leftrightarrow$, are the same as for the standard boolean operations.

- For every non-empty list

$$T = (\tau_1, \dots, \tau_n) \in \mathcal{T}^*$$

the set $\mathcal{F}$ contains the following functional symbols (which are called **projections**):
$\pi_1 : \quad type(\pi_1) = (T \to \tau_1)$;
$\dots$
$\pi_n : \quad type(\pi_n) = (T \to \tau_n)$.
Note that for every list $T \in \mathcal{T}^*$ the projections are denoted by the same symbol π_k.

For every $k = 1, \dots, n$ the mapping

$$\pi_k : \mathcal{D}_{\tau_1} \times \dots \times \mathcal{D}_{\tau_n} \to \mathcal{D}_{\tau_k}$$

is a projection on the k–th component.

- For every $\tau \in \mathcal{T}$ the following functional symbols belong to $\mathcal{F}$:

1. **head** : $\quad type(\mathbf{head}) = (\bar{\tau} \to \hat{\tau})$,
2. **tail** : $\quad type(\mathbf{tail}) = (\bar{\tau} \to \bar{\tau})$,
3. $\cdot$ (concatenation) : $\quad type(\cdot) = ((\hat{\tau}, \bar{\tau}) \to \bar{\tau})$.

The mappings associated with the above functional symbols are defined according to the concepts in subsection 2.4, i.e. the mapping **head** maps every queue Q to the element $\mathbf{head}(Q)$, and so on.

2.7 Variables

We assume that a set $\mathcal{X}$ is given. The elements of $\mathcal{X}$ are called **variables**. Every $x \in \mathcal{X}$ is associated with some type $type(x) \in \mathcal{T}$.

2.8 Data expressions

The set $\mathcal{E}$ of **data expressions** is defined by induction. Every data expression $e \in \mathcal{E}$ is associated with some type, which is denoted by the symbol $type(e)$.

1. Every data value $d \in \mathcal{D}$ is a data expression of the type $type(d)$.
2. Every variable $x \in \mathcal{X}$ is a data expression of the type $type(x)$.
3. For every $f \in \mathcal{F}$, and every list E of data expressions, such that $type(E) = dom\_type(f)$, the string $f(E)$ is a data expression of the type $im\_type(f)$.

For every data expression $e \in \mathcal{E}$, the symbol $Var(e)$ denotes the set of variables that have an occurrence in e.

2.9 Evaluation of data expressions

Let $X \subseteq \mathcal{X}$. An **evaluation** of variables from the set X is a mapping

$$Eval : X \to \mathcal{D}$$

such that for every $x \in X$

$$Eval(x) \in \mathcal{D}_{type(x)}.$$

The mapping $Eval$ can be extended to the mapping which is denoted by the same symbol $Eval$, and has the form

$$Eval : \{e \in \mathcal{E} \mid Var(e) \subseteq X\} \to \mathcal{D}.$$

This mapping is defined as follows:

1. If $e = d \in \mathcal{D}$, then $Eval(e) \stackrel{\text{def}}{=} d$.
2. If $e = f \in \mathcal{F}$, where $dom\_type(f) = \Lambda$, then
$$Eval(e) \stackrel{\text{def}}{=} f \quad (\in \mathcal{D}_{im\_type(f)}).$$
3. If $e = f(e_1, \ldots, e_n)$, then
$$Eval(e) \stackrel{\text{def}}{=} f(Eval(e_1), \ldots, Eval(e_n)).$$

3 Sequential agents

In this section we define the concept of a sequential agent. A sequential agent is a formal model of sequential subsystems of distributed communicating systems.

3.1 Input ports and output ports

Assume that the pair $Inputs, Outputs$ of disjoint sets is given, and every element $p \in Inputs \sqcup Outputs$ is associated with some queue type $type(p)$. Elements of the set $Inputs$ are called **input ports**, elements of the set $Outputs$ are called **output ports**. For every input or output port p there is a variable of the type $type(p)$, which is denoted by the same symbol p. Hereafter every port p and the variable p which correspons to this port are considered as the same entities.

3.2 Definition of a sequential agent

A **sequential agent** is an oriented graph SA with the following properties.

1. SA has one chosen node $root(SA)$ called a **root**.
2. Every edge a of SA has a label $\langle a \rangle$ of one of the following forms:

   input action: $\langle a \rangle = \mathbf{input}(p, x)$,
   where $p \in Inputs$, $x \in \mathcal{X}$, $type(p) = \overline{type(x)}$,

   output action: $\langle a \rangle = \mathbf{output}(p, e)$,
   where $p \in Outputs$, $e \in \mathcal{E}$, $type(p) = \overline{type(e)}$,

   default action: $\langle a \rangle = \mathbf{default}$,

   assignment action: $\langle a \rangle = (X := E)$, where

   (a) $X \in \mathcal{X}^*$ is a non-empty list of distinct variables, and
   (b) the list $E \in \mathcal{E}^*$ is such that
   $type(E) = type(X)$.

   boolean condition: $\langle a \rangle = b$, where $b \in \mathcal{E}_{bool}$.

3. For every node n of SA one and only one of the following conditions holds.

(a) There are only two edges outgoing from n.
One of these edges has the label of the form $\mathbf{input}(p, x)$.
The second edge has the label **default**.

(b) There are only two edges outgoing from n.
One of these edges has the label of the form $\mathbf{output}(p, e)$.
The second edge has the label **default**.

(c) There is only one edge outgoing from n.
This edge has the label of the form $(X := E)$.

(d) There are only two edges outgoing from n.
One of these edges has the label of the form b, where $b \in \mathcal{E}_{bool}$.
The second edge has the label $\neg b$.

(e) There are no edges outgoing from n.
(Nodes with such property are called **terminal nodes**.)

We will use the following notations.

1. $Nodes(SA)$ is the set of nodes of SA.

2. $Edges(SA)$ is the set of edges of SA.

3. For every $a \in Edges(SA)$ the symbols $start(a)$ and $end(a)$ denote nodes which are the start and the end of the edge a, respectively.

4. $Inputs(SA)$ is the set of all input ports $p \in Inputs$ such that there is an edge in SA with the label of the form $\mathbf{input}(p, x)$.

5. $Outputs(SA)$ is the set of all output ports $p \in Outputs$ such that there is an edge in SA with the label of the form $\mathbf{output}(p, e)$.

6. $Ports(SA) \stackrel{\text{def}}{=} Inputs(SA) \cup Outputs(SA)$,

7. $Var(SA)$ is the set of all variables that occurre in the labels of the edges of SA and does not belong to $Ports(SA)$.

3.3 Paths in sequential agents

Let SA be a sequential agent.

A **path** in SA is a finite or infinite sequence A of edges of SA of the form

$$A = (a_1, \ldots, a_m) \quad (m \geq 1) \quad \text{or} \quad A = (a_1, a_2, \ldots)$$

such that for every $i \geq 1$

$$end(a_i) = start(a_{i+1}),$$

if A has edges with the numbers i and $i+1$.

The node $start(a_1)$ is called the **start** of the path A, and is denoted also by the symbol $start(A)$.

If the path A is finite, i.e. $A = (a_1, \ldots, a_m)$, then the node $end(a_m)$ is called the **end** of the path A, and is denoted also by the symbol $end(A)$.

We assume that any sequential agent SA under consideration has the following property: for every node
$n \in Nodes(SA)$ there is a finite path A such that

$$start(A) = root(SA), \quad end(A) = n.$$

3.4 Traces

Let SA be a sequential agent, and Q be a $Ports(SA)$–indexed set of queues of the form

$$Q = \{Q_p \in \mathcal{D}_{type(p)} \mid p \in Ports(SA)\}.$$

A **trace** of SA associated with the set Q, is any finite or infinite sequence tr of the form

$$tr = \{(Node(t), Edge(t), Eval(t)) \mid t \geq 1\}$$

where for every $t \geq 1$

1. $Node(t) \in Nodes(SA)$,
2. $Edge(t) \in (Edges(SA))^{\wedge} \stackrel{\text{def}}{=} Edges(SA) \sqcup \{\omega\}$,

3. $Eval(t)$ is an evaluation of variables from the set $Var(SA)$,

   (hereafter for every data expression e the value of e with respect to the evaluation $Eval(t)$ will be denoted by the symbol $e(t)$)

such that

- $Node(1) = root(SA)$,
- the trace tr is finite, and consists of t components (where $t \geq 1$) if and only if
  - the node $Node(t)$ is terminal,
  - $\forall t' < t$ the node $Node(t')$ is not terminal,
  - $Edge(t) = \omega$,

and for every $t \geq 1$ such that tr has components with the numbers t and $t + 1$, the following conditions hold:

1. $start(Edge(t)) = Node(t)$.
2. $end(Edge(t)) = Node(t + 1)$.
3. If $\langle Edge(t) \rangle = \mathbf{input}(p, x)$, then
   (a) $Q_p[t] \neq \omega$,
   (b) $\forall q \in Ports(SA) \setminus \{p\} \quad Q_q[t] = \omega$,
   (c) $x(t + 1) = Q_p[t]$,
   (d) $\forall y \in Var(SA) \setminus \{x\} \quad y(t + 1) = y(t)$.
4. If $\langle Edge(t) \rangle = \mathbf{output}(p, e)$, then
   (a) $Q_p[t] = e(t)$,
   (b) $\forall q \in Ports(SA) \setminus \{p\} \quad Q_q[t] = \omega$,
   (c) $\forall x \in Var(SA) \quad x(t + 1) = x(t)$.
5. If $\langle Edge(t) \rangle = \mathbf{default}$, then
   (a) $\forall p \in Ports(SA) \quad Q_p[t] = \omega$,

(b) $\forall x \in Var(SA) \quad x(t+1) = x(t)$.

6. If $\langle Edge(t) \rangle = (X := E)$, where $X = (x_1, \ldots, x_n)$ and $E = (e_1, \ldots, e_n)$, then

   (a) $\forall p \in Ports(SA) \quad Q_p[t] = \omega$,

   (b) $\forall i = 1, \ldots, n \quad x_i(t+1) = e_i(t)$,

   (c) $\forall y \in Var(SA) \setminus \{x_1, \ldots, x_n\} \quad y(t+1) = y(t)$.

7. If $\langle Edge(t) \rangle = b \in \mathcal{E}_{bool}$, then

   (a) $\forall p \in Ports(SA) \quad Q_p[t] = \omega$,

   (b) $b(t) = \top$,

   (c) $\forall x \in Var(SA) \quad x(t+1) = x(t)$.

The symbol $Beh(SA)$ denotes the set of all $Ports(SA)$–indexed sets Q of the above form, such that there is a trace of SA associated with the set Q.

4 Distributed agents

In this section we define the concept of a distributed agent. A distributed agent is a formal model of a distributed communicating system on the whole.

4.1 Definition of a distributed agent

A **distributed agent** is a list

$$DA \stackrel{\text{def}}{=} (SA^1, \ldots, SA^k; Channels),$$

where

1. $SA^1, \ldots, SA^k$ is some list of sequential agents, such that for every pair of distinct indices $i, j \in \{1, \ldots, k\}$
   $Inputs(SA^i) \cap Inputs(SA^j) = \emptyset$, and
   $Outputs(SA^i) \cap Outputs(SA^j) = \emptyset$.

2. *Channels* is a binary relation of the form

$$Channels \subseteq \bigcup_{i=1}^{k} Inputs(SA^i) \ \times \ \bigcup_{i=1}^{k} Outputs(SA^i)$$

such that

(a) if $(p, q) \in Channels$ and $(p, q') \in Channels$, then $q = q'$,

(b) if $(p, q) \in Channels$ and $(p', q) \in Channels$, then $p = p'$,

(c) $\forall (p, q) \in Channels \quad type(p) = type(q)$.

Every element (p, q) of the set *Channels* is called **a communication channel**, which connects the input port p with the output port q.

4.2 Input ports and output ports of distributed agents

Let $DA = (SA^1, \ldots, SA^k; Channels)$ be a distributed agent.

The sets $Inputs(DA)$ and $Outputs(DA)$ of **input and output ports of** DA are defined as follows:

1. $Inputs(DA) \stackrel{\text{def}}{=}$
   $\stackrel{\text{def}}{=} (\bigcup_{i=1}^{k} Inputs(SA^i)) \setminus \{p \mid \exists\, q : (p, q) \in Channels\}$,

2. $Outputs(DA) \stackrel{\text{def}}{=}$
   $\stackrel{\text{def}}{=} (\bigcup_{i=1}^{k} Outputs(SA^i)) \setminus \{q \mid \exists\, p : (p, q) \in Channels\}$,

3. $Ports(DA) \stackrel{\text{def}}{=} Inputs(DA) \cup Outputs(DA)$.

For every $i = 1, \ldots, k$

- the set $Obs\_Inputs(SA^i)$ of **observable input ports** of SA^i is the set $Inputs(DA) \cap Inputs(SA^i)$,

- the set $Hid\_Inputs(SA^i)$ of **hidden input ports** of SA^i is the set $Inputs(SA^i) \setminus Obs\_Inputs(SA^i)$,

- the set $Obs\_Outputs(SA^i)$ of **observable output ports** of SA^i is the set $Outputs(DA) \cap Outputs(SA^i)$,

- the set $Hid\_Outputs(SA^i)$ of **hidden output ports** of SA^i is the set $Outputs(SA^i) \setminus Obs\_Outputs(SA^i)$.

4.3 Pictorial representation of distributed agents

Every distributed agent

$$DA \stackrel{\text{def}}{=} (SA^1, \ldots, SA^k; Channels)$$

can be represented by its **flow graph**, which displays connections between the hidden input ports and the hidden output ports.

The flow graph for DA consists of

- a list $Oval^1, \ldots, Oval^k$ of ovals where for every $i = 1, \ldots, k$ the oval $Oval^i$ corresponds to the sequential agent SA^i, and has white and black circles with labels from the set $Ports(SA^i)$ on its exterior, which depict respectively the input and output ports of the sequential agent SA^i:
  - labels of observable input and output ports are depicted by the normal font, and
  - labels of hidden input and output ports are depicted by a smaller font,
- a set of lines that represent the pairs from the set $Channels$: for every pair $(p, q) \in Channels$ there is a line from the white circle that corresponds to the input port p to the black circle that corresponds to the output port q.

For example, the flow graph for the distributed agent

$$DA \stackrel{\text{def}}{=} (SA^1, SA^2, SA^3; Channels),$$

where

- $Inputs(SA^1) = \{p_1\}$, $Outputs(SA^1) = \{q_1\}$,
- $Inputs(SA^2) = \{p_2, \}$, $Outputs(SA^2) = \{q_2\}$,
- $Inputs(SA^3) = \{p, p'_1, p'_2\}$,
  $Outputs(SA^3) = \{q, q'_1, q'_2\}$,

- $Channels \stackrel{\mathrm{def}}{=} \{(p_1, q_1'),\ (p_1', q_1),\ (p_2, q_2'),\ (p_2', q_2)\}$,

is represented pictorially as follows:

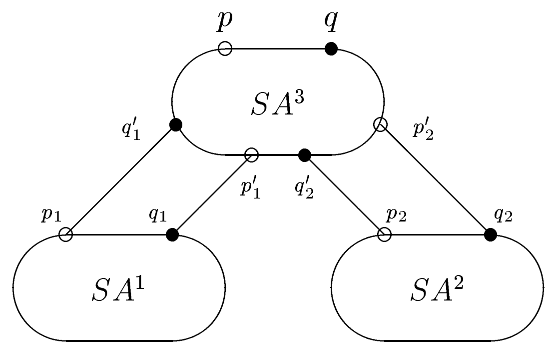

Figure 1: an example of a flow graph.

4.4 Traces of distributed agents

Let $DA \stackrel{\mathrm{def}}{=} (SA^1, \ldots, SA^k; Channels)$ be a distributed agent, and Q be a list of the form $Q = (Q^1, \ldots, Q^k)$, such that

- for every $i = 1, \ldots, k$

$$Q^i = \{Q^i_p \mid p \in Ports(SA^i)\} \in Beh(SA^i),$$

- for every pair $i, j \in \{1, \ldots, k\}$ and every pair $(p, q) \in Channels$, such that $p \in Ports(SA^i)$ and $q \in Ports(SA^j)$, the equality $Q^i_p = Q^j_q$ holds.

A **trace** of DA associated with Q, is a list

$$\left(\begin{array}{l} tr^1 = \{(Node^1(t), Edge^1(t), Eval^1(t)) \mid t \geq 1\} \\ \ldots \\ tr^k = \{(Node^k(t), Edge^k(t), Eval^k(t)) \mid t \geq 1\} \end{array} \right)$$

of traces of $SA^1, \ldots, SA^k$ associated with $Q^1, \ldots, Q^k$ respectively, such that for every $(p, q) \in Channels$, and every $t \geq 1$ the following statement holds:

- if there is an edge outgoing from $Node^i(t)$, where i is such that $p \in Inputs(SA^i)$, and the label of this edge is of the form $\mathbf{input}(p, x)$, and

- if there is an edge outgoing from $Node^j(t)$, where j is such that $q \in Outputs(SA^j)$, and the label of this edge is of the form $\mathbf{output}(q, e)$,

then $\langle Edge^i(t)\rangle = \mathbf{input}(p, x)$, $\langle Edge^j(t)\rangle = \mathbf{output}(q, e)$.

For every list Q of the above form **the observable part of** Q is the $Ports(DA)$–indexed set $Obs(Q)$, which consists of all queues of the form Q^i_p, where
$p \in Ports(DA)$, and i is such that $p \in Ports(SA^i)$.

The symbol $Beh(DA)$ denotes the set of all $Ports(DA)$–indexed sets of the form $Obs(Q)$, where the list Q is such that there is a trace of DA associated with Q.

5 Specification language

In this section we describe the specification language, which allows to express properties of sequential and distributed agents. The main novelty of this language is the use of fixpoint constructions, which are a generalization of temporal operators.

5.1 Relational symbols

We assume that the set $\mathcal{R}$ of **relational symbols** is given. Every symbol $\rho \in \mathcal{R}$ is associated with a type $type(\rho)$ of the form

$$type(\rho) = (\tau_1, \ldots, \tau_m) \in \mathcal{T}^*.$$

5.2 Specification systems

We assume that a set of **symbols of specification systems** $\mathcal{S}$ is given. Every symbol $\Sigma \in \mathcal{S}$ is associated with a **specification system**, i.e. with a set (which is denoted by the same symbol Σ) of formal equations of the form

$$\{\rho_i(X_i) = s_i \mid i \in \Im\},$$

where $\Im$ is an arbitrary set of indices, and for every $i \in \Im$

- ρ_i is a relational symbol such that $\rho_i \neq \rho_j$ if $i \neq j$,
- X_i is a list of distinct variables such that

$$type(\rho_i) = type(X_i),$$

- s_i is a specification expression.

5.3 Specification expressions

In this subsection we define the set *Spec* of specification expressions. Every $s \in Spec$ is associated with a type
$type(s) \in \mathcal{T}$. For every $s \in Spec$

- the symbol $Var(s)$ denotes the set of all variables that have an occurrence in s,
- the symbol $Var\_List(s)$ denotes a list which consists of all elements of the set $Var(s)$,
  (we assume that the list $Var\_List(s)$ is uniquely determined by the specification expression s)
- the symbol $Rel(s)$ denotes the set of all relational symbols that have an occurrence in s.

The set *Spec* of specification expressions is defined as follows.

data expressions:
Every data expression $e \in \mathcal{E}$ is a specification expression of the type $type(e)$.

application of functional symbols:
For every list $S \in Spec^*$, and every $f \in \mathcal{F}$, such that

$$dom\_type(f) = type(S),$$

the string $f(S)$ is a specification expression of the type $im\_type(f)$.

application of relational symbols:

For every $\rho \in \mathcal{R}$, and every list $S \in Spec^*$, such that

$$type(S) = type(\rho),$$

the string $\rho(S)$ is a specification expression of the type *bool*.

fixpoint constructions:

For

- every specification system

  $$\Sigma = \{\rho_i(X_i) = s_i \mid i \in \Im\},$$

  such that for every $i \in \Im$
  - $type(s_i) = bool$,
  - s_i does not contain symbols from $\mathcal{S}$,
  - $Rel(s_i) \subseteq \{\rho_i \mid i \in \Im\}$,
  - for every variable x from the set $Var(s_i)$
    x belongs to the list X_i,
- every $i \in \Im$, and
- every list S of specification expressions, such that

  $$type(S) = type(X_i),$$

the string $\Sigma_i(S)$ is a specification expression of the type *bool*.

By definition,

$$Var\_List(\Sigma_i(S)) \stackrel{\text{def}}{=} Var\_List(\rho_i(S)).$$

A specification expression of the type *bool* is called **a boolean specification expression**.

For every n–tuple $s_1, \ldots, s_n$ of boolean specification expressions, the conjunction $s_1 \wedge \ldots \wedge s_n$ and the disjunction $s_1 \vee \ldots \vee s_n$ will also be denoted as

$$\left\{ \begin{array}{c} s_1 \\ \ldots \\ s_n \end{array} \right\} \quad \text{and} \quad \left[\begin{array}{c} s_1 \\ \ldots \\ s_n \end{array} \right]$$

respectively.

5.4 Substitutions in specification expressions

A **substitution operator** is a pair

$$\theta = (X, S) \in \mathcal{X}^* \times Spec^*,$$

where X is a list of distinct variables, and
$type(S) = type(X)$.

For every substitution operator $\theta = (X, S)$ the symbol θ denotes also a mapping $\theta : Spec \to Spec$, which is defined as follows:

1. If $X = (x_1, \ldots, x_m)$, $S = (s_1, \ldots, s_m)$, and there is $i \in \{1, \ldots, m\}$, such that $s = x_i$, then
$\theta(s) \stackrel{\text{def}}{=} s_i$.

2. If $s = d \in \mathcal{D}$ or $s = f \in \mathcal{F}$ where $dom\_type(f) = \Lambda$, or $s \in \mathcal{X} \setminus \{x_1, \ldots, x_m\}$, then $\theta(s) \stackrel{\text{def}}{=} s$.

3. If $s = f(s'_1, \ldots, s'_k)$, where $f \in \mathcal{F}$ and $k \geq 1$, then $\theta(s) \stackrel{\text{def}}{=} f(\theta(s'_1), \ldots, \theta(s'_k))$.

4. If $s = \rho(s'_1, \ldots, s'_k)$, where $\rho \in \mathcal{R}$ and $k \geq 1$, then $\theta(s) \stackrel{\text{def}}{=} \rho(\theta(s'_1), \ldots, \theta(s'_k))$.

5. If $s = \Sigma_i(s'_1, \ldots, s'_k)$, then $\theta(s) \stackrel{\text{def}}{=} \Sigma_i(\theta(s'_1), \ldots, \theta(s'_k))$.

If the list X is a concatenation of lists $X_1, \ldots, X_m$ and the list S is a concatenation of lists $S_1, \ldots, S_m$ of corresponding types, i.e. the lists $X_1, \ldots, X_m$ and $S_1, \ldots, S_m$ have the form

$$\begin{array}{lll} X_1 = (x_{1,1}, \ldots, x_{1,l_1}), & \ldots, & X_m = (x_{m,1}, \ldots, x_{m,l_m}), \\ S_1 = (s_{1,1}, \ldots, s_{1,l_1}), & \ldots, & S_m = (s_{m,1}, \ldots, s_{m,l_m}). \end{array}$$

and

$$\begin{array}{l} X = (x_{1,1}, \ldots, x_{1,l_1}, \quad \ldots, \quad x_{m,1}, \ldots, x_{m,l_m}) \\ S = (s_{1,1}, \ldots, s_{1,l_1}, \quad \ldots, \quad s_{m,1}, \ldots, s_{m,l_m}) \end{array}$$

then the substitution operator $\theta = (X, S)$ has the equivalent notation

$$\left\{ \begin{array}{l} X_1 := S_1 \\ \ldots \\ X_m := S_m \end{array} \right\}.$$

5.5 Interpretation of specification expressions without relational symbols

For every $s \in Spec$ such that $Rel(s) = \emptyset$, **an interpretation of** s is a mapping

$$[\![s]\!] : \mathcal{D}_{type(Var\_List(s))} \to \mathcal{D}_{type(s)},$$

which is defined as follows:

1. If $s = x \in \mathcal{X}$, then
$$[\![s]\!] : \mathcal{D}_{type(x)} \to \mathcal{D}_{type(x)}$$
is the identity mapping.

2. If $s = d \in \mathcal{D}$ or $s = f \in \mathcal{F}$, where $dom\_type(f) = \Lambda$, then
$$[\![s]\!] : \{1\} \to \mathcal{D}_{type(s)}$$
maps the element 1 to the element d or f respectively.

3. If $s = f(s_1, \dots, s_m)$, where $f \in \mathcal{F}$, $m \geq 1$, and
$Var\_List(s_1) = (x_1, \dots, x_k)$,
$\dots$
$Var\_List(s_m) = (y_1, \dots, y_l)$,
$Var\_List(s) = (z_1, \dots, z_n)$,
then the mapping $[\![s]\!]$ is a composition of the form
$$f \circ ([\![s_1]\!] \times \dots \times [\![s_m]\!]).$$
Here the mapping
$$[\![s_1]\!] \times \dots \times [\![s_m]\!] : \mathcal{D}_{type(Var\_List(s))} \to \mathcal{D}_{dom\_type(f)}$$
maps every list $(d_1, \dots, d_n)$ to the list
$$([\![s_1]\!](d_{i_1}, \dots, d_{i_k}), \dots, [\![s_n]\!](d_{j_1}, \dots, d_{j_l})),$$
where the numbers $i_1, \dots, i_k$, ..., $j_1, \dots, j_l$ are such that
$$z_{i_1} = x_1, \dots, z_{i_k} = x_k,$$
$$\dots$$
$$z_{j_1} = y_1, \dots, z_{j_l} = y_l.$$

5.6 Interpretation of specification expressions with the fixpoint constructions

Let $\Sigma = \{\rho_i(X_i) = s_i \mid i \in \Im\}$ be a specification system.

The concept of a fixpoint of Σ is defined below.

An **evaluation** of relational symbols from the set $\{\rho_i \mid i \in \Im\}$ is an $\Im$–indexed set ε of mappings of the following form:

$$\varepsilon \stackrel{\text{def}}{=} \{[\![\rho_i]\!]_\varepsilon : \mathcal{D}_{type(\rho_i)} \to \mathcal{D}_{bool} \mid i \in \Im\}.$$

For every specification expression s_i from the system Σ **an ε–interpretation of s_i** is a mapping $[\![s_i]\!]_\varepsilon$, which is defined in the previous subsection, with the following assumption: for every $i \in \Im$ the relational symbol ρ_i is considered as a functional symbol of the type

$$(type(\rho_i) \to bool),$$

which is associated with the mapping $[\![\rho_i]\!]_\varepsilon$.

The evaluation ε is called **a fixpoint of** Σ, iff for every $i \in \Im$ and every $D = (d_1, \ldots, d_m) \in \mathcal{D}_{type(\rho_i)}$

$$[\![\rho_i]\!]_\varepsilon(d_1, \ldots, d_m) = [\![s_i]\!]_\varepsilon(d_{j_1}, \ldots, d_{j_k}),$$

where the numbers $j_1, \ldots, j_k \in \{1, \ldots, m\}$ are such that if the list X_i has the form

$$X_i = (x_1, \ldots, x_m),$$

then $Var\_List(s_i) = (x_{j_1}, \ldots, x_{j_k})$.

Note that not every specification system has a fixpoint.

If the system Σ has a fixpoint, then for every specification expression of the form $\Sigma_i(S)$, where the list S has the form $(s'_1, \ldots, s'_k)$, we say that **its interpretation $[\![\Sigma_i(S)]\!]$ does exist**, and the mapping $[\![\Sigma_i(S)]\!]$ is equal to the composition

$$[\![\rho_i]\!] \circ ([\![s'_1]\!] \times \ldots \times [\![s'_k]\!]),$$

where the mapping $[\![\rho_i]\!] : \mathcal{D}_{type(\rho_i)} \to \mathcal{D}_{bool}$ maps every list $D \in \mathcal{D}_{type(\rho_i)}$ to the element

- $\top$, if there is a fixpoint ε of Σ, such that $[\![\rho_i]\!]_\varepsilon$ maps D to $\top$,

- $\perp$, otherwise.

If the system Σ has no fixpoints, then we say that an interpretation of the specification expression $\Sigma_i(S)$ does not exist.

Hereafter any specification expression under consideration is assumed to be such that its interpretation does exist.

5.7 Notation

For every $s \in Spec$ and every list D of data values of the type $type(Var\_List(s))$ the symbol θ_D denotes the following substitution operator:

$$\theta_D \stackrel{\text{def}}{=} \Big\{ \ Var\_List(s) := D \ \Big\}.$$

Hereafter the specification expression $\theta_D(s)$ and the element $[\![s]\!](D)$ are considered as the same entities.

5.8 Tautologies

A boolean specification expression s is **a tautology**, iff

$$\forall D \in \mathcal{D}_{type(Var\_List(s))} \quad \theta_D(s) = \top.$$

It is not so difficult to prove that if s is a tautology, then for every substitution operator θ the specification expression $\theta(s)$ also is a tautology.

If s is a tautology, then this fact is denoted by the formula $s = \top$. If a specification expression of the form $s_1 \to s_2$ is a tautology, then this fact is denoted by the formula $s_1 \leq s_2$.

5.9 Models of specification expressions

Let $P = \{p_1, \ldots, p_k\}$ be a finite subset of the set $Inputs \cup Outputs$.

A specification expression associated with P is a boolean specification expression s such that $Var(s) \subseteq P$.

Let s be a specification expression associated with P, and Q be a P–indexed set of queues of the form

$$Q = \{Q_p \in \mathcal{D}_{type(p)} \mid p \in P\}.$$

The set Q is a **model** of s, iff $\theta(s) = \top$, where the substitution operator θ has the form

$$\theta = \left\{ \begin{array}{l} p_1 := Q_{p_1} \\ \dots \\ p_k := Q_{p_k} \end{array} \right\}$$

The symbol $Q \models s$ denotes the statement that Q is a model of s.

If a sequential agent SA is such that

$$\forall Q \in Beh(SA) \quad Q \models s,$$

then this statement is denoted by the symbol $SA \models s$. The symbol $DA \models s$ denotes the analogous statement for a distributed agent DA.

6 Verification of distributed agents

In this section we represent the new approach to the problem of verification of distributed agents. The approach consists of

1. Proving of the correctness of all sequential agents that are constituents of the distributed agent.
2. Proving that the conjunction of specifications of the above sequential agents and the conditions of equality of queues on connected ports implies the specification of the distributed agent.

6.1 The problem of verification of distributed agents

The problem of **verification** of a distributed agent consists of the following: given a distributed agent DA, and a specification expression s associated with $Ports(DA)$, prove that $DA \models s$.

The statement $DA \models s$ can be proven with use of the following theorem.

Theorem 1.
Given

- a distributed agent DA of the form

$$DA \stackrel{\text{def}}{=} (SA^1, \ldots, SA^k; Channels),$$

  where the set $Channels$ has the form

$$Channels = \{(p'_1, p''_1), \ldots, (p'_u, p''_u)\},$$

- and a specification expression s associated with $Ports(DA)$.

Then the statement $DA \models s$ holds, if there is a list $s^1, \ldots, s^k$ of specification expressions associated with
$Ports(SA^1), \ldots, Ports(SA^k)$ respectively, such that

1. $\forall i = 1, \ldots, k \quad SA^i \models s^i$, and
2. $\left\{ \begin{array}{l} s^1 \wedge \ldots \wedge s^k \\ (p'_1 = p''_1) \wedge \ldots \wedge (p'_u = p''_u) \end{array} \right\} \leq s.$

Proof.
Let $Ports(DA) = \{p_1, \ldots, p_m\}$, and Q be a list of the form $Q = (Q^1, \ldots, Q^k)$, where

- for every $i = 1, \ldots, k$

$$Q^i = \{Q^i_p \mid p \in Ports(SA^i)\} \in Beh(SA^i),$$

- for every pair $i, j \in \{1, \ldots, k\}$ and every pair $(p', p'') \in Channels$, such that $p' \in Ports(SA^i)$ and $p'' \in Ports(SA^j)$, the equality $Q^i_{p'} = Q^j_{p''}$ holds,

- there is a trace $tr = (tr^1, \ldots, tr^k)$ of DA associated with Q.

We must prove that $\theta(s) = \top$, where the substitution operator θ has the form

$$\theta = \left\{ \begin{array}{l} p_1 := Q^{i_1}_{p_1} \\ \ldots \\ p_m := Q^{i_m}_{p_m} \end{array} \right\}$$

and the numbers $i_1, \ldots, i_m$ are such that

$$p_1 \in Ports(SA^{i_1}), \ldots, p_m \in Ports(SA^{i_m}).$$

We will use the following notations:

- for every $i \in \{1, \ldots, k\}$ the symbol $List(Ports(SA^i))$ denotes a list of elements of the set $Ports(SA^i)$,
- for every $i \in \{1, \ldots, k\}$ the symbol $List(Q^i)$ denotes a list of elements of the set Q^i, such that if $List(Ports(SA^i)) = (p^i_1, \ldots, p^i_r)$, then $List(Q^i) = (Q^i_{p^i_1}, \ldots, Q^i_{p^i_r})$,
- the symbol θ_Q denotes the substitution operator

$$\theta_Q \stackrel{\text{def}}{=} \left\{ \begin{array}{l} List(Ports(SA^1)) := List(Q^1) \\ \ldots \\ List(Ports(SA^k)) := List(Q^k) \end{array} \right\}$$

The assumption that s associated with $Ports(DA)$, i.e. $Var(s) \subseteq \{p_1, \ldots, p_m\}$, implies the equality $\theta_Q(s) = \theta(s)$.

The assumption

$$\forall i = 1, \ldots, k \quad SA^i \models s^i \quad \text{and} \quad Q^i \in Beh(SA^i),$$

implies that

$$\theta_Q(s^1) = \ldots = \theta_Q(s^k) = \top.$$

The condition that for every pair $i, j \in \{1, \ldots, k\}$ and every pair $(p', p'') \in Channels$, such that $p' \in Ports(SA^i)$
and $p'' \in Ports(SA^j)$, the equality $Q^i_{p'} = Q^j_{p''}$ holds, implies the equalities

$$\theta_Q(p'_1 = p''_1) = \ldots = \theta_Q(p'_u = p''_u) = \top.$$

Thus,

$$\theta_Q(s^1 \wedge \ldots \wedge s^k \wedge (p'_1 = p''_1) \wedge \ldots \wedge (p'_u = p''_u)) = \top,$$

and, since

$$\theta_Q(s^1 \wedge \ldots \wedge s^k \wedge (p'_1 = p''_1) \wedge \ldots \wedge (p'_u = p''_u)) \leq \theta_Q(s),$$

we have:

$$\theta(s) = \theta_Q(s) = \top.$$

■

For every sequential agent SA^i the statement
$SA^i \models s^i$ can be proven using the concept of local correctness presented in subsection 6.5.

6.2 Additional notations

Let $(p_1, \ldots, p_k)$ be a list of variables of queue types. Then

- the symbol $\mathbf{tail}(p_1, \ldots, p_k)$ denotes the list

$$(\mathbf{tail}(p_1), \ldots, \mathbf{tail}(p_k)),$$

- the symbol $\mathbf{head}(p_1, \ldots, p_k) = \omega$ denotes the boolean specification expression

$$\left\{ \begin{array}{l} \mathbf{head}(p_1) = \omega \\ \ldots \\ \mathbf{head}(p_k) = \omega \end{array} \right\}.$$

For every sequential agent SA the symbol $Spec(SA)$ denotes the set

$$\{s \in Spec_{bool} \mid Var(s) \subseteq Var(SA) \sqcup Ports(SA)\}.$$

Let

- SA be a sequential agent,

- the set $Var(SA)$ has the form

$$Var(SA) = \{x_1, \ldots, x_l\},$$

- the set $Ports(SA)$ has the form

$$Ports(SA) = \{p_1, \ldots, p_k\},$$

- Q be a $Ports(SA)$–indexed set of queues:

$$Q = \{Q_p \in \mathcal{D}_{type(p)} \mid p \in Ports(SA)\},$$

- tr be a trace of SA associated with Q:

$$tr = \{(Node(t), Edge(t), Eval(t)) \mid t \geq 1\}.$$

Then for every $t \geq 1$, such that tr has a component with the number t, the symbol $\theta_{tr,t}$ denotes the following substitution operator:

$$\theta_{tr,t} \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} x_1 := x_1(t) & \\ \ldots & \\ x_l := x_l(t) & \\ p_1 := p_1(t) & (\stackrel{\text{def}}{=} \mathbf{tail}^{t-1}(Q_{p_1})) \\ \ldots & \\ p_k := p_k(t) & (\stackrel{\text{def}}{=} \mathbf{tail}^{t-1}(Q_{p_k})) \end{array} \right\},$$

where $\mathbf{tail}^0$ is an identity mapping, and for every $t \geq 1$ $\mathbf{tail}^t \stackrel{\text{def}}{=} \mathbf{tail} \circ \mathbf{tail}^{t-1}$.

Note that $\forall i = 1, \ldots, k$ and $\forall t \geq 1$

$$\begin{array}{l} \mathbf{head}(\theta_{tr,t}(p_i)) = \mathbf{head}(\mathbf{tail}^{t-1}(Q_{p_i})) = Q_{p_i}[t], \\ \mathbf{tail}(\theta_{tr,t}(p_i)) = \theta_{tr,t+1}(p_i). \end{array}$$

6.3 Transformations at edges

Let SA be a sequential agent, and P be a list $(p_1, \ldots, p_k)$ of variables of queue types such that $\{p_1, \ldots, p_k\} = Ports(SA)$.

For every $a \in Edges(SA)$ **a transformation at** a is the pair (φ_a, θ_a), where

- φ_a is a boolean data expression, which is called **a condition at** a, and
- θ_a is a substitution operator, which is called **an action at** a.

The components φ_a and θ_a are defined as follows.

1. If $\langle a \rangle = \mathbf{input}(p_i, x)$, where $i \in \{1, \ldots, k\}$, then
$\varphi_a \stackrel{\text{def}}{=} \left\{ \begin{array}{l} \mathbf{head}(p_i) \neq \omega \\ \mathbf{head}(p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_k) = \omega \end{array} \right\}$,
$\theta_a \stackrel{\text{def}}{=} \left\{ \begin{array}{l} x := \mathbf{head}(p_i) \\ P := \mathbf{tail}(P) \end{array} \right\}$.

2. If $\langle a \rangle = \mathbf{output}(p_i, e)$, where $i \in \{1, \ldots, k\}$, then
$\varphi_a \stackrel{\text{def}}{=} \left\{ \begin{array}{l} \mathbf{head}(p_i) = e \\ \mathbf{head}(p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_k) = \omega \end{array} \right\}$,
$\theta_a \stackrel{\text{def}}{=} \{P := \mathbf{tail}(P)\}$.

3. If $\langle a \rangle = \mathbf{default}$, then
$\varphi_a \stackrel{\text{def}}{=} (\mathbf{head}(P) = \omega)$,
$\theta_a \stackrel{\text{def}}{=} \{P := \mathbf{tail}(P)\}$.

4. If $\langle a \rangle = (X := E)$, then
$\varphi_a \stackrel{\text{def}}{=} (\mathbf{head}(P) = \omega)$,
$\theta_a \stackrel{\text{def}}{=} \left\{ \begin{array}{l} X := E \\ P := \mathbf{tail}(P) \end{array} \right\}$.

5. If $\langle a \rangle = b \in \mathcal{E}_{bool}$, then
$\varphi_a \stackrel{\text{def}}{=} \left\{ \begin{array}{l} b \\ \mathbf{head}(P) = \omega \end{array} \right\}$,
$\theta_a \stackrel{\text{def}}{=} \{P := \mathbf{tail}(P)\}$.

Theorem 2.
Given

- a sequential agent SA,
- a $Ports(SA)$–indexed set Q of queues of the form:

$$Q = \{Q_p \in \mathcal{D}_{type(p)} \mid p \in Ports(SA)\},$$

- a trace tr of SA associated with Q:

$$tr = \{(Node(t), Edge(t), Eval(t)) \mid t \geq 1\}.$$

Then for every $t \geq 1$, such that tr has the components with the numbers t and $t+1$, the following formulas hold:

1. $\theta_{tr,t}(\varphi_a) = \top$,
2. $\forall s \in Spec(SA) \quad \theta_{tr,t+1}(s) = \theta_{tr,t}(\theta_a(s))$,

where $a \stackrel{\text{def}}{=} Edge(t)$.

Proof.
For proving the formula (2) it is necessary and sufficient to prove that $\forall x \in Var(SA) \sqcup Ports(SA)$

$$\theta_{tr,t+1}(x) = \theta_{tr,t}(\theta_a(x)).$$

Note that $\forall p \in Ports(SA) \quad \theta_{tr,t+1}(p) = \theta_{tr,t}(\theta_a(p))$. Indeed,

- $\theta_{tr,t+1}(p) \stackrel{\text{def}}{=} \mathbf{tail}^t(Q_p)$, and
- $\theta_{tr,t}(\theta_a(p)) = \theta_{tr,t}(\mathbf{tail}(p)) = \mathbf{tail}(\theta_{tr,t}(p)) =$
  $= \mathbf{tail}(\mathbf{tail}^{t-1}(Q_p)) = \mathbf{tail}^t(Q_p)$.

So, for completing of the proof of theorem 2 it is sufficient to prove that

1. $\theta_{tr,t}(\varphi_a) = \top$, and
2. $\forall x \in Var(SA) \quad \theta_{tr,t+1}(x) = \theta_{tr,t}(\theta_a(x))$.

Now we prove the formulas (1) and (2) for all possible values of $\langle a \rangle$.
Let $\{p_1, \ldots, p_k\}$ be the set $Ports(SA)$.

1. If $\langle a \rangle = \mathbf{input}(p_i, x)$, where $i \in \{1, \ldots, k\}$, then the definitions of tr, φ_a and θ_a imply that
   (a) $\theta_{tr,t+1}(x) = \mathbf{head}(\theta_{tr,t}(p_i))$,
   (b) $\forall j \in \{1, \ldots, k\} \setminus \{i\} \quad \mathbf{head}(\theta_{tr,t}(p_j)) = \omega$,

(c) $\forall y \in Var(SA) \setminus \{x\} \quad \theta_{tr,t+1}(y) = \theta_{tr,t}(y)$,

(d) $\varphi_a \stackrel{\text{def}}{=} (\mathbf{head}(p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_k) = \omega)$,

(e) $\theta_a(x) = \mathbf{head}(p_i)$,

(f) $\forall y \in Var(SA) \setminus \{x\} \quad \theta_a(y) = y$.

The formula (1) follows from (b) and (d).

The formula (2) follows from (a), (c), (e) and (f):

- $\theta_{tr,t+1}(x) = \mathbf{head}(\theta_{tr,t}(p_i)) = \theta_{tr,t}(\theta_a(x))$,
- $\forall y \in Var(SA) \setminus \{x\}$
  $\theta_{tr,t+1}(y) = \theta_{tr,t}(y) = \theta_{tr,t}(\theta_a(y))$.

2. If $\langle a \rangle = \mathbf{output}(p_i, e)$, then the definitions of tr, φ_a and θ_a imply that

(a) $\mathbf{head}(\theta_{tr,t}(p_i)) = e(t)$,

(b) $\forall j \in \{1, \ldots, k\} \setminus \{i\} \quad \mathbf{head}(\theta_{tr,t}(p_j)) = \omega$,

(c) $\forall x \in Var(SA) \quad x(t+1) = x(t)$,

(d) $\varphi_a \stackrel{\text{def}}{=} \left\{ \begin{array}{l} \mathbf{head}(p_i) = e \\ \mathbf{head}(p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_k) = \omega \end{array} \right\}$

(e) $\forall x \in Var(SA) \quad \theta_a(x) = x$.

The formula (1) follows from (a), (b) and (d).

The formula (2) follows from (c) and (e).

3. If $\langle a \rangle = \mathbf{default}$, then the definitions of tr, φ_a and θ_a imply that

(a) $\forall p \in Ports(SA) \quad \mathbf{head}(\theta_{tr,t}(p)) = \omega$,

(b) $\forall x \in Var(SA) \quad x(t+1) = x(t)$,

(c) $\varphi_a \stackrel{\text{def}}{=} (\mathbf{head}(P) = \omega)$

(d) $\forall x \in Var(SA) \quad \theta_a(x) = x$.

The formula (1) follows from (a) and (c).

The formula (2) follows from (b) and (d).

4. If $\langle a \rangle = (X := E)$, where $X = (x_1, \ldots, x_m)$ and $E = (e_1, \ldots, e_m)$, then the definitions of tr, φ_a and θ_a imply that

   (a) $\forall p \in Ports(SA) \quad \mathbf{head}(\theta_{tr,t}(p)) = \omega$,

   (b) $x_1(t+1) = e_1(t), \ \ldots, \ x_m(t+1) = e_m(t)$,

   (c) $\forall y \in Var(SA) \setminus \{x_1, \ldots, x_m\} \quad y(t+1) = y(t)$,

   (d) $\varphi_a \stackrel{\text{def}}{=} (\mathbf{head}(P) = \omega)$,

   (e) $\theta_a(x_1) \stackrel{\text{def}}{=} e_1, \ldots, \theta_a(x_m) \stackrel{\text{def}}{=} e_m$,

   (f) $\forall y \in Var(SA) \setminus \{x_1, \ldots, x_m\} \quad \theta_a(y) \stackrel{\text{def}}{=} y$.

   The formula (1) follows from (a) and (d).

   The formula (2) follows from (b), (c), (e) and (f).

5. If $\langle a \rangle = b \in \mathcal{E}_{bool}$, then the definitions of tr, φ_a and θ_a imply that

   (a) $\forall p \in Ports(SA) \quad \mathbf{head}(\theta_{tr,t}(p)) = \omega$,

   (b) $b(t) = \top$,

   (c) $\forall x \in Var(SA) \quad x(t+1) = x(t)$,

   (d) $\varphi_a \stackrel{\text{def}}{=} \left\{ \begin{array}{l} b \\ \mathbf{head}(P) = \omega \end{array} \right\}$

   (e) $\forall x \in Var(SA) \quad \theta_a(x) = x$.

   The formula (1) follows from (a), (b) and (d).

   The formula (2) follows from (c) and (e).

■

6.4 Asymptotic specifications

Let SA be a sequential agent, and $n \in Nodes(SA)$.

We say that **the node n has infinite index**, if there is an infinite path A in SA, such that $start(A) = n$.

Let $Nodes^{\infty}(SA)$ be a set of all nodes from the set $Nodes(SA)$, which have infinite index.

For every $n \in Nodes^{\infty}(SA)$ the symbol $Edges^{\infty}(n)$ denotes the set of all edges outgoing from the node n, ends on which belong to the set $Nodes^{\infty}(SA)$.

The symbol $\Sigma^{\infty}(SA)$ denotes the specification system of the form

$$\{\rho_n(X) = s_n \mid n \in Nodes^{\infty}(SA)\}$$

where X is a list of all variables from $Var(SA) \sqcup Ports(SA)$, and for every $n \in Nodes^{\infty}(SA)$ the specification expression s_n is defined as follows: let

- $Edges^{\infty}(n) = \{a_1, \ldots, a_r\}$,
- $n_1 \stackrel{\text{def}}{=} end(a_1); \ldots ; n_r \stackrel{\text{def}}{=} end(a_r)$,

then $s_n \stackrel{\text{def}}{=} \left[\begin{array}{l} \varphi_{a_1} \wedge \theta_{a_1}(\rho_{n_1}(X)) \\ \ldots \\ \varphi_{a_r} \wedge \theta_{a_r}(\rho_{n_r}(X)) \end{array}\right]$.

For every $n \in Nodes^{\infty}(SA)$ the specification expression $\Sigma^{\infty}(SA)_n(X)$ is denoted by the symbol $Asymp_n$, and is called **an asymptotic specification** of the node n.

Theorem 3.

Given a sequential agent SA, and an infinite trace tr of SA:

$$tr = \{(Node(t), Edge(t), Eval(t)) \mid t \geq 1\}.$$

Then for every $n \in Nodes^{\infty}(SA)$ and every number $t \geq 1$ such that $Node(t) = n$, the following formula holds:

$$\theta_{tr,t}(Asymp_n) = \top.$$

Proof.

Let $\Sigma^{\infty}(SA) \stackrel{\text{def}}{=} \{\rho_n(X) = s_n \mid n \in Nodes^{\infty}(SA)\}$.

For every $n \in Nodes^{\infty}(SA)$ and every number $k \geq 0$ define the subset $False_k(n)$ of the set $\mathcal{D}_{type(\rho_n)}$ by induction.

Let $n \in Nodes^{\infty}(SA)$, and

- the set $Edges^{\infty}(n)$ has the form

$$Edges^{\infty}(n) = \{a_1, \ldots, a_r\},$$

- $n_1 \stackrel{\text{def}}{=} end(a_1), \ldots, n_r \stackrel{\text{def}}{=} end(a_r)$.

1. The set $False_0(n)$ consists of all lists $D \in \mathcal{D}_{type(\rho_n)}$ such that
$$\theta_D(\varphi_{a_1}) = \ldots = \theta_D(\varphi_{a_r}) = \bot$$
(see subsection 5.7 for the definition of the substitution operator θ_D).

2. For every $k \geq 0$ the set $False_{k+1}(n)$ consists of all lists $D \in \mathcal{D}_{type(\rho_n)}$ such that
   - either $D \in False_k(n)$,
   - or for every $i \in \{1, \ldots, r\}$ the following implication holds:
$$\theta_D(\varphi_{a_i}) = \top \quad \Rightarrow \quad \theta_D(\theta_{a_i}(X)) \in False_k(n_i).$$

Define an evaluation
$$\varepsilon \stackrel{\text{def}}{=} \{[\![\rho_n]\!]_\varepsilon : \mathcal{D}_{type(\rho_n)} \to \mathcal{D}_{bool} \mid n \in Nodes^\infty(SA)\}.$$

For every list $D \in \mathcal{D}_{type(\rho_n)}$
$$[\![\rho_n]\!]_\varepsilon(D) \stackrel{\text{def}}{=} \begin{cases} \bot, & \text{if } \exists k \geq 0 : \quad D \in False_k(n), \\ \top, & \text{if } \forall k \geq 0 : \quad D \notin False_k(n). \end{cases}$$

Now we prove that the evaluation ε is a fixpoint of $\Sigma^\infty(SA)$, i.e. for every $n \in Nodes^\infty(SA)$ and for every $D \in \mathcal{D}_{type(\rho_n)}$
$$[\![\rho_n]\!]_\varepsilon(D) = [\![s_n]\!]_\varepsilon(D).$$

Let s_n has the form $\left[\begin{array}{l} \varphi_{a_1} \wedge \theta_{a_1}(\rho_{n_1}(X)) \\ \ldots \\ \varphi_{a_r} \wedge \theta_{a_r}(\rho_{n_r}(X)) \end{array}\right]$.

Therefore
$$[\![s_n]\!]_\varepsilon(D) = \left[\begin{array}{l} \theta_D(\varphi_{a_1}) \wedge [\![\rho_{n_1}]\!]_\varepsilon(\theta_D(\theta_{a_1}(X))) \\ \ldots \\ \theta_D(\varphi_{a_r}) \wedge [\![\rho_{n_r}]\!]_\varepsilon(\theta_D(\theta_{a_r}(X))) \end{array}\right].$$

By definition, the formula $[\![\rho_n]\!]_\varepsilon(D) = \bot$ holds iff for every $i = 1, \ldots, r$
$$\theta_D(\varphi_{a_i}) = \top \quad \Rightarrow \quad [\![\rho_{n_i}]\!]_\varepsilon(\theta_D(\theta_{a_i}(X))) = \bot.$$
This condition is equivalent to the formula $[\![s_n]\!]_\varepsilon(D) = \bot$.

Thus, ε is a fixpoint of Σ.

The definition of ε implies that for

- every fixpoint ε' of Σ,
- every node $n \in Nodes^{\infty}(SA)$, and
- every list $D \in \mathcal{D}_{type(\rho_n)}$

the following implication holds:

$$[\![\rho_n]\!]_{\varepsilon}(D) = \bot \quad \Rightarrow \quad [\![\rho_n]\!]_{\varepsilon'}(D) = \bot.$$

Consequently, for every $n \in Nodes^{\infty}(SA)$

$$[\![Asymp_n]\!] = [\![\rho_n]\!]_{\varepsilon}.$$

Let a number $t \geq 1$ is such that $Node(t) = n$. We now prove that $\theta_{tr,t}(Asymp_n) = \top$.

Since $[\![Asymp_n]\!] = [\![\rho_n]\!]_{\varepsilon}$, the following equality holds:

$$\theta_{tr,t}(Asymp_n) = [\![\rho_n]\!]_{\varepsilon}(\theta_{tr,t}(X)).$$

Since the trace tr is infinite, then $n \in Nodes^{\infty}(SA)$ and $a \stackrel{\text{def}}{=} Edge(t) \in Edges^{\infty}(n)$. Let $n' \stackrel{\text{def}}{=} end(a)$.

According to theorem 2, the following formulas hold:

1. $\theta_{tr,t}(\varphi_a) = \top$,
2. $\forall s \in Spec(SA) \quad \theta_{tr,t+1}(s) = \theta_{tr,t}(\theta_a(s))$.

Consequently,

$$\begin{array}{ll} & \theta_{tr,t}(Asymp_n) \\ = & [\![\rho_n]\!]_{\varepsilon}(\theta_{tr,t}(X)) \\ = & [\![s_n]\!]_{\varepsilon}(\theta_{tr,t}(X)) \\ = & \left[\begin{array}{l} \theta_{tr,t}(\varphi_{a_1}) \wedge [\![\rho_{n_1}]\!]_{\varepsilon}(\theta_{tr,t}(\theta_{a_1}(X))) \\ \ldots \\ \theta_{tr,t}(\varphi_{a_r}) \wedge [\![\rho_{n_r}]\!]_{\varepsilon}(\theta_{tr,t}(\theta_{a_r}(X))) \end{array}\right] \\ \geq & [\![\rho_{n'}]\!]_{\varepsilon}(\theta_{tr,t}(\theta_a(X))) \\ = & [\![\theta_{tr,t}(\theta_a(\rho_{n'}(X)))]\!]_{\varepsilon} \\ = & [\![\theta_{tr,t+1}(\rho_{n'}(X))]\!]_{\varepsilon} \\ = & \theta_{tr,t+1}(Asymp_{n'}). \end{array}$$

The above formulas and inductive reasonings imply that for every $k \geq 0$

$$\theta_{tr,t}(X) \notin False_k(n).$$

Consequently, $\theta_{tr,t}(Asymp_n) = [\![\rho_n]\!]_\varepsilon(\theta_{tr,t}(X)) = \top$.
■

6.5 Local correctness

Let

- SA be a sequential agent,
- P be a list $(p_1, \dots, p_k)$ of variables of queue types such that $\{p_1, \dots, p_k\} = Ports(SA)$,
- $C = \{c_n \mid n \in Nodes(SA)\}$ be a $Nodes(SA)$–indexed set of data expressions from $Spec(SA)$,
- $G = \{g_n \mid n \in Nodes(SA)\}$ be a $Nodes(SA)$–indexed sets of specification expressions from $Spec(SA)$.

We say that SA is **locally correct with respect to the pair** (C, G), iff for every $a \in Edges(SA)$ the following formulas hold:

$$\begin{array}{rcl} \varphi_a \wedge c_a & \leq & \theta_a(c'_a) \\ \theta_a(g'_a) \wedge \varphi_a \wedge c_a & \leq & g_a \end{array}$$

where

$$c_a \stackrel{\text{def}}{=} c_{start(a)},\ c'_a \stackrel{\text{def}}{=} c_{end(a)},$$
$$g_a \stackrel{\text{def}}{=} g_{start(a)},\ g'_a \stackrel{\text{def}}{=} g_{end(a)}.$$

For every $n \in Nodes(SA)$

- c_n is called a **safety assertion** at the node n,
- g_n is called a **liveness assertion** at the node n.

The meaning of safety assertions and liveness assertions is explained below.

The sequential agent SA can be interpreted as a deterministic dynamical system with discrete time. The process of functioning of this system consists of transitions from a node to another node, with an execution of actions which are labels of traversed edges. Every transition is a triple

$$(n, n', a) \in Nodes(SA) \times Nodes(SA) \times Edges(SA),$$

such that $start(a) = n, \quad end(a) = n'$. A record of traversed nodes, traversed edges, and current values of variables of SA at every moment of its functioning forms a trace of SA:

$$tr = \{(Node(t), Edge(t), Eval(t)) \mid t \geq 1\},$$

for every number $t > 1$ the component

$$(Node(t), Edge(t), Eval(t))$$

is a record of

1. current node $Node(t)$ at the moment t,
2. current edge $Edge(t)$ at the moment t, and
3. current values of variables $Eval(t)$ at the moment t.

For every $n \in Nodes(SA)$ the safety assertion c_n is some logical condition on the current values of variables of SA for every $t \geq 1$ such that $Node(t) = n$.

The inequality $\varphi_a \wedge c_a \leq \theta_a(c'_a)$ can be interpreted as the following statement: for every number $t \geq 1$, such that

$$Edge(t) = a,$$

if the current values of variables at the moment t satisfy the safety assertion c_a, then after the execution the action $\langle a \rangle$ the values of variables at the next moment of time will satisfy the assertion c'_a.

For every $n \in Nodes(SA)$ the liveness assertion g_n expresses properties of a subtrace

$$tr_k = \{(Node(t), Edge(t), Eval(t)) \mid t \geq k\}$$

of any trace

$$tr = \{(Node(t), Edge(t), Eval(t)) \mid t \geq 1\}$$

of SA, where k is any number such that $Node(k) = n$.

The inequality $\theta_a(g'_a) \wedge \varphi_a \wedge c_a \leq g_a$ can be interpreted as the following statement. Let $t \geq 1$ be a number such that

- $Edge(t) = a$,
- the values of variables at the moment t satisfy the safety assertion c_a.

Then we can prove that the functioning of SA starting from the moment t with the initial evaluation $Eval(t)$ has the properties which are expressed by the specification expression g_a, if we can prove that after the execution of the action $\langle a \rangle$ the functioning of SA (which is started from the node $end(a)$) has the properties which are expressed by the specification expression g'_a.

6.6 Verification of sequential agents

In this subsection we prove the main theorem about verification of sequential agents.

Lemma.

Given

- a sequential agent SA,
- a $Nodes(SA)$–indexed set

  $$C = \{c_n \mid n \in Nodes(SA)\}$$

  of data expressions from $Spec(SA)$, and a $Nodes(SA)$–indexed set

  $$G = \{g_n \mid n \in Nodes(SA)\}$$

  of specification expressions from $Spec(SA)$, such that SA is locally correct with respect to the pair (C, G),
- a $Ports(SA)$–indexed set Q of queues of the form

  $$Q = \{Q_p \in \mathcal{D}_{type(p)} \mid p \in Ports(SA)\},$$

- a trace tr of SA associated with Q of the form

$$tr = \{(Node(t), Edge(t), Eval(t)) \mid t \geq 1\}.$$

Then for every $t \geq 1$, such that tr has the components with the numbers t and $t+1$, the following formulas hold:

$$\begin{array}{rcl} \theta_{tr,t}(c_a) & \leq & \theta_{tr,t+1}(c'_a) \\ \theta_{tr,t+1}(g'_a) \wedge \theta_{tr,t}(c_a) & \leq & \theta_{tr,t}(g_a) \end{array}$$

where $a \stackrel{\text{def}}{=} Edge(t)$, and the specification expressions c_a, c'_a, g_a and g'_a are defined at the beginning of subsection 6.5.

Proof.
Since

- $\varphi_a \wedge c_a \leq \theta_a(c'_a)$, and
- $\theta_a(g'_a) \wedge \varphi_a \wedge c_a \leq g_a$,

the following formulas hold:

$$\begin{array}{rcl} \theta_{tr,t}(\varphi_a) \wedge \theta_{tr,t}(c_a) & \leq & \theta_{tr,t}(\theta_a(c'_a)), \\ \theta_{tr,t}(\theta_a(g'_a)) \wedge \theta_{tr,t}(\varphi_a) \wedge \theta_{tr,t}(c_a) & \leq & \theta_{tr,t}(g_a). \end{array}$$

These inequalities can be transformed to the desired inequalities by using the results of theorem 2.

■

Theorem 4.
Given a sequential agent SA, and a specification expression s associated with $Ports(SA)$.
Then the statement $SA \models s$ holds, if there are

- a $Nodes(SA)$–indexed set

$$C = \{c_n \mid n \in Nodes(SA)\}$$

of data expressions from $Spec(SA)$, and

- a $Nodes(SA)$–indexed set

$$G = \{g_n \mid n \in Nodes(SA)\}$$

  of specification expressions from $Spec(SA)$,

such that

1. SA is locally correct with respect to (C, G),
2. $c_{root(SA)} = \top, \quad g_{root(SA)} = s$,
3. for every terminal node $n \in Nodes(SA)$

$$c_n \leq g_n,$$

4. there is a subset $N \subseteq Nodes^{\infty}(SA)$ such that
   - $\forall n \in N \quad c_n \wedge Asymp_n \leq g_n$,
   - for every infinite trace of SA

$$tr = \{(Node(t), Edge(t), Eval(t)) \mid t \geq 1\}$$

     there is a number t such that $Node(t) \in N$.

Proof.
Let the set $Ports(SA)$ has the form

$$Ports(SA) = \{p_1, \ldots, p_k\},$$

and Q be a set of queues of the form

$$Q = \{Q_{p_i} \in \mathcal{D}_{type(p_i)} \mid i = 1, \ldots, k\}$$

such that there is a trace tr of SA associated with Q:

$$tr = \{(Node(t), Edge(t), Eval(t)) \mid t \geq 1\}.$$

We must prove that $\theta(s) = \top$, where

$$\theta = \left\{ \begin{array}{l} p_1 := Q_{p_1} \\ \ldots \\ p_k := Q_{p_k} \end{array} \right\}.$$

According to the lemma in this subsection, for every $t \geq 1$, such that tr has the components with the numbers t and $t+1$, the following formulas hold:

$$\begin{array}{rcl} \theta_{tr,t}(c_a) & \leq & \theta_{tr,t+1}(c'_a) \\ \theta_{tr,t+1}(g'_a) \wedge \theta_{tr,t}(c_a) & \leq & \theta_{tr,t}(g_a) \end{array}$$

where $a \stackrel{\text{def}}{=} Edge(t)$, the specification expressions c_a, c'_a, g_a and g'_a are defined in subsection 6.5, and the substitution operators $\theta_{tr,t} \quad (t \geq 1)$ are defined in subsection 6.2. These formulas imply that for every $t \geq 1$, such that tr has the component with the number t, the following formulas hold:

$$\begin{array}{l} \theta_{tr,1}(c_1) \leq \theta_{tr,2}(c_2) \leq \ldots \leq \theta_{tr,t}(c_t) \\ \theta_{tr,t}(g_t) \wedge \theta_{tr,1}(c_1) \leq \theta_{tr,1}(g_1), \end{array}$$

where for every $i = 1, \ldots, t$

$$c_i \stackrel{\text{def}}{=} c_{Node(i)}, \quad g_i \stackrel{\text{def}}{=} g_{Node(i)}.$$

The equality $c_1 = \top$ implies that

- $\theta_{tr,1}(c_1) = \theta_{tr,2}(c_2) = \ldots = \theta_{tr,t}(c_t) = \top$, and
- $\theta_{tr,t}(g_t) \leq \theta_{tr,1}(g_1)$.

By assumption, $g_1 = s$, and $Var(s) \subseteq Ports(SA)$. Thus, $\theta_{tr,1}(g_1) = \theta_{tr,1}(s) = \theta(s)$, and $\theta_{tr,t}(g_t) \leq \theta(s)$.

If the trace tr is finite and consists of t components, then by assumption $c_t \leq g_t$, and consequently

$$\theta_{tr,t}(c_t) \leq \theta_{tr,t}(g_t) \leq \theta(s).$$

Since $\theta_{tr,t}(c_t) = \top$, we have: $\theta(s) = \top$.

Assume that the trace tr is infinite.

By assumption, in this case there is a number t such that $Node(t) = n \in N$.

According to theorem 3, $\theta_{tr,t}(Asymp_n) = \top$.

Since $c_n \wedge Asymp_n \leq g_n$ is given, we obtain:

$$\theta_{tr,t}(c_n) \wedge \theta_{tr,t}(Asymp_n) \leq \theta_{tr,t}(g_n).$$

Since $\theta_{tr,t}(c_n) = \top$ and $\theta_{tr,t}(Asymp_n) = \top$, we have:

$$\theta_{tr,t}(g_n) = \top.$$

As it was stated before, $\theta_{tr,t}(g_n) \leq \theta(s)$.
Consequently, $\theta(s) = \top$.
■

One of difficulties in practical use of theorem 4 is the problem of checking the condition $c_n \wedge Asymp_n \leq g_n$. In subsection 6.8 we prove a theorem which can be used for checking this condition.

6.7 Transformation graphs

Let $X = (x_1, \ldots, x_m) \in \mathcal{X}^*$ be a list of distinct variables.

A **transformation over** X is a pair (φ, θ), where

- φ is a boolean data expression such that

$$Var(\varphi) \subseteq \{x_1, \ldots, x_m\},$$

- θ is a substitution operator of the form (X, S), where $S = (s_1, \ldots, s_m)$ is such that for every $i = 1, \ldots, m$

$$s_i \in \mathcal{E}, \quad Var(s_i) \subseteq \{x_1, \ldots, x_m\}.$$

A **transformation graph over** X is a rooted labelled graph TG, every edge of which is labelled by some transformation over X.

The symbols $Nodes(TG)$, $root(TG)$ and $Edges(TG)$ denote the set of nodes of TG, the root of TG, and the set of edges of TG, respectively.

For every edge $a \in Edges(TG)$

- the symbol $\langle a \rangle$ denotes its label,

- the symbols φ_a and θ_a denote the first and the second components of $\langle a \rangle$ respectively.

  Thus, $\langle a \rangle \stackrel{\text{def}}{=} (\varphi_a, \theta_a)$.

A node $n \in Nodes(TG)$ is said to be **terminal**, iff the set of outgoing edges from n is empty. Any terminal node of TG is denoted by "ω".

The concept of a **path** in TG is defined similar to the one defined for sequential agents in subsection 3.3.

For every finite path A in TG a **transformation at** A is a pair (φ_A, θ_A), which is defined as follows:

- if A has the form (a), where $a \in Edges(TG)$, then $\varphi_A \stackrel{\text{def}}{=} \varphi_a$ and $\theta_A \stackrel{\text{def}}{=} \theta_a$,
- if A has the form $(a_1, \ldots, a_k)$, where $k \geq 2$, and $B \stackrel{\text{def}}{=} (a_2, \ldots, a_k)$, then
  - $\varphi_A \stackrel{\text{def}}{=} \varphi_{a_1} \wedge \theta_{a_1}(\varphi_B)$,
  - $\theta_A(X) \stackrel{\text{def}}{=} \theta_{a_1}(\theta_B(X))$.

Let TG and TG' be transformation graphs over X.

A **morphism** from TG to TG' is a binary relation μ from $Nodes(TG)$ to $Nodes(TG')$:

$$\mu \subseteq Nodes(TG) \times Nodes(TG')$$

such that

1. $(root(TG), root(TG')) \in \mu$,
2. let n_1, n_2 be nodes from $Nodes(TG)$ and n'_1 be a node from $Nodes(TG')$ such that
   - $(n_1, n'_1) \in \mu$,
   - there is a path A from n_1 to n_2,

   then there is a node $n'_2 \in Nodes(TG')$ such that

   (a) $(n_2, n'_2) \in \mu$,

   (b) there is a path A' in TG' from n'_1 to n'_2, such that
   - $\varphi_A \leq \varphi_{A'}$,
   - for every $D \in \mathcal{D}_{type(X)}$ the following implication holds:

$$\theta_D(\varphi_A) = \top \;\; \Rightarrow \;\; \theta_D(\theta_A(X)) = \theta_D(\theta_{A'}(X)).$$

The symbols $node(\mu, n_1, n_2, n_1', A)$ and $path(\mu, n_1, n_2, n_1', A)$ denote the node n_2' and the path A' which have the above properties.

The formula $\mu : TG \to TG'$ denotes the fact that μ is a morphism from TG to TG'.

Let

- TG be a transformation graph over X,
- n be a node from $Nodes(TG)$, and
- D be a list from $\mathcal{D}_{type(X)}$.

The node n is said to be **open for** D, iff there is an infinite path $A = (a_1, a_2, \ldots)$ in TG outgoing from n, such that for every $k \geq 1$ the prefix $A_k \stackrel{\text{def}}{=} (a_1, \ldots, a_k)$ of A satisfies the following condition: $\theta_D(\varphi_{A_k}) = \top$.

The definition of a morphism of transformation graphs implies that if

1. TG and TG' are transformation graphs over X,
2. $n \in Nodes(TG)$ and $n' \in Nodes(TG')$,
3. there is a morphism $\mu : TG \to TG'$, such that $(n, n') \in \mu$,
4. $D \in \mathcal{D}_{type(X)}$,
5. the node n is open for D,

then the node n' is open for D.

Let TG be a transformation graph, and (φ, θ) be a transformation over X.

The symbol $(\varphi, \theta).TG$ denotes a transformation graph, which is defined as follows:

- $Nodes((\varphi, \theta).TG) \stackrel{\text{def}}{=} Nodes(TG) \sqcup \{n\}$, where n is a new node,
- $root((\varphi, \theta).TG) \stackrel{\text{def}}{=} n$,
- The set $Edges((\varphi, \theta).TG)$ consists of all edges from the set $Edges(TG)$ and of a new edge from n to $root(TG)$, label of which is (φ, θ).

6.8 Linear specification systems

A specification system $\Sigma = \{\rho_i(X_i) = s_i \mid i \in \Im\}$ is said to be **linear**, iff every formal equation of Σ has the form

$$\rho_i(X_i) = \left[\begin{array}{l} \varphi_0 \\ \varphi_1 \wedge \rho_{j_1}(S_1) \\ \dots \\ \varphi_u \wedge \rho_{j_u}(S_u) \end{array}\right]$$

where

- $\varphi_0, \varphi_1, \dots, \varphi_u$ are boolean data expressions,
- $\rho_{j_1}, \dots, \rho_{j_u}$ are relational symbols from the set $\{\rho_i \mid i \in \Im\}$,
- $S_1, \dots, S_u$ are lists of data expressions such that

$$\forall k = 1, \dots, u \quad type(S_k) = type(\rho_{j_k}).$$

Let s be a specification expression of the form $\Sigma_i(S)$, where Σ is a linear system of the form

$$\Sigma = \{\rho_i(X) = s_i \mid i \in \Im\}.$$

A **transformation graph** $TG(s)$ **over** X **associated with** s is defined as follows:

- $Nodes(TG(s)) \stackrel{\text{def}}{=} \{root\} \sqcup \{\rho_i \mid i \in \Im\} \sqcup \{\omega\}$,
- $root(TG(s)) \stackrel{\text{def}}{=} root$,
- the graph $TG(s)$ contains an edge with the label $(\top, \theta)$ from *root* to ρ_i, where θ is the substitution operator (X, S),
- for every formal equation of the form given above from Σ
  - the graph $TG(s)$ contains an edge with the label (φ_0, θ_X) from ρ_i to ω, where $\theta_X \stackrel{\text{def}}{=} (X, X)$,

– for every $k = 1, \ldots, u$ the graph $TG(s)$ contains an edge with the label (φ_k, θ_k) from ρ_i to ρ_{j_k}, where θ_k is the substitution operator (X, S_k).

Theorem 5.
Given

- a sequential agent SA,
- a set H of specification expressions from $Spec(SA)$ of the form

$$H = \{h_n \mid n \in Nodes^\infty(SA)\}$$

such that for every $n \in Nodes^\infty(SA)$

– the expression h_n has the form $\Sigma_i^n(S^n)$, where Σ^n is a linear system,

– for every $a \in Edges^\infty(n)$ there is a morphism $\mu_a : (\varphi_a, \theta_a).TG(h_{end(a)}) \to TG(h_n)$.

Then for every $n \in Nodes^\infty(SA) \quad Asymp(n) \leq h_n$.

Proof.
The condition $Asymp_n \leq h_n$ is equivalent to the following statement: for every $D \in \mathcal{D}_{type(X)}$

$$[\![Asymp_n]\!](D) = \top \quad \Rightarrow \quad \theta_D(h_n) = \top.$$

The formula $[\![Asymp_n]\!](D) = \top$ is equivalent to the following statement: there is an infinite path
$A = (a_1, a_2, \ldots)$ in SA outgoing from n, such that for every $k \geq 1$ the prefix $A_k \stackrel{\text{def}}{=} (a_1, \ldots, a_k)$ of A satisfies the condition

$$\theta_D(\varphi_{A_k}) = \top.$$

We now prove that $root(TG(h_n))$ is open for D, i.e. there is a path $B = (b_1, b_2, \ldots)$ in $TG(h_n)$ outgoing from $root(TG(h_n))$, such that for every $k \geq 1$ the prefix

$$B_k \stackrel{\text{def}}{=} (b_1, \ldots, b_k)$$

of B satisfies the equality

$$\theta_D(\varphi_{B_k}) = \top.$$

Let $n_0 \stackrel{\text{def}}{=} n$, $G_0 \stackrel{\text{def}}{=} TG(h_n)$, and for every $i \geq 1$

$$n_i \stackrel{\text{def}}{=} end(a_i), \quad G_i \stackrel{\text{def}}{=} (\varphi_{a_i}, \theta_{a_i}).TG(h_{n_i}).$$

By assumption, for every $i \geq 1$ there is a morphism

$$\mu_i : G_i \to TG(h_{n_{i-1}}).$$

This implies that for every $i \geq 0$ there exist an infinite sequence

$$(n_i^0, n_i^1, \ldots)$$

of nodes of G_i and an infinite sequence

$$(B_i^0, B_i^1, \ldots)$$

of finite paths in G_i such that

- $\forall i \geq 0 \quad n_i^0 = root(G_i) = start(B_i^0)$,
- $\forall i \geq 1 \quad n_i^1 = root(TG(h_{n_i}))$,
- $\forall i \geq 0 \quad \forall j \geq 1$
$$n_i^j = end(B_i^{j-1}) = start(B_i^j),$$
- $\forall i \geq 0 \quad \forall j \geq 1$
$$n_i^{j+1} = node(\mu_{i+1}, n_{i+1}^{j-1}, n_{i+1}^j, n_i^j, B_{i+1}^{j-1}),$$
$$B_i^j = path(\mu_{i+1}, n_{i+1}^{j-1}, n_{i+1}^j, n_i^j, B_{i+1}^{j-1}).$$

The required path B is defined as the concatenation

$$B \stackrel{\text{def}}{=} B_0^0 \cdot B_0^1 \cdot B_0^2 \cdot \ldots$$

The statement that $root(TG(h_n))$ is open for D and the definition of h_n imply the required equality:

$$\theta_D(h_n) = \top.$$

■

This theorem can be used for checking the conditions

$$c_n \wedge Asymp_n \leq g_n$$

in theorem 4 (which are equivalent to the conditions $Asymp_n \leq c_n \to g_n$) in the case when for every $n \in Nodes^{\infty}(SA)$ the specification g_n has the form

$$g_n = \left[\begin{array}{l} \zeta_0 \\ \zeta_1 \wedge \Sigma^1_{i_1}(S_1) \\ \ldots \\ \zeta_m \wedge \Sigma^m_{i_m}(S_m) \end{array}\right]$$

where

- $\zeta_0, \zeta_1, \ldots, \zeta_m$ are boolean data expressions,
- $\Sigma^1, \ldots, \Sigma^m$ are linear specification systems.

In this case the specification expression $c_n \to g_n$ has the same interpretation as a specification expression of the form $\Sigma_i(S)$, where Σ is linear.

6.9 Simplified condition of local correctness

The construction of the sets C, G of safety assertions and liveness assertions for verification of a given sequential agent SA can be very difficult and non-trivial procedure.

In this subsection we prove a theorem (theorem 6) which can be used for simplification of construction of the sets C and G.

This theorem claims that it is sufficient to define safety assertions and liveness assertions not for all nodes of SA, but only for some of them.

For the formulation of this theorem it is necessary the following definition of a transformation at a path of SA.

Let $A = (a_1, \ldots, a_m)$ be a finite path in SA.

A **transformation at** A is the pair (φ_A, θ_A), where

- φ_A is a boolean data expression, which is called **a condition at** A, and

- θ_A is a substitution operator, which is called **an action at** A, and has the form
$$\theta_A \stackrel{\text{def}}{=} \left\{ \begin{array}{l} x_1 := s_1 \\ \dots \\ x_l := s_l \\ P := \mathbf{tail}^m(P) \end{array} \right\}$$
where $\{x_1, \dots, x_l\} = Var(SA)$, P is a list of elements of the set $Ports(SA)$, and m is a number of edges in the path A.

The components φ_A and θ_A are defined as follows.

- If $m = 1$, i.e. $A = (a)$, and θ_a has the form
$$\theta_a \stackrel{\text{def}}{=} \left\{ \begin{array}{l} x_{i_1} := s'_1 \\ \dots \\ x_{i_h} := s'_h \\ P := \mathbf{tail}(P) \end{array} \right\}$$
where $\{i_1, \dots, i_h\} \subseteq \{1, \dots, l\}$, then $\varphi_A \stackrel{\text{def}}{=} \varphi_a$, and
$$\theta_A \stackrel{\text{def}}{=} \left\{ \begin{array}{l} x_1 := s_1 \\ \dots \\ x_l := s_l \\ P := \mathbf{tail}(P) \end{array} \right\}$$
where $\forall j = 1, \dots, l$
$$s_j \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} x_j, & \text{if } j \not\in \{i_1, \dots, i_h\}, \\ s'_g, & \text{if } j = i_g \text{ for some } g \in \{1, \dots, h\}. \end{array} \right.$$

- Let
  - A has the form $(a_1, \dots, a_m)$, where $m > 1$,
  - B be the path $(a_2, \dots, a_m)$.

  Then

  - $\varphi_A \stackrel{\text{def}}{=} \left\{ \begin{array}{l} \varphi_{a_1} \\ \theta_{a_1}(\varphi_B) \end{array} \right\}$,

$$- \ \theta_A = \left\{ \begin{array}{l} x_1 := \theta_{a_1}(\theta_B(x_1)) \\ \dots \\ x_l := \theta_{a_1}(\theta_B(x_l)) \\ P := \mathbf{tail}^m(P) \end{array} \right\}.$$

Lemma.

Given

- a sequential agent SA,
- a path $A = (a_1, \dots, a_m)$ in SA such that $m > 1$,
- specification expressions c, g, c', g' from $Spec(SA)$ such that

$$\varphi_A \wedge c \leq \theta_A(c') \quad \text{and} \quad \theta_A(g') \wedge \varphi_A \wedge c \leq g.$$

Let

- B be the path $(a_2, \dots, a_m)$,
- $c_1 \stackrel{\text{def}}{=} \varphi_B \to \theta_B(c'), \quad g_1 \stackrel{\text{def}}{=} \theta_B(g') \wedge \varphi_B \wedge \theta_B(c')$.

Then the following inequalities hold:

1. $\varphi_{a_1} \wedge c \leq \theta_{a_1}(c_1)$,
2. $\theta_{a_1}(g_1) \wedge \varphi_{a_1} \wedge c \leq g$,
3. $\varphi_B \wedge c_1 \leq \theta_B(c')$,
4. $\theta_B(g') \wedge \varphi_B \wedge c_1 \leq g_1$.

Proof.

The inequalities are proved by using of the definitions of a condition and an action on the path A:

1. the inequality $\varphi_A \wedge c \leq \theta_A(c')$ is equivalent to the inequality

$$\varphi_{a_1} \wedge \theta_{a_1}(\varphi_B) \wedge c \leq \theta_{a_1}(\theta_B(c')),$$

which is equivalent to

$$\varphi_{a_1} \wedge c \leq \theta_{a_1}(\varphi_B) \to \theta_{a_1}(\theta_B(c')),$$

i.e. $\varphi_{a_1} \wedge c \leq \theta_{a_1}(\varphi_B \to \theta_B(c')) = \theta_{a_1}(c_1)$,

2. $\theta_{a_1}(g_1) \wedge \varphi_{a_1} \wedge c$
$= \theta_{a_1}(\theta_B(g') \wedge \varphi_B \wedge \theta_B(c')) \wedge \varphi_{a_1} \wedge c$
$= \left\{ \begin{array}{l} \theta_{a_1}(\theta_B(g')) \\ \theta_{a_1}(\varphi_B) \\ \theta_{a_1}(\theta_B(c')) \\ \varphi_{a_1} \\ c \end{array} \right\} = \left\{ \begin{array}{l} \theta_A(g') \\ \theta_{a_1}(\varphi_B) \\ \theta_A(c') \\ \varphi_{a_1} \\ c \end{array} \right\}$
$= \left\{ \begin{array}{l} \theta_A(g') \\ \varphi_A \\ \theta_A(c') \\ c \end{array} \right\} \leq \left\{ \begin{array}{l} \theta_A(g') \\ \varphi_A \\ c \end{array} \right\} \leq g,$

3. $\varphi_B \wedge c_1 = \varphi_B \wedge (\varphi_B \to \theta_B(c')) \leq \theta_B(c'),$

4. $\theta_B(g') \wedge \varphi_B \wedge c_1$
$= \theta_B(g') \wedge \varphi_B \wedge (\varphi_B \to \theta_B(c'))$
$\leq \theta_B(g') \wedge \varphi_B \wedge \theta_B(c') = g_1.$

■

Theorem 6.
Given

- a sequential agent SA,
- a subset $N \subseteq Nodes(SA)$, such that for every $n \in Nodes(SA) \setminus N$ there is a unique edge $a \in Edges(SA)$ with the property
$$end(a) = n,$$
- a pair C, G of N–indexed sets of specification expressions from $Spec(SA)$ of the form
$$C \stackrel{\text{def}}{=} \{c_n \mid n \in N\}, \quad G \stackrel{\text{def}}{=} \{g_n \mid n \in N\},$$
where $\forall n \in N \quad c_n \in \mathcal{E}$,
- a finite set $Paths$ of paths in SA such that

- for every $a \in Edges(SA)$ there is
  $A \in Paths$, such that $a \in A$,
- let A be a path from the set $Paths$:

$$A = (a_1, \ldots, a_h),$$

  then for every pair $j_1, j_2 \in \{1, \ldots, h\}$

$$j_1 \neq j_2 \quad \Rightarrow \quad end(a_{j_1}) \neq end(a_{j_2}),$$

  and $\forall j \in 1, \ldots, h-1 \quad end(a_j) \notin N$,
- for every $A \in Paths$ the following conditions hold:
  * $start(A) \in N$ and $end(A) \in N$,
  * $\varphi_A \wedge c_A \leq \theta_A(c'_A)$,
    $\theta_A(g'_A) \wedge \varphi_A \wedge c_A \leq g_A$, where

$$\begin{array}{l} c_A \stackrel{\text{def}}{=} c_{start(A)},\ c'_A \stackrel{\text{def}}{=} c_{end(A)}, \\ g_A \stackrel{\text{def}}{=} g_{start(A)},\ g'_A \stackrel{\text{def}}{=} g_{end(A)}. \end{array}$$

Then there is a pair of $Nodes(SA)$–indexed sets

$$\begin{array}{l} \tilde{C} = \{\tilde{c}_n \mid n \in Nodes(SA)\}, \\ \tilde{G} = \{\tilde{g}_n \mid n \in Nodes(SA)\} \end{array}$$

of specification expressions from $Spec(SA)$, such that

1. $\forall n \in Nodes(SA) \quad c_n \in \mathcal{E}$,
2. $\forall n \in N \quad \tilde{c}_n = c_n$ and $\tilde{g}_n = g_n$,
3. SA is locally correct with respect to the pair $(\tilde{C}, \tilde{G})$.

Proof.
We prove this theorem by induction for the cardinality of the set $Nodes(SA) \setminus N$.

If the set $Nodes(SA) \setminus N$ is empty, then the conclusion of the theorem holds: in this case $\tilde{C} = C$ and $\tilde{G} = G$.

Let $N \neq Nodes(SA)$, i.e. $\exists n \in Nodes(SA) \setminus N$.

There is $b \in Edges(SA)$, such that $end(b) = n$.

By assumption, $\exists A \in Paths$ such that $b \in A$.

Let A has the form $(a_1, \ldots, a_h)$. If $end(a_1) \in N$, then $A = (a_1)$, and, consequently, $a_1 = b$, i.e. $end(b) \in N$. This is impossible.

Thus, the edge $a \stackrel{\text{def}}{=} a_1$ has the property:

$$\begin{array}{l} start(a) = start(A) \in N, \\ end(a) \notin N. \end{array}$$

Let

- $\{A_1, \ldots, A_m\}$ be the set of all paths from the set $Paths$, such that the first edge of all of them is the edge a; this set is not empty because it contains A
- $\{B_1, \ldots, B_m\}$ be the set of paths in SA such that for every $i = 1, \ldots, m$ the path A_i is a concatenation of (a) and B_i.

According to the lemma in this subsection, for every $i = 1, \ldots, m$ there is a pair c_i, g_i of specification expressions from the set $Spec(SA)$, such that the following inequalities hold:

1. $\varphi_a \wedge c \leq \theta_a(c_i)$,
2. $\theta_a(g_i) \wedge \varphi_a \wedge c \leq g$, which is equivalent to the inequality
$$\theta_a(g_i) \leq (\varphi_a \wedge c) \to g,$$
3. $\varphi_{B_i} \wedge c_i \leq \theta_{B_i}(c'_{B_i})$,
4. $\theta_{B_i}(g'_{B_i}) \wedge \varphi_{B_i} \wedge c_i \leq g_i$,

where

$$\begin{array}{l} c \stackrel{\text{def}}{=} c_{start(a)}, \ c'_{B_i} \stackrel{\text{def}}{=} c_{end(B_i)}, \\ g \stackrel{\text{def}}{=} g_{start(a)}, \ g'_{B_i} \stackrel{\text{def}}{=} g_{end(B_i)}. \end{array}$$

Define the specification expressions $c_{end(a)}$ and $g_{end(a)}$ as follows:

$$c_{end(a)} \stackrel{\text{def}}{=} c_1 \wedge \ldots \wedge c_m, \quad g_{end(a)} \stackrel{\text{def}}{=} g_1 \vee \ldots \vee g_m.$$

Now we prove that the following inequalities hold:

1. $\varphi_a \wedge c \leq \theta_a(c_{end(a)})$, i.e. $\varphi_a \wedge c \leq \theta_a(c_1) \wedge \ldots \wedge \theta_a(c_m)$,

2. $\theta_a(g_{end(a)}) \wedge \varphi_a \wedge c \leq g$, which is equivalent to the inequality

$$\theta_a(g_1) \vee \ldots \vee \theta_a(g_m) \leq (\varphi_a \wedge c) \to g,$$

3. $\varphi_{B_i} \wedge c_1 \wedge \ldots \wedge c_m \leq \theta_{B_i}(c'_{B_i})$,

4. $\theta_{B_i}(g'_{B_i}) \wedge \varphi_{B_i} \wedge c_1 \wedge \ldots \wedge c_m \leq g_1 \vee \ldots \vee g_m$.

The first inequality follows from the property of the operator "$\wedge$". The second inequality follows from the property of the operator "$\vee$". The third and fourth inequalities are trivial.

We now prove that all the conditions of theorem 6 hold, if

- the set N is augmented to

$$N_a \stackrel{\text{def}}{=} N \sqcup \{end(a)\},$$

- the sets $C = \{c_n \mid n \in N\}$ and $G = \{g_n \mid n \in N\}$ are augmented to the sets

$$C_a \stackrel{\text{def}}{=} \{c_n \mid n \in N_a\} \quad \text{and} \quad G_a \stackrel{\text{def}}{=} \{g_n \mid n \in N_a\}$$

  respectively, which are obtained from C and G by adding the specification expressions $c_{end(a)}$ and $g_{end(a)}$, that are defined above,

- the set $Paths$ is changed on the set $Paths_a$:

$$\begin{array}{l} Paths_a \stackrel{\text{def}}{=} \\ (Paths \setminus \{A_1, \ldots, A_m\}) \sqcup \{(a)\} \sqcup \{B_1, \ldots, B_m\}. \end{array}$$

The conditions of theorem 6 for the sets N_a, C_a, G_a and $Paths_a$ have the following form.

1. For every $n \in Nodes(SA) \setminus N_a$ there is a unique edge $b \in Edges(SA)$ with the property $end(b) = n$.

   This condition holds because

$$(Nodes(SA) \setminus N_a) \subseteq (Nodes(SA) \setminus N).$$

2. For every $b \in Edges(SA)$ there is a path
$B \in Paths_a$, such that $b \in B$.

   This condition holds because by assumption there is a path $A \in Paths$, such that $b \in A$, and

   - if $A \notin \{A_1, \ldots, A_m\}$, then $A \in Paths_a$, and the required path B is equal to A,
   - if $A \in \{A_1, \ldots, A_m\}$, i.e. $A = A_i$ for some $i \in \{1, \ldots, m\}$ then
     - either $b = a$, in this case the required path B is equal to (a),
     - or $b \in B_i$, in this case the required path B is equal to B_i.

3. Let B be a path from the set $Paths_a$:

$$B = (b_1, \ldots, b_h),$$

   then for every pair $j_1, j_2 \in \{1, \ldots, h\}$

$$j_1 \neq j_2 \quad \Rightarrow \quad end(b_{j_1}) \neq end(b_{j_2}).$$

   This condition holds because

   - the path (a) satisfies this condition, and
   - every path from the set $Paths_a \setminus \{(a)\}$ is a subpath of some path from the set $Paths$.

4. Let B be a path from the set $Paths_a$:

$$B = (b_1, \ldots, b_h),$$

   then $\forall j \in 1, \ldots, h-1 \quad end(b_i) \notin N_a$.

   This condition holds because

   - the path (a) satisfies this condition, and

- if a path $B = (b_1, \ldots, b_h)$ from the set $Paths_a \setminus \{(a)\}$ does not satisfy this condition, then there is a number $i \in 1, \ldots, h-1$, such that

$$end(b_i) = end(a).$$

By assumption, for every $n \in Nodes(SA) \setminus N$ there is a unique edge $b \in Edges(SA)$ with the property $end(b) = n$.

In particular, for $n \stackrel{\text{def}}{=} end(a)$, there is a unique edge $b \in Edges(SA)$ with the property $end(b) = end(a)$.

Consequently, $b_i = a$. This is impossible, because

  - If $i > 1$, then by assumption

$$start(a) = end(b_{i-1}) \notin N,$$

    that contradicts to the definition of a.
  - If $i = 1$, then $B \notin \{B_1, \ldots, B_m\}$, and consequently $B \in Paths \setminus \{A_1, \ldots, A_m\}$.
    This contradicts to the definition of the set $\{A_1, \ldots, A_m\}$.

5. For every $B \in Paths_a$

$$start(B) \in N_a \quad \text{and} \quad end(B) \in N_a.$$

This property follows from the definitions of the sets N_a and $Paths_a$.

6. For every $B \in Paths_a$ the following conditions hold:

$$\begin{array}{rcl} \varphi_B \wedge c_B & \leq & \theta_B(c'_B) \\ \theta_B(g'_B) \wedge \varphi_B \wedge c_B & \leq & g_B \end{array}$$

where

$$c_B \stackrel{\text{def}}{=} c_{start(B)}, \ c'_B \stackrel{\text{def}}{=} c_{end(B)}, \\ g_B \stackrel{\text{def}}{=} g_{start(B)}, \ g'_B \stackrel{\text{def}}{=} g_{end(B)}.$$

This property has been proved above.

Thus, the conditions of theorem 6 hold if the sets N, C, G and $Paths$ are changed to the set N_a, C_a, G_a and $Paths_a$ respectively.

The cardinality of the set $Nodes(SA) \setminus N_a$ is less that the cardinality of the set $Nodes(SA) \setminus N$.

Consequently, by the induction, there is a pair of $Nodes(SA)$–indexed sets

$$\tilde{C} = \{\tilde{c}_n \mid n \in Nodes(SA)\},$$
$$\tilde{G} = \{\tilde{g}_n \mid n \in Nodes(SA)\}$$

of specification expressions from $Spec(SA)$, such that

1. $\forall n \in N_a \quad \tilde{c}_n = c_n$ and $\tilde{g}_n = g_n$,
2. SA is locally correct with respect to the pair $(\tilde{C}, \tilde{G})$.

Since $N \subset N_a$, then the sets $\tilde{C}$ and $\tilde{G}$ satisfy the statement of theorem 6.
■

7 Alternating bit protocol (ABP) example

In this section we consider the Alternating Bit Protocol ([1]) implemented by a distributed agent as an example of the concepts introduced so far in the report.

7.1 The ABP distributed agent

The **ABP distributed agent** is defined as follows:

$$ABP \stackrel{\text{def}}{=} (Sender, Receiver, Env^1, Env^2; Channels)$$

where

- $Inputs(Sender) \stackrel{\text{def}}{=} \{in, \delta\}$,
  $Outputs(Sender) \stackrel{\text{def}}{=} \{\alpha\}$,
- $Inputs(Receiver) \stackrel{\text{def}}{=} \{\beta\}$,
  $Outputs(Receiver) \stackrel{\text{def}}{=} \{\gamma, out\}$,

- $Inputs(Env^1) \stackrel{\text{def}}{=} \{p, p_1\}$,
  $Outputs(Env^1) \stackrel{\text{def}}{=} \{q_1\}$,

- $Inputs(Env^2) \stackrel{\text{def}}{=} \{p_2\}$,
  $Outputs(Env^2) \stackrel{\text{def}}{=} \{q_2\}$,

- $Channels \stackrel{\text{def}}{=} \{(\alpha, p_1),\ (q_1, \beta),\ (\gamma, p_2),\ (q_2, \delta)\}$,

and the definitions of $Sender, Receiver, Env^1, Env^2$ are presented below.

The flow graph of ABP has the following form:

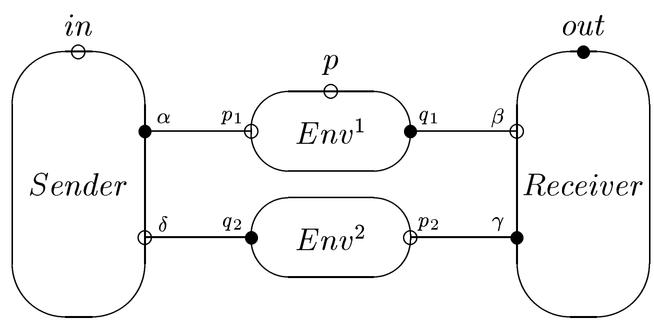

Figure 2: the flow graph for ABP.

The functioning of ABP consists of transmission of messages from the $Sender$ to the $Receiver$ through the environments Env^1, where Env^1 is assumed to be noisy (i.e it may corrupt messages). The $Sender$ receives acknowledgements through the Env^2, which is assumed to be noise free.

The model of the noisy environment Env^1 is assumed to be as follows. After receiving from the port p_1 a message which has the form (u, x) (where u is a control bit and x is a body), it receives a corruption signal $corr$, which is a boolean value. If $corr = \top$, then Env^1 outputs on the port q_1 the pair $(\neg u, *)$, where the symbol $*$ denotes a corrupted body. If $corr = \bot$, then Env^1 outputs on the port q_1 the original pair (u, x).

The functioning of the sequential agents $Sender$, $Receiver$, Env^1 and Env^2 can be informally described as follows.

- *Initialization*:
  The $Sender$ and $Receiver$ have boolean variables u and v, respectively, called **control bits**, whose values are initialized to $\top$.

- *Sender*:

  1. receives a message x on the input port *in*,
  2. adds the control bit u to this message,
  3. sends the pair (u, x) to the agent Env^1,
  4. receives an acknowledgement bit η from the port δ,
  5. if the acknowledgement bit η is equal to the current control bit u, then *Sender* inverts the control bit, and starts another cycle from step (1),
  6. otherwise it sends the current message (u, x) again, until the acknowledgement bit η is equal to the current control bit u.

- Env^1:

  1. receives a pair (u, x) on the input port p_1,
  2. receives the corruption signal *corr* on the input port p ,
  3. if $corr = \perp$, then Env^1 sends the unmodified (u, x) to the *Receiver*,
  4. if $corr = \top$, then Env^1 sends $(\neg u, *)$ to the *Receiver*.

- *Receiver*:

  1. receives a pair (ξ, y) on the input port β,
  2. if ξ is equal to the current control bit v of the *Receiver*, then the *Receiver*
     (a) extracts the message y, and sends it to the output port *out*,
     (b) sends the control bit v on the port γ,
     (c) inverts v,
     (d) and starts execution of new cycle of its work from its step (1),
  3. otherwise it sends the inverted current control bit (that is $\neg v$) to the output port γ, and starts execution of new cycle of its work from its step (1).

- Env^2

  1. receives a bit η on the input port p_2,

2. sends the bit η to the output port q_2.

The specification expression $Spec(ABP)$, which specifies the behavior of ABP, can be choosen as follows:

$$Spec(ABP) \stackrel{\text{def}}{=} ABP(in, out)$$

where the specification system ABP consists of the equation

$$\rho(in, out) \stackrel{\text{def}}{=} \left[\begin{array}{l} \left\{ \begin{array}{l} \mathbf{head}(in) = \omega \\ \rho(\mathbf{tail}(in), out) \end{array} \right\} \\ \\ \left\{ \begin{array}{l} \mathbf{head}(out) = \omega \\ \rho(in, \mathbf{tail}(out)) \end{array} \right\} \\ \\ \left\{ \begin{array}{l} \mathbf{head}(out) = \mathbf{head}(in) \\ \rho(\mathbf{tail}(in), \mathbf{tail}(out)) \end{array} \right\} \end{array} \right]$$

7.2 The sequential agent $Sender$

The set of variables of $Sender$ is defined as follows:
$Var(Sender) \stackrel{\text{def}}{=} \{u, \chi, \eta, x\}$, where

1. $type(u) = type(\chi) = type(\eta) = bool$,
2. $\overline{type(x)} = type(in)$.

The sequential agent $Sender$ has the following form:

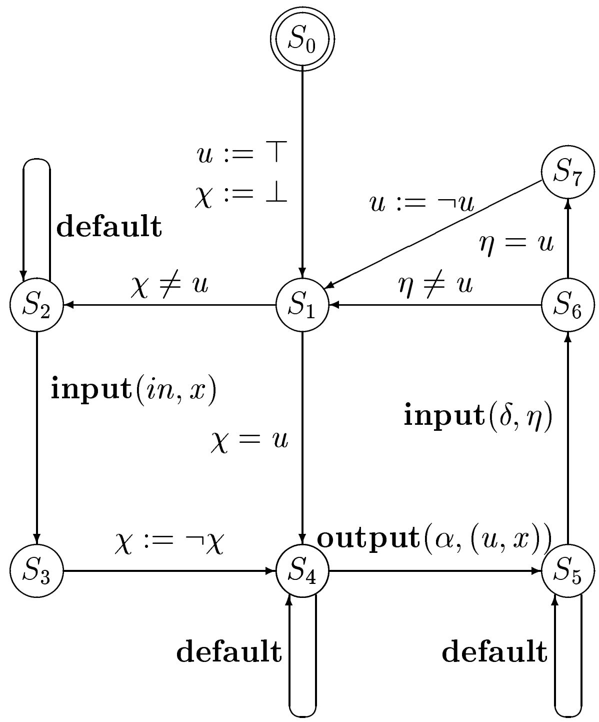

Figure 3: *Sender*.

The specification expression $Spec(Sender)$, which specifies the behavior of $Sender$, can be choosen as follows:

$$Spec(Sender) \stackrel{\text{def}}{=} Sender_{\top}(in, \alpha, \delta)$$

where the specification system $Sender$ has the following form:

$$\begin{cases} \rho_{\top}(in, \alpha, \delta) = sender_{\top} \\ \rho_{\bot}(in, \alpha, \delta) = sender_{\bot} \end{cases}$$

and the specification expressions $sender_u$ for $u = \top$ and $\bot$ are defined as follows:

$$
sender_u \stackrel{\text{def}}{=} \left[\begin{array}{l}
\left\{\begin{array}{l} \mathbf{head}(in) = \omega \\ \rho_u(\mathbf{tail}(in), \alpha, \delta) \end{array}\right\} \\ \\
\left\{\begin{array}{l} \mathbf{head}(\alpha) = \omega \\ \rho_u(in, \mathbf{tail}(\alpha), \delta) \end{array}\right\} \\ \\
\left\{\begin{array}{l} \mathbf{head}(\delta) = \omega \\ \rho_u(in, \alpha, \mathbf{tail}(\delta)) \end{array}\right\} \\ \\
\left\{\begin{array}{l} \mathbf{head}(in, \alpha, \delta) \neq \omega \\ \mathbf{head}(\alpha) = (u, \mathbf{head}(in)) \\ \mathbf{head}(\delta) = u \\ \rho_{\neg u}(\mathbf{tail}(in), \mathbf{tail}(\alpha), \mathbf{tail}(\delta)) \end{array}\right\} \\ \\
\left\{\begin{array}{l} \mathbf{head}(in, \alpha, \delta) \neq \omega \\ \mathbf{head}(\alpha) = (u, \mathbf{head}(in)) \\ \mathbf{head}(\delta) = \neg u \\ \rho_u(in, \mathbf{tail}(\alpha), \mathbf{tail}(\delta)) \end{array}\right\}
\end{array}\right]
$$

Let $send_u(in, \alpha, \delta)$ denote the following specification expression:

$$
\left[\begin{array}{l} (u = \top) \wedge sender_\top(in, \alpha, \delta) \\ (u = \bot) \wedge sender_\bot(in, \alpha, \delta) \end{array}\right].
$$

Verification of the agent *Sender* can be done with the use the following safety assertions and liveness assertions:

- $g_0 \stackrel{\text{def}}{=} sender_\top(in, \alpha, \delta)$,
  $c_0 \stackrel{\text{def}}{=} \top$,
- $g_1 \stackrel{\text{def}}{=} \left[\begin{array}{l} (\chi \neq u) \wedge send_u(in, \alpha, \delta) \\ (\chi = u) \wedge send_u(x \cdot in, \alpha, \delta) \end{array}\right]$,
  $c_1 \stackrel{\text{def}}{=} \top$,
- $g_2 = send_u(in, \alpha, \delta)$,
  $c_2 \stackrel{\text{def}}{=} (\chi \neq u)$,

- $g_3 = send_u(x \cdot in, \alpha, \delta)$,
  $c_3 \stackrel{\text{def}}{=} (\chi \neq u)$,

- $g_4 = send_u(x \cdot in, \alpha, \delta)$,
  $c_4 \stackrel{\text{def}}{=} (\chi = u)$,

- $g_5 = send_u(x \cdot in, (u, x) \cdot \alpha, \delta)$,
  $c_5 \stackrel{\text{def}}{=} (\chi = u)$,

- $g_6 = send_u(x \cdot in, (u, x) \cdot \alpha, \eta \cdot \delta)$,
  $c_6 \stackrel{\text{def}}{=} (\chi = u)$,

- $g_7 = send_{\neg u}(in, \alpha, \delta)$,
  $c_7 \stackrel{\text{def}}{=} (\chi = u)$.

The checking of the inequalities from the definition of local correctness is trivial for all edges with the exception of the edges, end of which is the node S_1.

For example, the checking of the inequalities at the edge from S_6 to S_1 has the following form.

We must prove the inequalities

$$\begin{array}{rcl} \varphi_a \wedge c_a & \leq & \theta_a(c'_a) \\ \theta_a(g'_a) \wedge \varphi_a \wedge c_a & \leq & g_a \end{array}$$

where

- $\varphi_a = (\eta \neq u) \wedge (\mathbf{head}(in, \alpha, \delta) = \omega)$,

- $\theta_a = \left\{ \begin{array}{l} in := \mathbf{tail}(\mathbf{in}) \\ \alpha := \mathbf{tail}(\alpha) \\ \delta := \mathbf{tail}(\delta) \end{array} \right\}$,

- $c_a = (\chi = u)$,

- $c'_a = \top$,

- $g_a = send_u(x \cdot in, (u, x) \cdot \alpha, \eta \cdot \delta)$,

- $g'_a = \left[\begin{array}{l} (\chi \neq u) \wedge send_u(in, \alpha, \delta) \\ (\chi = u) \wedge send_u(x \cdot in, \alpha, \delta) \end{array} \right]$.

Since $c'_a = \top$, the inequality $\varphi_a \wedge c_a \leq \theta_a(c'_a)$ holds.
The inequality $\theta_a(g'_a) \wedge \varphi_a \wedge c_a \leq g_a$ has the following form:

$$\left\{ \begin{array}{l} \left[\begin{array}{l} (\chi \neq u) \wedge \theta_a(send_u(in, \alpha, \delta)) \\ (\chi = u) \wedge \theta_a(send_u(x \cdot in, \alpha, \delta)) \end{array} \right] \\ \\ (\eta \neq u) \wedge (\mathbf{head}(in, \alpha, \delta) = \omega) \\ \\ (\chi = u) \end{array} \right\} \leq$$

$$send_u(x \cdot in, (u, x) \cdot \alpha, \eta \cdot \delta).$$

The last inequality is equivalent to the inequality

$$\left\{ \begin{array}{l} send_u(x \cdot \mathbf{tail}(in), \mathbf{tail}(\alpha), \mathbf{tail}(\delta)) \\ (\eta \neq u) \\ (\mathbf{head}(in, \alpha, \delta) = \omega) \\ (\chi = u) \end{array} \right\} \leq$$

$$send_u(x \cdot in, (u, x) \cdot \alpha, \eta \cdot \delta).$$

Using the definition of the expression $send_u(in, \alpha, \delta)$, the last inequality can be rewritten as follows:

$$\left[\begin{array}{l}\left\{\begin{array}{l}(u=\top)\\ sender_{\top}(x\cdot\mathbf{tail}(in),\mathbf{tail}(\alpha),\mathbf{tail}(\delta))\\ (\eta=\bot)\\ (\mathbf{head}(in,\alpha,\delta)=\omega)\\ (\chi=\top)\end{array}\right\}\\ \\ \left\{\begin{array}{l}(u=\bot)\\ sender_{\bot}(x\cdot\mathbf{tail}(in),\mathbf{tail}(\alpha),\mathbf{tail}(\delta))\\ (\eta=\top)\\ (\mathbf{head}(in,\alpha,\delta)=\omega)\\ (\chi=\bot)\end{array}\right\}\end{array}\right]\leq$$

$$\left[\begin{array}{l}\left\{\begin{array}{l}(u=\top)\\ sender_{\top}(x\cdot in,(\top,x)\cdot\alpha,\eta\cdot\delta)\end{array}\right\}\\ \\ \left\{\begin{array}{l}(u=\bot)\\ sender_{\bot}(x\cdot in,(\bot,x)\cdot\alpha,\eta\cdot\delta)\end{array}\right\}\end{array}\right].$$

The last inequality follows from the following pair of inequalities:

$$\left\{\begin{array}{l}sender_{\top}(x\cdot\mathbf{tail}(in),\mathbf{tail}(\alpha),\mathbf{tail}(\delta))\\ (\mathbf{head}(in,\alpha,\delta)=\omega)\end{array}\right\}\leq$$

$$sender_{\top}(x\cdot in,(\top,x)\cdot\alpha,\bot\cdot\delta),$$

$$\left\{\begin{array}{l}sender_{\bot}(x\cdot\mathbf{tail}(in),\mathbf{tail}(\alpha),\mathbf{tail}(\delta))\\ (\mathbf{head}(in,\alpha,\delta)=\omega)\end{array}\right\}\leq$$

$$sender_{\bot}(x\cdot in,(\bot,x)\cdot\alpha,\top\cdot\delta).$$

A specification expression s_1 is said to be **equivalent** to a specification expression s_2, if the inequalities $s_1 \leq s_2$ and $s_2 \leq s_1$ hold. The definition of the specification expressions $sender_u$ for $u = \top, \bot$ implies that the specification expression

$$sender_{\top}(x\cdot in,(\top,x)\cdot\alpha,\bot\cdot\delta)$$

is equivalent to the specification expression

$$sender_{\top}(x\cdot in,\alpha,\delta),$$

and the specification expression

$$sender_{\perp}(x \cdot in, (\perp, x) \cdot \alpha, \top \cdot \delta)$$

is equivalent to the specification expression

$$sender_{\perp}(x \cdot in, \alpha, \delta).$$

Thus, we must prove the following pair of inequalities:

$$\left\{ \begin{array}{l} sender_{\top}(x \cdot \mathbf{tail}(in), \mathbf{tail}(\alpha), \mathbf{tail}(\delta)) \\ (\mathbf{head}(in, \alpha, \delta) = \omega) \end{array} \right\} \leq$$

$$sender_{\top}(x \cdot in, \alpha, \delta),$$

$$\left\{ \begin{array}{l} sender_{\perp}(x \cdot \mathbf{tail}(in), \mathbf{tail}(\alpha), \mathbf{tail}(\delta)) \\ (\mathbf{head}(in, \alpha, \delta) = \omega) \end{array} \right\} \leq$$

$$sender_{\perp}(x \cdot in, \alpha, \delta),$$

which follows from the inequalities:

$$\begin{array}{l} sender_{\top}(x \cdot \mathbf{tail}(in), \mathbf{tail}(\alpha), \mathbf{tail}(\delta)) \leq \\ sender_{\top}(x \cdot \omega \cdot \mathbf{tail}(in), \omega \cdot \mathbf{tail}(\alpha), \omega \cdot \mathbf{tail}(\delta)), \end{array}$$

$$\begin{array}{l} sender_{\perp}(x \cdot \mathbf{tail}(in), \mathbf{tail}(\alpha), \mathbf{tail}(\delta)) \leq \\ sender_{\perp}(x \cdot \omega \cdot \mathbf{tail}(in), \omega \cdot \mathbf{tail}(\alpha), \omega \cdot \mathbf{tail}(\delta)). \end{array}$$

The last inequalities follow from the definition of the expressions $sender_{\top}$ and $sender_{\perp}$.

7.3 The sequential agent *Receiver*

The set of variables of *Receiver* is defined as follows:
$Var(Receiver) \stackrel{\text{def}}{=} \{\xi, v, y, z\}$, where

1. $type(v) = type(\xi) = bool$,
2. $\overline{type(y)} = type(out)$,
3. $type(z) = (bool, type(y))$.

The sequential agent *Receiver* has the following form: (note that some of the steps the *Receiver* given in subsection 7.1 have been modified to reduce the number of its nodes)

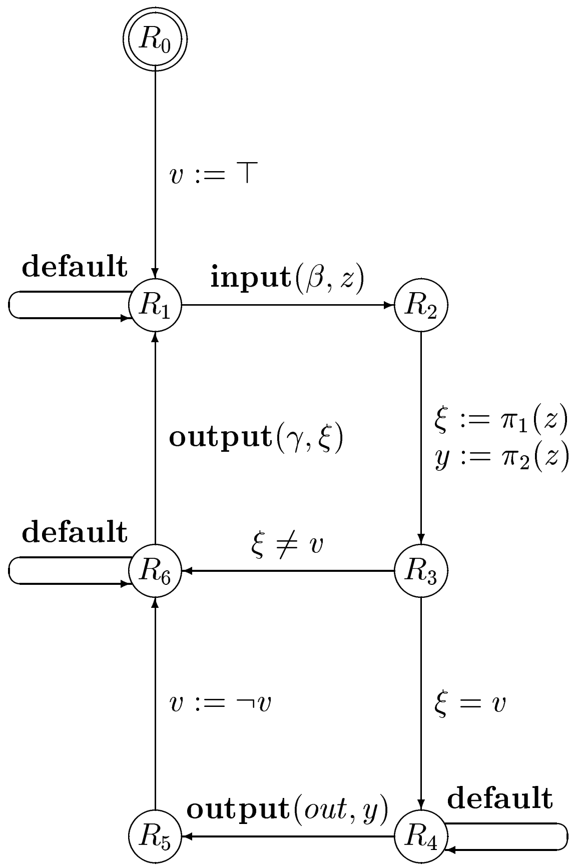

Figure 4: *Receiver*.

The specification expression $Spec(Receiver)$, which specifies the behavior of *Receiver*, can be choosen as follows:

$$Spec(Receiver) \stackrel{\text{def}}{=} Receiver_{\top}(out, \beta, \gamma)$$

where the specification system *Receiver* has the following form:

$$\begin{cases} \rho_{\top}(out, \beta, \gamma) = receiver_{\top} \\ \rho_{\perp}(out, \beta, \gamma) = receiver_{\perp} \end{cases}$$

and the specification expressions $receiver_v$ for $v = \top, \bot$ are defined as follows:

$$receiver_v \stackrel{\text{def}}{=} \left[\begin{array}{l} \left\{\begin{array}{l} \mathbf{head}(out) = \omega \\ \rho_v(\mathbf{tail}(out), \beta, \gamma) \end{array}\right\} \\ \\ \left\{\begin{array}{l} \mathbf{head}(\beta) = \omega \\ \rho_v(out, \mathbf{tail}(\beta), \gamma) \end{array}\right\} \\ \\ \left\{\begin{array}{l} \mathbf{head}(\gamma) = \omega \\ \rho_v(out, \beta, \mathbf{tail}(\gamma)) \end{array}\right\} \\ \\ \left\{\begin{array}{l} \mathbf{head}(out, \beta, \gamma) \neq \omega \\ \pi_1(\mathbf{head}(\beta)) = v \\ \pi_2(\mathbf{head}(\beta)) = \mathbf{head}(out) \\ \mathbf{head}(\gamma) = v \\ \rho_{\neg v}(\mathbf{tail}(out), \mathbf{tail}(\beta), \mathbf{tail}(\gamma)) \end{array}\right\} \\ \\ \left\{\begin{array}{l} \mathbf{head}(out, \beta, \gamma) \neq \omega \\ \pi_1(\mathbf{head}(\beta)) = \neg v \\ \mathbf{head}(\gamma) = \neg v \\ \rho_v(out, \mathbf{tail}(\beta), \mathbf{tail}(\gamma)) \end{array}\right\} \end{array}\right]$$

Let $receiv_v(out, \beta, \gamma)$ denote the following specification expression:

$$\left[\begin{array}{l} (v = \top) \wedge receiver_\top(out, \beta, \gamma) \\ (v = \bot) \wedge receiver_\bot(out, \beta, \gamma) \end{array}\right].$$

Verification of the agent *Receiver* can be done with the use the following safety assertions and liveness assertions:

- $g_0 \stackrel{\text{def}}{=} receiver_\top(out, \beta, \gamma)$,
  $c_0 \stackrel{\text{def}}{=} \top$,
- $g_1 \stackrel{\text{def}}{=} receiv_v(out, \beta, \gamma)$,
  $c_1 \stackrel{\text{def}}{=} \top$,
- $g_2 \stackrel{\text{def}}{=} receiv_v(out, z \cdot \beta, \gamma)$,
  $c_2 \stackrel{\text{def}}{=} \top$,

- $g_3 \stackrel{\text{def}}{=} receiv_v(out, z \cdot \beta, \gamma)$,
  $c_3 \stackrel{\text{def}}{=} (\xi = \pi_1(z)) \wedge (y = \pi_2(z))$,

- $g_4 \stackrel{\text{def}}{=} receiv_v(out, z \cdot \beta, \gamma)$,
  $c_4 \stackrel{\text{def}}{=} (\xi = \pi_1(z)) \wedge (y = \pi_2(z)) \wedge (\xi = v)$,

- $g_5 \stackrel{\text{def}}{=} receiv_v(y \cdot out, z \cdot \beta, \gamma)$,
  $c_5 \stackrel{\text{def}}{=} (\xi = \pi_1(z)) \wedge (y = \pi_2(z)) \wedge (\xi = v)$,

- $g_6 = \left[\begin{array}{l} receiv_{\neg v}(y \cdot out, z \cdot \beta, \gamma) \\ receiv_v(out, z \cdot \beta, \gamma) \end{array} \right]$,
  $c_6 \stackrel{\text{def}}{=} (\xi = \pi_1(z)) \wedge (y = \pi_2(z)) \wedge (\xi = \neg v)$.

7.4 The sequential agent Env^1

The set of variables of Env^1 is defined as follows:
$Var(Env^1) \stackrel{\text{def}}{=} \{\kappa, corr\}$, where

- $type(\kappa) = (bool, \tau)$, where τ is such that $\bar{\tau} = type(in)$,
- $type(corr) = bool$.

We assume that the domain $\mathcal{D}_\tau$ contains a special element $*$, which denotes a corrupted message.

The sequential agent Env^1 has the following form:

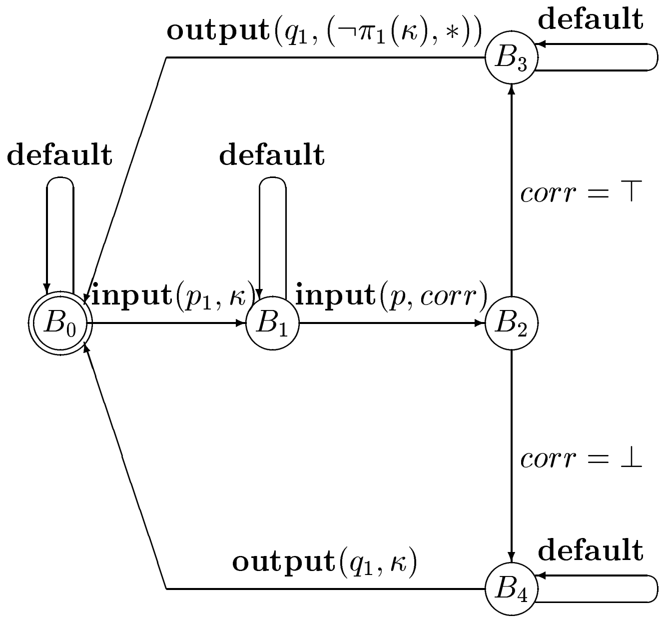

Figure 5: Env^1.

The specification expression $Spec(Env^1)$, which specifies the behavior of Env^1, can be choosen as follows:

$$Spec(Env^1) \stackrel{\text{def}}{=} Env^{\,1}(p_1, q_1)$$

where the specification system Env^1 consists of the equation

$$\rho(p_1, q_1) =$$

$$= \left[\begin{array}{l} \left\{ \begin{array}{l} \mathbf{head}(p_1) = \omega \\ \rho(\mathbf{tail}(p_1), q_1) \end{array} \right\} \\ \\ \left\{ \begin{array}{l} \mathbf{head}(q_1) = \omega \\ \rho(p_1, \mathbf{tail}(q_1)) \end{array} \right\} \\ \\ \left\{ \begin{array}{l} \mathbf{head}(p_1, q_1) \neq \omega \\ \mathbf{head}(q_1) = \mathbf{head}(p_1) \\ \rho(\mathbf{tail}(p_1), \mathbf{tail}(q_1)) \end{array} \right\} \\ \\ \left\{ \begin{array}{l} \mathbf{head}(p_1, q_1) \neq \omega \\ \pi_1(\mathbf{head}(q_1)) = \neg \pi_1(\mathbf{head}(p_1)) \\ \rho(\mathbf{tail}(p_1), \mathbf{tail}(q_1)) \end{array} \right\} \end{array} \right]$$

Note that the specification expression given above specifies the behavior of Env^1 partially. In particular, the port p is not included in the specification expression. The reason of using the partial specification is that for the proving of correctness of the ABP the information about behavior on the port p is not essential.

Verification of the agent Env^1 can be done with the use the following safety assertions and liveness assertions:

- $g_0 \stackrel{\text{def}}{=} Env^1(p_1, q_1)$,
  $c_0 \stackrel{\text{def}}{=} \top$,

- $g_1 \stackrel{\text{def}}{=} g_2 \stackrel{\text{def}}{=} g_3 \stackrel{\text{def}}{=} g_4 \stackrel{\text{def}}{=} Env^1(\kappa \cdot p_1, q_1)$,
  $c_1 \stackrel{\text{def}}{=} c_2 \stackrel{\text{def}}{=} c_3 \stackrel{\text{def}}{=} c_4 \stackrel{\text{def}}{=} \top$.

7.5 The sequential agent Env^2

The set of variables of Env^2 is defined as follows:
$Var(Env^2) \stackrel{\text{def}}{=} \{\eta\}$, where $type(\eta) = bool$.

The sequential agent Env^2 has the following form:

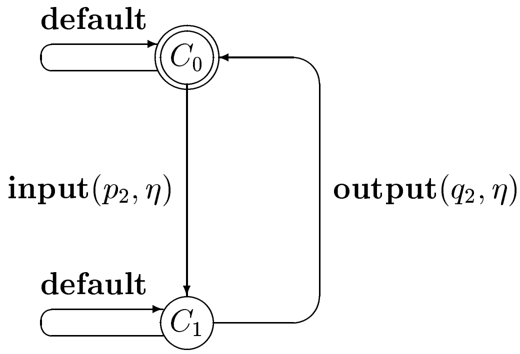

Figure 6: Env^2.

The specification expression $Spec(Env^2)$, which specifies the behavior of Env^2, can be choosen as follows:

$$Spec(Env^2) \stackrel{\text{def}}{=} Env^2(p_2, q_2)$$

where the specification system Env^2 consists of the equation

$$\rho(p_2, q_2) \stackrel{\text{def}}{=} \left[\begin{array}{l} \left\{ \begin{array}{l} \mathbf{head}(p_2) = \omega \\ \rho(\mathbf{tail}(p_2), q_2) \end{array} \right\} \\ \\ \left\{ \begin{array}{l} \mathbf{head}(q_2) = \omega \\ \rho(p_2, \mathbf{tail}(q_2)) \end{array} \right\} \\ \\ \left\{ \begin{array}{l} \mathbf{head}(p_2, q_2) \neq \omega \\ \mathbf{head}(q_2) = \mathbf{head}(p_2) \\ \rho(\mathbf{tail}(p_2), \mathbf{tail}(q_2)) \end{array} \right\} \end{array} \right]$$

Verification of the agent Env^2 can be done with the use the following safety assertions and liveness assertions:

- $g_0 \stackrel{\text{def}}{=} Env^2(p_2, q_2)$,
  $c_0 \stackrel{\text{def}}{=} \top$,
- $g_1 \stackrel{\text{def}}{=} Env^2(\eta \cdot p_2, q_2)$,
  $c_1 \stackrel{\text{def}}{=} \top$.

8 Conclusion

In the present report we have described a new approach to the problem of specification and verification of distributed communicating systems.

The proposed approach to specification and verification has the following advantages in comparison with other approaches to specification and verification of DCSs:

1. The proposed verification technique allows the implementation the verification process in an interactive form. (In this sense the proposed technique is similar to the verification technique founded on the Floyd's inductive assertion method, which allows use of programmer's intuition for construction of the necessary inductive assertions.)

2. Use of fixpoint constructions in the specification language allows a simple and precise description of the behavior of a DCS. In comparison with encoding of specifications by temporal formulas, the fixpoint constructions are more natural for representation of semantics of the specified properties.

3. The operation of cartesian product commonly used in construction of state transition graphs for the whole DCS is not employed; this results into a drastic reduction of the complexity of verification of a DCS.

In order to apply the proposed technique we need the development of a methodology for:

- the construction of appropriate specification expressions for sequential agents that are components of a given distributed agent,

- the construction of appropriate safety assertions and liveness assertions for proving that the sequential agents meet their specification expressions,

- proving that conjunction of specification expressions for the sequential agents and the conditions of equality of queues corresponding to the connected ports implies the specification expression for the distributed agent.

Acknowledgements

The authors would like to thank Dmitry Zhukov, researcher of Institute of System Programming of Russian Academy of Sciences, Moscow, for his comments and corrections.

This research is partially supported by the grant OGP0089 from Natural Sciences and Engineering Research Council of Canada.

References

[1] **K.A.Bartlett**, **R.A.Scantlebury**, **P.T.Wilkinson:** A note on reliable full-duplex transmission over half-duplex links, *Communications of the ACM, volume 12, no.5, pp. 260-261, 265, 1969.*

[2] **K.M.Chandy, J.Misra:** Parallel program design: a foundation, *Addison-Wesley, Reading, MA, 1988.*

[3] **E.M.Clarke, E.A.Emerson, A.P.Sistla:** Automatic verification of finite-state concurrent systems using temporal logic specifications, *ACM Transactions on Programming Languages and Systems, 8(2):244-263, April 1986.*

[4] **P. Degano, U. Montanari:** A Model of distributed systems based on graph rewriting, *Journal of the ACM, Vol. 34, no. 2, April 1987, pp. 411-449.*

[5] **P. Degano, R. De Nicola, U. Montanari:** A distributed operational semantics for CCS based on condition/event systems, *Acta Informatica 26, pp.59-91, 1988.*

[6] **P. Degano, R. De Nicola, U. Montanari:** A partial ordering semantics for CCS, *Theoretical Computer Science, 75, pp. 223-262, 1990.*

[7] **E.A.Emerson:** Temporal and modal logic,, *in: Jan van Leeuwen, editor: Handbook of Theoretical Computer Science, vol. B, ch.16, Elsevier and MIT press, 1990.*

[8] **R.W.Floyd:** Assigning meanings to programs, *Proc. Symp. Appl. Math.,* **19***; in: J.T.Schwartz (ed.), Mathematical Aspects of Computer*

Science, pp. 19-32, American Mathematical Society, Providence, R.I., 1967.

[9] **R. Gerth, R. Kuiper, D. Peled, W. Penczek:** A partial order approach to branching time logic model checking, *Information and Computation, bf 150, pp. 132-152, 1999.*

[10] **P. Godefroid:** Using partial orders to improve automatic verification methods, *in: "Computer Aided Verification 1990" (E.M.Clarke and R.P.Kurshan, Eds.), DIMACS, Vol.3, pp. 321-339, 1991.*

[11] **C.A.R.Hoare:** Communicating sequential processes, *Prentice-Hall, 1985.*

[12] **L.Lamport:** The temporal logic of actions, *ACM Transactions on Programming Languages and Systems, 16(3):872-923, May, 1994.*

[13] **N.Lynch:** Distributed algorithms, *Morgam Kaufmann Publishers, 1986.*

[14] **N.Lynch, M.Tuttle:** Hierarchical correctness proofs for distributed algorithms, *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, p.137-151, Vancouver, British Columbia, Canada, August 1987.*

[15] **Z.Manna:** Mathematical theory of computation, *McGraw-Hill, 1974.*

[16] **Z.Manna, A.Pnueli:** The temporal logic of reactive and concurrent systems: specification, *Springer-Verlag, New York, 1992.*

[17] **R.A.Milner:** A Calculus of communicating systems, *Lecture Notes in Computer Science, vol. 92, Springer-Verlag, Berlin, 1980.*

[18] **R.A.Milner:** Communication and concurrency, *Prentice Hall, 1989.*

[19] **R.A.Milner:** Communicating and mobile systems: the π–calculus, *Cambridge University Press, 1999.*

[20] **U. Montanari, F. Rossi:** Graph rewriting for a partial ordering semantics of concurrent constraint programming, *Theoretical Computer Science, 109 (1993) pp. 225-256.*

[21] **D.Peled:** Combining partial order reductions with on-the fly model checking, *in: "6th Conference on Computer Aided Verification", Lecture Notes in Computer Science, Vol. 818, pp. 377-390, 1994.*

[22] **A. Valmari:** Stubborn sets for reduced state space generation, *in: "10th International Conference on Application and Theory of Petri Nets", Lecture Notes in Computer Science, Vol. 483, pp. 491-515, 1989.*