Stepwise Refinement of Specifications

Dr. Timo Karvi

Department of Computer Science, University of Helsinki

P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland

E-mail: Timo.Karvi@cs.Helsinki.FI

Abstract

This article will study how to build specifications of distributed systems stepwise. It is supposed that the specifications are based on labeled transition systems or process algebras. In order to explain what it means for a specification to be correct, the concepts of a service specification and an equivalence are defined. First a bisimulation-based refinement technique is introduced. This has already been known for some time. Secondly, it is shown that the same kind of approach is possible in other semantics. Especially, divergence and trace semantics are described in detail.

1 Introduction

Distributed systems are prone to many kinds of errors. Errors, which are especially difficult to detect in testing, are related to synchronization. It is typical for these errors that complicated timing constraints must be satisfied before an error occurs. In testing it is very difficult to consider all the possible message sequences between several processes, so we need more than ordinary testing in order to convince ourselves of the correctness of a system.

Formal specifications are used to automatize the verification of distributed systems. The processes of the system are described in a very high-level language, so called specification language. The service of the system, i.e. the interface it gives to the users of the system, is written

in the language as well. Usually the service is much simpler than the specification of the system itself. After these two specifications have been written manually, it is possible to proceed automatically, for example as follows.

Suppose the specification language is CCS, CSP or Lotos (see [10, 8, 2], respectively). There are tools, based on these languages, which generate a certain kind of graph from the specification. This graph is known by the name *global state graph* or *reachability graph*. It contains information about all the possible message sequences between the processes in the system. Deadlocks and livelocks are visible in the graph, too. By comparing the reachability graph of the system with the graph of the service, one sees whether the system satisfies the service. This comparison involves the concept of an equivalence. In other words, if the two graphs are equivalent with respect to the given equivalence, the system is considered correct or verified. If the graphs are not equivalent, then something is wrong. The incorrect thing may be the description of the system, the description of the service or the algorithm or protocol itself. In any case, the reason for the problem must be detected and then the verification is tried again.

If the system is large, the specification will also be large. If problems are found during the verification, it may be difficult to detect the right place in the specification, which should be modified. The stepwise refinement of a specification is a method to build the specification gradually and to verify at every step that the construction proceeds in the right direction. The first paper dealing with this method explicitly seems to be [4], although all the necessary formal concepts have been introduced already in [5] and [3].

This article introduces the work in [5] and [4]. However, the Lotos is used instead of CCS and some new observations are made. The equivalence used in the above articles is the well-known bisimulation equivalence. Finally, it is shown that the same approach can be applied with respect to other equivalences as well.

2 Labeled transition systems

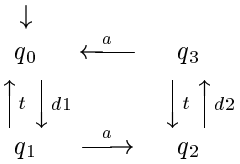
2.1 Definition of a transition system

Labeled transition systems are used to model processes in distributed systems.

Definition 1. *A labeled transition system* is a tuple (S, A, T, q_0), where

- S is a countable set of states;
- A is a countable set of actions; it can contain a special element i, the internal or invisible action;
- $T \subset S \times A \times S$ is a transition relation; if $(p, a, q) \in T$, we often write $p \xrightarrow{a} q$;
- $q_0 \in S$ is the initial state; we demand that every state in S is reachable from the initial state q_0 when the system is interpreted as a graph.

As an example consider a simple process which sends a data packet $d1$, waits for an acknowledgement a, sends another packet $d2$, waits again for a and then starts from the beginning. There is also a timer, which times out, if the acknowledgement does not come. The process as a transition system is as follows:



Thus sending and receiving messages is modelled by transitions. A transition does not tell, if its action is a sending or receiving action. In some versions of transition systems the symbols '!' and '?' are used to mark sending and receiving, respectively. In Lotos, sending and receiving are marked in a specification, but no longer in the labeled transition system generated from the specification. This article follows the Lotos conventions.

We note that internal calculations in a process are not visible in transition systems. Internal calculations can be modelled with the internal action i, if necessary. The action i can be used in many other situations as well, for example when modelling nondeterminism.

If P is a process representated as a transition system and P' is a state in P, we can consider P' as a new process. The initial state of the process P' is the state P'. The graph of P' is the same as the graph of P

with the exception of the initial state. (Of course, states not reachable from the new initial state should be removed.)

2.2 Equivalences of transition systems

The automatic verification of distributed systems with the help of transition systems is based on the concept of equivalence. There are many kinds of equivalences, but one of the most established is the so called *weak bisimulation equivalence*. Before we can define it, some auxiliary concepts are needed.

- Let $\mathcal{A}$ be the set of all possible actions including i.
- If $P = (S, A, T, q_0)$ is a transition system, $a \in A$ and q_1, q_2 are states in S, we write

  $$q_1 \stackrel{a}{\Longrightarrow} q_2 \ ,$$

  if there is a path

  $$q_1 = p_1 \stackrel{i}{\rightarrow} \cdots \stackrel{i}{\rightarrow} p_i \stackrel{a}{\rightarrow} p_{i+1} \stackrel{i}{\rightarrow} \cdots \stackrel{i}{\rightarrow} p_n = q_2 \ .$$

- We write $q_1 \Longrightarrow q_2$ or $q_1 \stackrel{\varepsilon}{\Longrightarrow} q_2$, if there is path

  $$q_1 = p_1 \stackrel{i}{\rightarrow} \cdots \stackrel{i}{\rightarrow} p_n = q_2 \ .$$

  Notice that n can be 1, i.e. $q_1 \Longrightarrow q_1$.

Definition 2. A relation $\mathcal{R}$ between transition systems is *a weak bisimulation*, if for every transition system pair $(P_1, Q_1) \in \mathcal{R}$ and all actions $a \in (\mathcal{A} \setminus \{i\}) \cup \{\varepsilon\}$ the following is valid:

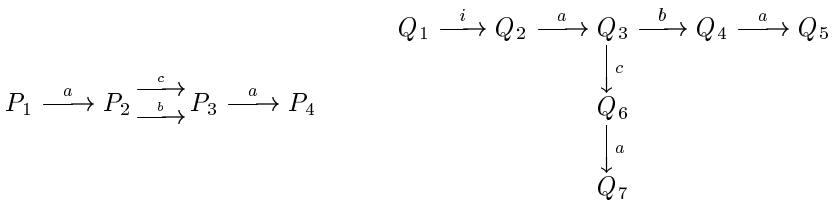
1. If $P_1 \stackrel{a}{\Longrightarrow} P_2$, then there is a Q_2 such that $Q_1 \stackrel{a}{\Longrightarrow} Q_2$ and $(P_2, Q_2) \in \mathcal{R}$.
2. If $Q_1 \stackrel{a}{\Longrightarrow} Q_2$, then there is a P_2 such that $P_1 \stackrel{a}{\Longrightarrow} P_2$ and $(P_2, Q_2) \in \mathcal{R}$.

Weak bisimilarity, $\approx_{\mathrm{wbis}}$, is defined as the largest weak bisimulation,

$$\approx_{\mathrm{wbis}} = \bigcup \{\mathcal{R} \mid \mathcal{R} \text{ is a weak bisimulation}\} .$$

Hence two transition systems (P, Q) are weakly bisimilar, if there is a weak bisimulation $\mathcal{R}$ containing the pair (P, Q).

For example, the processes



are weakly bisimilar, whereas the processes

are not.

2.3 Parallel operator

We can now model single processes as transition systems. But processes change messages in a distributed system and we want to model and analyse the whole system, not only single processes. In fact, we can model the whole system as a transition system, if we assume that actions in the processes are interleaved. This means that two actions never happen at the same time, but always one happens before the other. This assumption can be made in most distributed systems.

The description of the whole system is best achieved with the help of a parallel operator. There are several different parallel operators, but we use the one from Lotos. It is flexible and well suited for practical applications.

If P and Q are transition systems, we can combine them with the parallel operator writing

$$P \,|[a_1, \ldots, a_n]|\, Q \ ,$$

where a_1, ..., a_n are visible actions and $|[a_1, \ldots, a_n]|$ is the parallel operator. It is defined by so called transition rules. Suppose P_1 is the initial state of P and Q_1 the initial state of Q. Then

1. if $P_1 \xrightarrow{a} P_2$ and $a \notin \{a_1, \ldots, a_n\}$, then

$$P_1 \,|[a_1, \ldots, a_n]|\, Q_1 \xrightarrow{a} P_2 \,|[a_1, \ldots, a_n]|\, Q_1 \ ;$$

2. if $Q_1 \xrightarrow{a} Q_2$ and $a \notin \{a_1, \ldots, a_n\}$, then

$$P_1 \,|[a_1, \ldots, a_n]|\, Q_1 \xrightarrow{a} P_1 \,|[a_1, \ldots, a_n]|\, Q_2 \ ;$$

3. if $P_1 \xrightarrow{a} P_2$, $Q_1 \xrightarrow{a} Q_2$ and $a \in \{a_1, \ldots, a_n\}$, then

$$P_1 \,|[a_1, \ldots, a_n]|\, Q_1 \xrightarrow{a} P_2 \,|[a_1, \ldots, a_n]|\, Q_2 \ .$$

For example, if

$$\begin{array}{ccccc} & & \downarrow & & \\ P_4 & \xleftarrow{c} & P_1 & \xrightarrow{a} & P_2 \\ & & \downarrow{\scriptstyle b} & & \\ & & P_3 & & \end{array} \qquad\qquad \begin{array}{ccccccc} & & \downarrow & & & & \\ Q_1 & \xrightarrow{a} & Q_2 & \xrightarrow{c} & Q_3 & \xrightarrow{d} & Q_4 \\ \downarrow{\scriptstyle b} & & & & & & \\ Q_5 & & & & & & \end{array}$$

are two processes with initial states P_1 and Q_1, then $P \,|[a, b]|\, Q$ is the transition system

$$\begin{array}{ccccccc} & & \downarrow & & & & \\ P_4Q_1 & \xleftarrow{c} & P_1Q_1 & \xrightarrow{a} & P_2Q_2 & \xrightarrow{c} & P_2Q_3 \xrightarrow{d} P_2Q_4 \\ & & \downarrow{\scriptstyle b} & & & & \\ & & P_3Q_5 & & & & \end{array}$$

3 Partially defined transition systems

3.1 Definition of a partially defined system

Our aim is to develop methods to design processes and distributed systems step by step. For this purpose we need partially defined transition systems. These are transition systems, where transitions can be added to specially marked states.

Definition 3. *A partially defined labeled transition system* P is a tuple $(S, A, T, \uparrow, q_0)$, where

- S is a countable set of states;
- A is a countable set of actions; it can contain the special element i, the internal or invisible action;
- $T \subset S \times A \times S$ is a transition relation. We denote by T_a the set $T \cap (S \times \{a\} \times S)$ for every $a \in A$; it is supposed that $T_a \neq \emptyset$ for every $a \in A$;
- $\uparrow \subset S$ is a partiality predicate;
- $q_0 \in S$ is the initial state; we demand that every state in S is reachable from the initial state q_0 when the system is interpreted as a graph.

We write $p \xrightarrow{a} q$ instead of $(p, a, q) \in T$ and $p\uparrow$ if $p \in \uparrow$. If $p \notin \uparrow$, then we use the notation $p\downarrow$. If $p\downarrow$ for all $p \in S$, then P is *totally defined or specified*, otherwise P is *partially specified*.

Let ALTS (augmented labeled transition systems) be the set of partially defined labeled transition systems. Note that in our terminology totally defined labeled transition systems also belong to ALTS. We assume that every action symbol is a finite string constructed using a fixed finite alphabet. In addition, it is assumed that there is a countable superset of states and the state set in an arbitrary transition system is a subset of this superset. By these assumptions ALTS is countable and well defined as a set. We do not define this superset explicitly and feel free to use whatever names seem to be practical in concrete situations. The same is true for action names, but it is sometimes convenient to have a symbol for the set of all actions. Let $\mathcal{A}$ be the set of all possible action symbols in all possible transition systems. If $(S, A, T, \uparrow, q_0)$ is a labeled transition system, then $A \subset \mathcal{A}$.

3.2 Equivalence

We can modify the weak bisimulation equivalence so that it also takes partially defined states into account.

If P is a transition system, we use the notation $P\Uparrow$, if there is path

$$P = P_1 \xrightarrow{i} P_2 \xrightarrow{i} \cdots \xrightarrow{i} P_n\uparrow .$$

Definition 4. A relation $\mathcal{R} \subset \text{ALTS} \times \text{ALTS}$ is a *weak (specification) bisimulation*, if $(P_1, Q_1) \in \mathcal{R}$ implies that:

1. $P_1\Uparrow$ if and only if $Q_1\Uparrow$;
2. for all $a \in (\mathcal{A} \setminus \{i\}) \cup \{\varepsilon\}$,
   a) $P_1 \stackrel{a}{\Longrightarrow} P_2$ implies that there is a Q_2 such that $Q_1 \stackrel{a}{\Longrightarrow} Q_2$ and $(P_2, Q_2) \in \mathcal{R}$;
   b) $Q_1 \stackrel{a}{\Longrightarrow} Q_2$ implies there is a P_2 such that $P_1 \stackrel{a}{\Longrightarrow} P_2$ and $(P_2, Q_2) \in \mathcal{R}$.

Weak bisimilarity, $\approx_{\text{wbis}}$, is now defined as the largest bisimulation,

$$\approx_{\text{wbis}} = \bigcup \{\mathcal{R} \mid \mathcal{R} \text{ is a bisimulation}\} .$$

Hence two processes P, Q are *weakly bisimilar*, $P \approx_{\text{wbis}} Q$, if there is a bisimulation containing the pair (P, Q).

We have defined the weak bisimilarity using processes instead of states. But as we have explained, a state can be considered as the new initial state of a process. Thus we can also speak about the equivalence of two states in a same process and combine the equivalent states to form a state minimal process.

Proposition 1. *The relation $\approx_{\text{wbis}}$ is an equivalence relation.*

Proof. (See [10].) The claim follows from the fact that the following relations are bisimulations:

a) $\mathcal{R} = \{(P, P) \mid P \in \text{ALTS}\}$.

b) $\mathcal{R}^{-1} = \{(P_2, P_1) \mid (P_1, P_2) \in \mathcal{R}\}$, where $\mathcal{R}$ is a bisimulation.

c) $\mathcal{R} = \{(P_1, P_3) \mid (P_1, P_2) \in \mathcal{R}_1, (P_2, P_3) \in \mathcal{R}_2 \text{ for some } P_2 \in \text{ALTS}\}$, where $\mathcal{R}_1$ and $\mathcal{R}_2$ are bisimulations. □

The weak bisimulation equivalence is a congruence with respect to parallel composition.

3.3 Bisimulation refinement

The bisimulation refinement, $\sqsubseteq_{\text{wbisref}}$, (see [5]), is used to determine when one process is more specified than another. The relation is defined in terms of prebisimulations.

In the definition we use the notation $P \Downarrow a$, which means that every possible path

$$P = P_1 \xrightarrow{i} \cdots \xrightarrow{i} P_k \xrightarrow{a} P_{k+1} \xrightarrow{i} \cdots \xrightarrow{i} P_n$$

contains no partially defined states, i.e. $P_j\downarrow$, $j = 1, \ldots, n$.

Definition 5. A relation $\mathcal{R} \subset \text{ALTS} \times \text{ALTS}$ is a *prebisimulation*, if $(P_1, Q_1) \in \mathcal{R}$ implies that the following holds for all $a \in (\mathcal{A} \setminus \{i\}) \cup \{\varepsilon\}$.

1. If $P_1 \stackrel{a}{\Longrightarrow} P_2$, then there is a Q_2 such that $Q_1 \stackrel{a}{\Longrightarrow} Q_2$ and $(P_2, Q_2) \in \mathcal{R}$.

2. If $P_1 \Downarrow a$, then
   a) $Q_1 \Downarrow a$,
   b) $Q_1 \stackrel{a}{\Longrightarrow} Q_2$ implies there is a P_2 such that $P_1 \stackrel{a}{\Longrightarrow} P_2$ and $(P_2, Q_2) \in \mathcal{R}$.

By the same kind of technique as in the construction of the weak bisimulation equivalence it can be shown that the largest prebisimulation

$$\sqsubseteq_{\text{wbisref}} \;=\; \bigcup \{\mathcal{R} \mid \mathcal{R} \text{ is a prebisimulation}\}$$

is a preorder. This is the *bisimulation refinement*. It is compositional with respect to the parallel operator.

4 Process algebras

4.1 From transition systems to process algebras

It would be very cumbersome to specify large systems using only transition diagrams. It is necessary to develop some other formalism. We have seen, for example, that the parallel operator shortens descriptions radically. In the same way we can take other kinds of operators to describe transition systems.

Consider first a sequential chain of actions

$$\rightarrow P_1 \xrightarrow{a} P_2 \xrightarrow{b} P_3 \xrightarrow{c} P_4 \ .$$

We can represent this algebraically by the formula

$$a;\ b;\ c;\ \textbf{stop}$$

If P is a process and a an action, the construction $a; P$ is called *action prefix* in Lotos. The branching

$$\begin{array}{l} P_1 \xrightarrow{a} P_2 \\ \quad \downarrow b \\ P_3 \end{array}$$

can be represented by the *choice* operator [ ]:

$$a;\ \textbf{stop}\ [\,]\ b;\ \textbf{stop}$$

Sometimes we do not want to keep certain actions visible. Actions can be hid with the *hiding* operator. If P is a process, then

$$\textbf{hide}\ a_1, \ldots, a_n\ \textbf{in}\ P$$

means a process which is the same as P with the exception that the actions $a_1, \ldots, a_n$ are replaced with the silent action i. We may use the expression $h_B(P)$ as well, where $B = \{a_1, \ldots, a_n\}$.

Action prefix, choice, parallel composition and hiding are the most important operators and they are in every process algebra such as Lotos,

CCS, CSP or ACP in one form or another. We have followed Lotos conventions.

Partially defined processes are constructed with the help of the process **udef**. In what follows, we think that **udef** is part of a specification language like **stop**. Its semantics is practically the same as the semantics of **stop**, i.e. no transition is possible from **udef**. But in order to be able to separate **stop** and **udef** formally, we define that **udef** $\xrightarrow{\eta}$ **udef**. Here η is a reserved action. It represents the fact that some actions are possible from the state **udef**. The process **udef** can be combined with other processes using the operators action prefix, choice, hiding and parallel composition.

For theoretical consideration, we introduce a predicate $\uparrow$ which says whether the initial state of a process is partially defined or not. If P is a process with a partially defined initial state, we write $P\uparrow$. This means in practice that new transitions can be added to the initial state. If the initial state of P is totally defined, we write $P\downarrow$. Formally we define

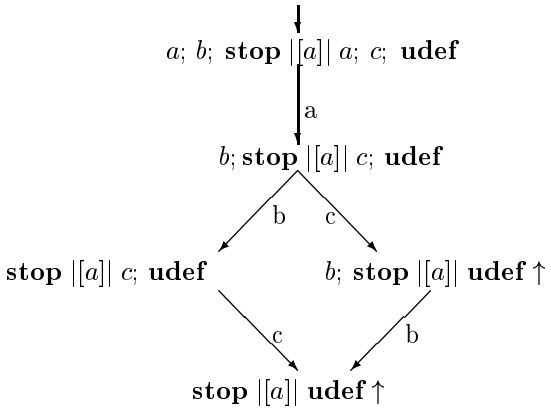
- **stop**$\downarrow$, **udef**$\uparrow$;
- if a is an action, B a set of actions, i, $\eta \notin B$, and P and Q are processes then
  1. $(a.P)\downarrow$;
  2. if $P\downarrow$ and $Q\downarrow$, then $(P[\,]Q)\downarrow$, $(P\,|B|\,Q)\downarrow$;
  3. if $P\downarrow$, then $h_B(P)\downarrow$.

These rules define when the initial state of a process is totally defined. If the initial state is not totally defined, it is partially defined.

An algebraic process expression with **udef** corresponds with a partially defined labeled transition system as follows. First we generate a labeled transition system in a normal way. Every state in this transition system corresponds with a subexpression in the original expression. We remove all the η-transitions and mark the corresponding states partially defined. For example, the expression

$$a;\ b;\ \mathbf{stop}\ |[a]|\ a;\ c;\ \mathbf{udef}$$

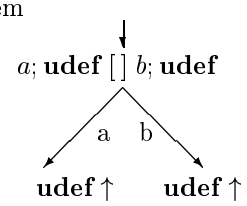
gives rise to the transition system



Consider also the process

$$a; \mathbf{udef} \,[\,]\, b; \mathbf{udef}\,.$$

It generates the transition system



Under the normal semantics this is the same as

$$a; \mathbf{udef} \,[\,]\, b; \mathbf{udef} \xrightarrow{a,\,b} \mathbf{udef} \uparrow\ .$$

This does not correspond with our intuition. In the original process definition there are two points where the refinement can start from and the two refinements can be different. On the other hand, in the latter diagram there is only one possible point of refinement. We draw the diagrams in the former way. We could define formally that every **udef** in an algebraic expression is indexed with a unique natural number. This guarantees that different **udef**-processes are not unified. We do not use indexes in diagrams, because we draw them so that **udef**-prosesses have already been separated. Furthermore, it is not necessary to use indexes in the definitions of equivalences and refinement relations, because the definitions are constructed so that they do not depend on this detail. For this reason we do not use indexes. Moreover, it would be nice, if the following was true. If $P \sqsubseteq_{\mathrm{wbisref}} Q$, then there are P' and Q' such that

$P \approx_{\text{wbis}} P'$, $Q \approx_{\text{wbis}} Q'$ and Q' is obtained from P' by adding transitions and nodes to undefined states. Unfortunately, this property is not true in bisimulation semantics, which shows that the refinement may be of a different type. However, in some other semantics the property is true, especially in some failure semantics, [9].

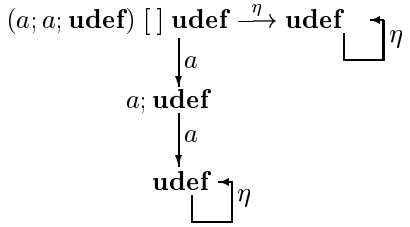
Every partially defined transition system can be transformed to a corresponding algebraic expression. Consider for example the following transition systems:

$$\begin{aligned} &\to \quad P1\uparrow \xrightarrow{a} P2 \xrightarrow{a} P3\uparrow , \\ &\to \quad Q1 \xrightarrow{a} Q2\uparrow \xrightarrow{a} Q3 , \\ &\to \quad R1 \xrightarrow{a} R2\uparrow \xrightarrow{a} R3 \xrightarrow{a} R4\uparrow . \end{aligned}$$

These correspond with the expressions

$$\begin{aligned} P &= (a; a; \mathbf{udef})[\,]\mathbf{udef} , \\ Q &= a; (a; \mathbf{stop}[\,]\mathbf{udef}) , \\ R &= a; ((a; a; \mathbf{udef})[\,]\mathbf{udef}) . \end{aligned}$$

For example, the usual transition system generated from the first expression is



4.2 Stepwise refinement

We present an example, which shows how the Alternating Bit Protocol or AB-protocol can be designed stepwise and how the wrong design decisions can be avoided at an early stage ([4]). We present the example using labeled transition systems with some operators. In this way the design is more comprehensible. A purely algebraic construction would demand more knowledge about algebraic languages than we have presented.

The AB-protocol consists of a sender and a receiver connected with a duplex channel. The sender takes data messages s from a user and sends them to the receiver. The receiver gives the received messages r to

another user. It is understood that the receiver gives every s only once to the user. The service description of the protocol is then very simple:

$$\text{ABservice} \;:=\; s; r; \text{ABservice}\ .$$

The channel can drop messages. In principle, it can also distort packets, but we think that erroneous messages are not delivered further, i.e. they are destroyed before they are given to the sender or receiver. In order to ensure that every message goes to the correct destination, s-messages are given sequence numbers 0 or 1. We denote by $d0$ an s-message with 0 and by $d1$ an s-message with 1. The receiver acknowledges the received $d0$ and $d1$ by $a0$ and $a1$, respectively. In the specification of the channel we must make a distinction between the incoming $d0$ and $d1$ and delivered $d0$ and $d1$. That is why we denote by $dd0$ and $dd1$ the incoming messages from the sender. Respectively, $aa0$ and $aa1$ are the incoming acknowledgements from the receiver. The channel C can be represented as

$$C \;:=\; C_1 \,|||\, C_2\ ,$$

where

$$C_1 \;:=\; dd0; (d0; C_1\ [\,]\ i; C_1)\ [\,]\ dd1; (d1; C_1\ [\,]\ i; C_1)$$

and

$$C_2 \;:=\; aa0; (a0; C_2\ [\,]\ i; C_2)\ [\,]\ aa1; (a1; C_2\ [\,]\ i; C_2)\ .$$

Here we have used the parallel operator $|||$, which is the same as $|[\,]|$, i.e. the synchronizing set is empty.

Now we can start to specify the AB-protocol itself. The complete system is

$$AB \;:=\; \textbf{hide}\ A\ \textbf{in}\ (S \,|||\, R)\ |B|\ C\ ,$$

where S is the sender, R is the receiver, A is the set containing all the actions in the system with the exception of s and r and

$$B = \{d0, dd0, d1, dd1, a0, aa0, a1, aa1\}.$$

Let the first designs for S and R be:

$$\begin{aligned} S_1 \quad &: \qquad \to S1\uparrow \xrightarrow{s} S2\uparrow \xrightarrow{dd0} S3\uparrow \\ R_1 \quad &: \qquad \to R1\uparrow \xrightarrow{d0} R2\uparrow \end{aligned}$$

Now

$$\textbf{hide}\ A\ \textbf{in}\ (S_1 \,|||\, R_1)\ |B|\ C \;\sqsubseteq_{\text{wbisref}}\ \text{ABservice}\ , \tag{4.1}$$

so S_1 and R_1 are correct starting points. Next we refine

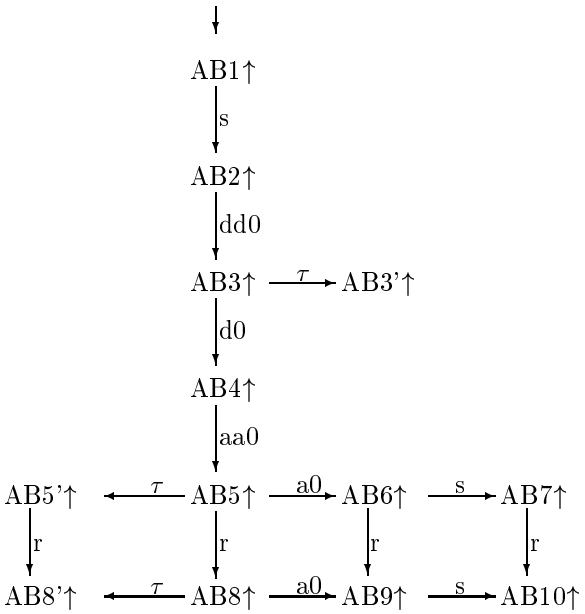
$$S_2 \quad : \quad \rightarrow S1\uparrow \xrightarrow{s} S2\uparrow \xrightarrow{dd0} S3\uparrow \xrightarrow{a0} S4\uparrow \xrightarrow{s} S5\uparrow$$

$$R_2 \quad : \quad \rightarrow R1\uparrow \xrightarrow{d0} R2\uparrow \xrightarrow{aa0} R3\uparrow \xrightarrow{r} R4\uparrow \ .$$

But now

$$\textbf{hide } A \textbf{ in } (S_2 \,|||\, R_2) \;|B|\; C \;\not\sqsubseteq_{\text{wbisref}} \text{ ABservice} \ ,$$

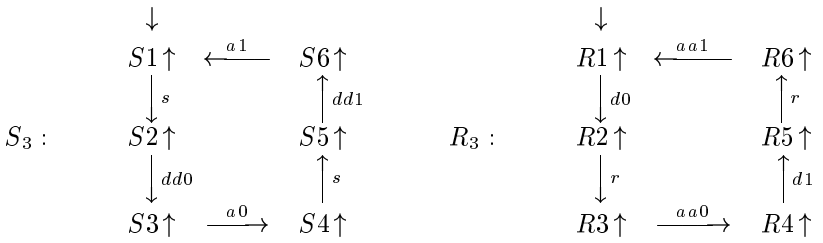
so we have made a mistake and S_2 and R_2 cannot be completed to the correct processes S and R. We can analyse the reason for the mistake, if we draw the transition graph of the whole, still partially defined, system.



Namely, s can happen two times before r happens. The reason for this is that R_2 acknowledges $d0$ before delivering r to the user. If we modify R_2 as follows

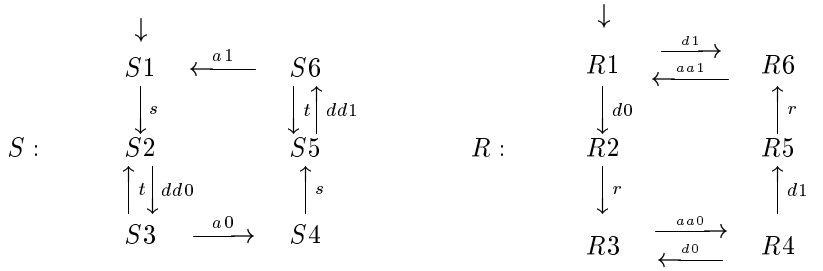
$$R_2 \ : \quad \rightarrow R1\uparrow \xrightarrow{d0} R2\uparrow \xrightarrow{r} R3\uparrow \xrightarrow{aa0} R4\uparrow \ ,$$

the anomaly disappears and we can continue to develop R_2 and S_2 into R_3 and S_3, respectively.



Now the equation (4.1) with S_1 and R_1 replaced with S_3 and R_3, respectively, is valid. Hence S_3 and R_3 describe the behaviour correctly in the case, where no transmission errors occur. Notice that the channels are totally defined and they can corrupt messages. Because the sender and receiver do not consider error situations, errors in channels lead to partially defined deadlocked states. This means that errors are handled in those parts of the sender and receiver that will be specified in the following steps.

Next we add error routines. First we remove the partial state symbol $\uparrow$ from the states $S1$, $S2$, $S4$, $S5$ and $R2$, $R3$, $R5$, $R6$ and test that the processes S_4 and R_4 thus acquired satisfy the equation (4.1). Because they satisfy, it is necessary to add more actions only to the states $S3$, $S6$ and $R1$, $R4$. The results are:



Now the whole system, AB, is totally defined and

$$AB \approx_{\text{wbis}} \text{ABservice}\,,$$

so we have completed the design.

5 Other semantics

It is possible to develop refinement relations also for other equivalences. In this section we show how to define a refinement relation for divergence bisimilarity and trace equivalence.

The weak bisimilarity can be generalized in a straightforward way to include divergences, as well. We define that a state s is divergent, if there is an infinite i-path from s. If s is a divergent state, $\mathrm{Div}(s)$ is true, otherwise false. In the next definition, we make a clear distinction between divergent and partially defined states. Our definition resembles closely the definitions in [6] and [7]. Other approaches are possible. For example, [11] considers partially defined states divergent in his definition of divergence bisimilarity.

Definition 6. A relation $\mathcal{R} \subset \mathrm{ALTS} \times \mathrm{ALTS}$ is a *divergence (specification) bisimulation*, if $(P_1, Q_1) \in \mathcal{R}$ implies that for all $a \in (\mathcal{A} \setminus \{\tau\}) \cup \{\varepsilon\}$:

1. $P_1 \Uparrow$ if and only if $Q_1 \Uparrow$,
2. $\mathrm{Div}(P_1)$ if and only if $\mathrm{Div}(Q_1)$,
3. a) $P_1 \stackrel{a}{\Longrightarrow} P_2$ implies that there is a Q_2 such that $Q_1 \stackrel{a}{\Longrightarrow} Q_2$ and $(P_2, Q_2) \in \mathcal{R}$,
   b) $Q_1 \stackrel{a}{\Longrightarrow} Q_2$ implies there is a P_2 such that $P_1 \stackrel{a}{\Longrightarrow} P_2$ and $(P_2, Q_2) \in \mathcal{R}$.

Divergence (specification) bisimilarity $\approx_{\mathrm{div}}$ is defined by

$$\approx_{\mathrm{div}} \; = \bigcup \{R \mid R \text{ is a divergence bisimulation}\} \; .$$

It is an equivalence relation and congruence with respect to the same operators as the weak bisimilarity.

If we want to get a divergence refinement, which is compositional with respect to hiding and sequential composition, we have to combine partiality and divergence in some way. We do not want to identify them as is done in [11], but instead try to find a different way to combine them.

Definition 7. A relation $\mathcal{R} \subset \mathrm{ALTS} \times \mathrm{ALTS}$ is a *divergence prebisimulation*, if $(P_1, Q_1) \in \mathcal{R}$ implies the following for all $a \in (\mathcal{A} \setminus \{\tau\}) \cup \{\varepsilon\}$:

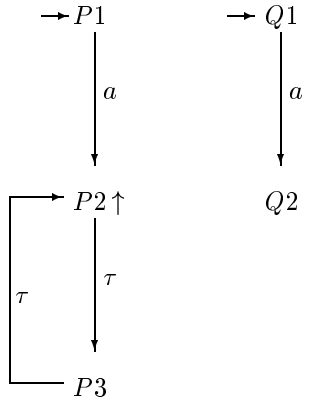
1. If $P_1 \stackrel{a}{\Longrightarrow} P_2$ then for some Q_2, $Q_1 \stackrel{a}{\Longrightarrow} Q_2$ and $(P_2, Q_2) \in \mathcal{R}$.
2. If $\mathrm{Div}(P_1)$, then $\mathrm{Div}(Q_1)$.

3. If $P_1 \Downarrow a$ then
   a) $Q_1 \Downarrow a$,
   b) $\mathrm{Div}(P_1)$ if and only if $\mathrm{Div}(Q_1)$,
   c) $Q_1 \stackrel{a}{\Longrightarrow} Q_2$ implies there is a P_2 such that $P_1 \stackrel{a}{\Longrightarrow} P_2$ and $(P_2, Q_2) \in \mathcal{R}$.

The *divergence refinement*, $\sqsubseteq_{\mathrm{divref}}$, is the largest divergence prebisimulation,

$$\sqsubseteq_{\mathrm{divref}} = \bigcup \{\mathcal{R} \mid \mathcal{R} \text{ is a divergence prebisimulation}\}.$$

The definition has two conditions for divergence, which may seem superfluous. Condition 2 guarantees that the divergence refinement behaves according to our intuition. For example, we want to exclude the case $P \sqsubseteq_{\mathrm{divref}} Q$ for the processes P and Q below:



It is intuitively not very satisfying, if a less defined process, in this example P, can contain a divergence, whereas its refinement Q contains no divergences. The condition 3.b) is needed, because we want the divergence refinement to coincide with the divergence bisimilarity on totally defined systems. In the case of convergent processes Definition 7 agrees with the bisimulation refinement.

Trace semantics is the simplest of so called failure semantics. We say that two processes P and Q are trace equivalent, if they have the same traces. A trace in P is a sequence of actions $a_1, \ldots, a_n$ such that $a_i \neq i$ and there is in P a path

$$P \stackrel{a_1}{\Longrightarrow} P_1 \stackrel{a_2}{\Longrightarrow} \cdots \stackrel{a_n}{\Longrightarrow} P_n \ .$$

Now we define the trace refinement.

Definition 8. A process Q is a trace refinement of a process P, if the following conditions are satisfied:

1. Every trace in P is also a trace in Q.
2. Every trace t in Q is either a trace in P or there is a prefix u of t such that u is a trace in P that leads to a partially defined state.

Using the same mechanism for traces as above, it is possible to develop refinement relations for more complicated failure semantics. For example, in BKO-semantics the branching structure of a process is taken into account, [2]. The CFFD-semantics is an extension of the BKO-semantics, because it also considers divergences, [12]. Refinement relations can be constructed in these semantics. The method is similar to the one for trace semantics, [9].

6 Conclusion

I have shown in this article how to verify distributed algorithms automatically step by step. The verification is usually done in the bisimulation semantics, but it can be done in other semantics as well. The examples of other semantics have been divergence bisimulation and trace semantics. This kind of approach has been used in many practical applications. The described methods have benefited especially computer network protocols and standards. There are many tools to make the use of those methods easier now. As a matter of fact, the industry seems to be interested in these methods, too.

Some problems remain. If a distributed algorithm is such that it allows an arbitrary number of processes, it is possible to show the correctness only for a fixed number of processes. In this sense, the proof is not mathematically complete. The other problem concerns infinite structures. For example, if an algorithm is such that it uses a stack without putting an upper limit to the stack size, the described automatical methods cannot handle this. The reason is that the global graph will be infinite. Infinite systems are today under study and it will be interesting to see, how general cases the automated methods can handle.

References

[1] J. Bergstra, J. Klop and E.-R. Olderog, *Failures without chaos: A new process semantics for fair abstraction.* In M. Wirsing, editor, Formal Description of Programming Concepts – III, pp. 77–101, IFIP, Elsevier Science Publishers B.V., 1987.

[2] T. Bolognesi and E. Brinksma, *Introduction to the ISO specification language LOTOS.* Computer Networks and ISDN Systems, No 14, pp. 25–59, 1987.

[3] U. Celikkan and R. Cleaveland, *Computing diagnostic information for incorrect processes.* Process Specification, Testing and Verification, XII, Elsevier Science Publishers B.V., IFIP, 1992.

[4] U. Celikkan and R. Cleaveland, *Generating diagnostic information for behavioural preorders.* Distrib. Comp., No 9, pp. 61–75, 1995.

[5] R. Cleaveland and B. Steffen, *A preorder for partial process specifications.* In J. Baeten and J. Klop (eds.), Theories of Concurrency: Unification and Extension, CONCUR'90, pp. 141–151. Lecture Notes in Computer Science 458, Springer Verlag, 1990.

[6] J. Eloranta, *Equivalence concepts and algorithms for transition systems and CCS-like languages.* Ph. Lic. Thesis C-1991-2, Department of Computer Science, University of Helsinki, 1991.

[7] J. Eloranta, *Minimal transition systems with respect to divergence preserving behavioural equivalences.* PhD thesis, Department of Computer Science, University of Helsinki, 1994.

[8] C. A. R. Hoare, *Communicating Sequential Processes.* Prentice Hall, London, 1985.

[9] T. Karvi, *Partially Defined Lotos Specifications and Their Refinement Relations*, dissertation, to be appear.

[10] R. Milner, *Communication and Concurrency.* Prentice Hall, 1989.

[11] D. Walker, *Bisimulation and divergence.* Information and Computation, No 85, pp. 202–241, 1990.

[12] A. Valmari and M. Tienari, *Compsitional Failure-Based Semantic Models for Basic LOTOS.* Formal Aspects of Computing, No 7, pp. 440–468, 1995.