Incremental Development of Deadlock-Free Communicating Systems*

Stephan Kleuker

University of Oldenburg - FB Informatik
P.O. Box 2503, 26111 Oldenburg, Germany
email: kleuker@informatik.uni-oldenburg.de

Abstract. A basic property which distributed communicating systems have to fulfill is deadlock-freedom. For systems consisting of the parallel composition of subsystems it is complex to check deadlock-freedom because the global state space of the composition has to be investigated.
This paper presents an approach by which the absence of deadlocks is preserved during the development. Small initial deadlock-free systems are stepwise extended with new functionalities to large complex systems by transformation rules which preserve deadlock-freedom. Systems are represented by finite automata extended with arbitrary local variables. A verification rule is presented which ensures that the parallel composition of such extended automata is deadlock-free. The advantage of this rule is that only information over pairs of connected subsystems is needed and not over the complete state space.

1 Introduction

Formal methods for the development of distributed systems which are developed in the academic area have to be applied to real world examples to prove or to disprove their suitability for certain application areas. Usually different application areas offer different challenges which are important in that area but are not supported by general approaches.

The presented work is motivated by the research project *Provably Correct Communication Networks* — abbreviated as CoCoN — which was carried out in close cooperation between Philips Research Laboratories Aachen and the Department of Computer Science at the University of Oldenburg from 1993 to 1996. One aim of the project CoCoN [15, 16] was to support a verified development of telecommunications systems from the requirement phase over the specification phase to an implementation. The approach was based on results of the ESPRIT Basic Research Action ProCoS [3, 4, 5, 24, 25] (Provably Correct Systems), which was a wide-spectrum verification project where embedded communicating systems are studied at various levels of abstraction ranging from requirements' capture over specification language and programming language down to the machine language. It emphasizes a constructive approach to correctness, using stepwise fully semantics preserving transformations between specifications, designs, programs, compilers, and hardware.

Telecommunications systems have in general the property that their development never terminates. New functionality (i.e. new services like call forwarding, conference

* This research was supported by the Leibniz Programme of the Deutsche Forschungsgemeinschaft (DFG) under grant No. Ol 98/1-1

calling) is added in several steps to the already running system. The possibility to extend existing systems is not treated in the ProCoS approach.

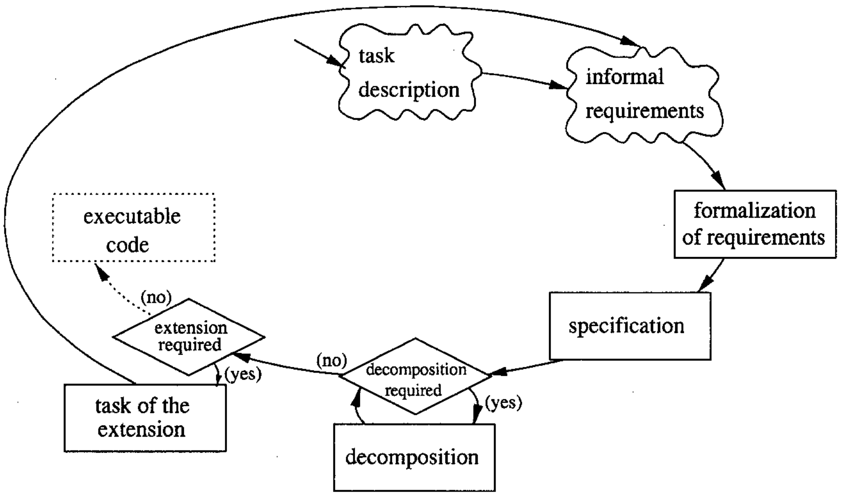


Fig. 1. Summary of the development steps

Therefore CoCoN extends ProCoS with a concept of *incremental development*. The idea is that first a small initial system is specified and developed. This system can then be extended stepwise by adding new functionalities that go beyond the specification of the initial system. As a consequence extension steps need not be any more fully semantics preserving. However, the idea of CoCoN is that specifications and proofs of properties of the initial system can be reused in the extension steps. The main development steps of the design process are sketched in figure 1. An overview of the complete method is presented in [16]. Other case studies have shown that an incremental development is also useful in the development of protocols for complex distributed systems which are not extended later on, e.g. a single track railway control for trains in both directions.

This paper focuses on the property of deadlock-freedom in the incremental development of systems. Deadlock-freedom is emphasized because once it is guaranteed developers can concentrate on the desired functionality [27]. We start with a small deadlock-free parallel composition of finite automata. By transformations presented in section 3 this initial system can be extended to rather complex systems in such a way that deadlock-freedom is preserved.

To increase their expressive power we add local variables to the automata. We present a verification rule that lifts the property of deadlock-freedom from communicating systems represented by finite automata to those represented by automata with variables. This rule needs only information about the connected subsystems, not about the global state space of the system.

Feasibility studies [15, 17, 18] have shown that the approach can be successfully applied by hand, but for more complex systems tool support is needed. We have made a prototype implementation in PROLOG for the validation and verification of specifications written in the ProCoS specification language SL. A version of the extension algorithm described in section 3 is implemented. Furthermore it is now intended to arrive at a tool with a graphical interface for the development of provably correct SL-specifications including a possibility to transform the specifications to correct code [19].

This paper is related to work on synthesis of systems [28] (see also the introduction of [8] for an overview). These approaches mainly support the development of asynchronous systems with their related problems like calculations w.r.t. the size of buffers. By contrast, the approach presented here supports systems with synchronous communication. Other approaches like [7, 14] focus on special cases like the combination of telecommunication services. The introduction of local variables in section 4 has a relation to approaches known from program verification [1]. These approaches use either the knowledge about all subsystems [22] or allow only a very simple system structure. Other approaches for the verification of distributed systems are usually done in the area of shared variables (e.g. for Unity [6, 9]) and have no or only very restricted support (superposition in Unity) for an incremental development.

The next section presents some basic definitions for communicating automata followed by the presentation of the extension approach for automata. This approach is extended in section 4 for automata with local state variables. A final discussion concludes this paper.

2 Basic notions

Initially we specify distributed systems using a parallel composition of non-terminating finite automata. Each component is described by one automaton with a designated initial state. A communication can only happen if it is possible as the next communication both for the sender component and for the receiver component (fully synchronized communication). The automaton changes its state to the next state after performing a communication. The following four definitions formalize this behavior.

Definition (automaton): A *(deterministic, non-terminating) automaton* $A = (\Delta, Q, \delta, q_0)$ consists of a finite set Δ of *communications* called the *interface*, a finite set Q of states, a partial transition function $\delta : Q \times \Delta \rightarrow Q$ which describes for a given state and a communication the next possible state, and an initial state $q_0 \in Q$. For all $q \in Q$ we require $next_A(q) \neq \emptyset$ where $next_A(q) = \{c \in \Delta | \; \delta(q,c) \text{ is defined}\}$ is the set of next possible communications in $q \in Q$. We define $\Delta(A) = \Delta$, $States(A) = Q$, $\delta_A = \delta$. □

Definition (possible traces): A *trace* is an element of Δ^*. Let ε denote the empty trace. Then the transition function δ is extended in the usual way from a single communication to traces: $\delta(q, \varepsilon) = q$, $\delta(q, t.t') = \delta(\delta(q,t), t')$, t, t' are traces. A trace t is *possible* in A iff $\delta(q_0, t)$ is defined. □

Definition (parallel composition): Let $A_i = (\Delta_i, Q_i, \delta_i, q_{0_i})$, $i \in \{1, 2\}$ be automata. The *parallel composition* $A_1 || A_2$ is defined as the automaton $A = (\Delta_1 \cup \Delta_2, Q_1 \times Q_2, \delta_A, (q_{0_1}, q_{0_2}))$ with: $\forall q_1 \in Q_1, q_2 \in Q_2, a \in \Delta_1 \cup \Delta_2 \bullet$

$$\delta_A((q_1,q_2),a) = \begin{cases} (r_1, r_2) \text{ if } a \in \Delta_1 \cap \Delta_2, \text{ and } \delta_1(q_1,a) = r_1 \wedge \delta_2(q_2,a) = r_2 \\ (q_1, r_2) \text{ if } a \in \Delta_2 - \Delta_1, \text{ and } \delta_2(q_2,a) = r_2 \\ (r_1, q_2) \text{ if } a \in \Delta_1 - \Delta_2, \text{ and } \delta_1(q_1,a) = r_1 \\ \text{undefined otherwise} \end{cases}$$

□

Remarks: Note that $||$ is symmetric and associative, i.e. we can write $A = A_1||A_2||\dots||A_n$ without brackets. It is assumed for simplicity (see also the remark at the end of section 3.4) that each communication belongs to the interface of two different automata ($\forall c \in \bigcup_{i=1}^{n} \Delta(A_i) \bullet \ |\{i|c \in \Delta(A_i)\}| = 2$). Therefore $||$ is the well-known *synchronization merge* operator of [13], here applied to n automata with point-to-point communication.

A state of A is called a *global* state of the parallel composition. For each communication $c \in \Delta(A_i) \cap \Delta(A_j)$ a function $snd(c) \in \{i, j\}$ determines the sender and $rcv(c) \in \{i, j\}$ the receiver ($snd(c) \neq rcv(c)$). Two automata A_i and A_j ($i \neq j$) are called *connected* if they have a common communication ($\Delta(A_i) \cap \Delta(A_j) \neq \emptyset$). □

The basic requirement which is emphasized in this paper is deadlock-freedom, i.e. it should always be possible for some communications to happen next. The following definition of deadlock-freedom is more restrictive because it ensures the desired property that after each possible trace t a new communication of *each* automaton of a parallel system can happen in the future.

Definition (deadlock-freedom): Let $A = A_1 \ || \dots || \ A_n$ be a parallel composition of automata with the initial state q_{0_A}. Then A is called *deadlock-free* iff

$$\forall t \bullet \ (\delta_A(q_{0_A}, t) \text{ defined} \rightarrow$$
$$(\forall 1 \leq i \leq n \ \exists t' \bullet \ (\delta_A(q_{0_A}, t.t') \text{ defined} \ \wedge \ t' \downarrow \Delta(A_i) \neq \varepsilon)))$$

The function $t \downarrow \Delta$ denotes the projection of a trace t onto an alphabet Δ. □

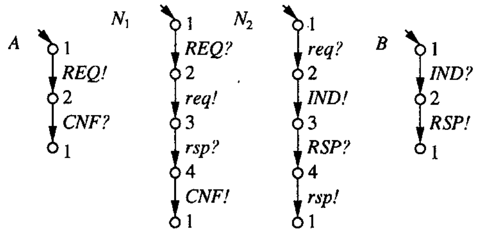


Fig. 2. Specification of a simple protocol

Example: Figure 2 describes the specification of a simple protocol. A process A asks a process B through two network nodes N_1 and N_2 in an infinite loop (the states on top and at the bottom are the same) for informations. Process A sends a request (*REQ*, a symbol *!* is used for the sender and a symbol *?* is used for the receiver [13]) to N_1 which is transmitted (*req*) to N_2 and send as an indication (*IND*) to B. Process B answers with a response (*RSP*) to N_2 which is transmitted (*rsp*) to N_1 and send as a confirmation (*CNF*) to A. □

The reader is referred to [16, 17, 18] for realistic case studies which present the application of the incremental approach in the telecommunications area.

3 Stepwise extension of specifications

Informal overview. As mentioned in the introduction, the basic idea of an incremental development technique is to come from small systems by the application of extension rules to larger systems. By *extension* we mean that new functionalities (e.g. new services, new features) are added to the system. We stipulate that each new functionality can be described by a trace which denotes the sequence of communications that should additionally be possible in a certain state of the system. The application of extension rules guarantees that this trace is indeed possible and that no deadlock occurs in the extended system.

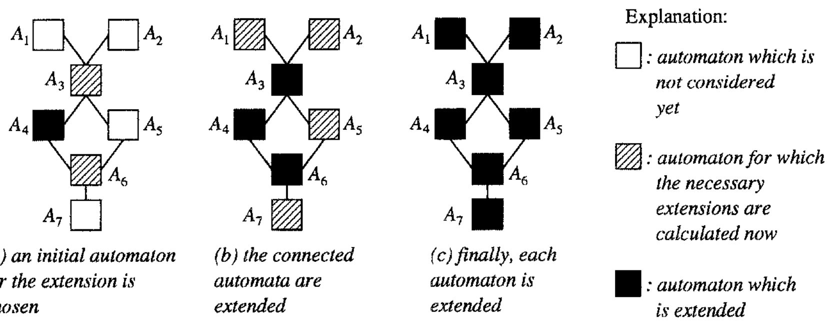

A_1
A_2
A_3
A_4
A_5
A_6
A_7
(a) an initial automaton for the extension is chosen
(b) the connected automata are extended
(c) finally, each automaton is extended
Explanation:
□ : automaton which is not considered yet
▨ : automaton for which the necessary extensions are calculated now
■ : automaton which is extended

Fig. 3. Stepwise extension of a distributed system with new functionality

The extension rules are based on an *extension algorithm* which informally works as follows. We start with a deadlock-free system, a sequence of communications which should be added, an arbitrary start state in a component (e.g. in automaton A_4 in figure 3(a)) in which this sequence should start, and a reachable global state[2] as a final state for the extension. Then, states are calculated which are *influenced* in each connected automata (figure 3(a), A_3 and A_6) separately. If a state of an automaton can be reached in the parallel composition together with the start state then such a state is called influenced. In the following steps it is calculated which states are influenced in the other automata that are connected with the already observed automata (figure 3(b)). Finally (figure 3(c)), the influenced states are calculated for each automaton and the parts of the trace which are relevant for each automaton are added in these states as transitions to the chosen final state.

The first rules of this extension technique have been described in [15]. This section presents a much more general approach w.r.t. allowed final states, necessary restrictions (the *uniqueness condition* in section 3.2 is added), generalisations (new solutions for the exception handling) and optimisations.

Formalization. We will now concentrate on the formalization to determine the necessary steps in detail and discuss the limitations. Let us assume that a system $A = A_1 \parallel \cdots \parallel A_n$, a new trace $t = c_1.c_2.\ldots.c_m$, a set $Q \subseteq States(A_1)$ denoting the desired initial states for t and a final reachable state $(f_1, \ldots, f_n)$ of A are given.

[2] Note that this is the only part in which a global information is needed. Nevertheless case studies have shown [17] that only the information about some relevant reachable global states is needed. Typical states which are reached again and again are the initial states because protocols reach these states each time when the protocol is completed. Another typical state in telecommunication protocols is reached when each participant has finished its initial phase and is connected to a conference. This active state in which information can be exchanged can be reached again and again until the conference is terminated. Global states can also be calculated by a stepwise simulation.

States that are influenced by extensions. We have to relate states such that whenever one automaton of a parallel composition is in a state q_i then a connected automaton can only be in one of the related states. A state q_i is in K-relation[3] to q_j iff (q_i, q_j) is a reachable state in the parallel composition of A_i and A_j. The definition of the K-relation can be seen as a global invariant.

Definition (K-related states):[15] Let A_i and A_j be two automata with initial states q_{0_i} and q_{0_j}. Let q_i be a state of A_i and q_j be a state of A_j. Then q_i is in *K-relation* to q_j (abbreviated $q_i\ {}^{A_i}\mathsf{K}^{A_j}\ q_j$) iff $\exists t \bullet\ \delta_{(A_i \| A_j)}((q_{0_i}, q_{0_j}), t) = (q_i, q_j)$
For $Q \subseteq States(A_i)$ we define
$\mathsf{K\_related}(Q, A_i, A_j) = \{q_j \in States(A_j) |\ \exists q_i \in Q \bullet\ q_i\ {}^{A_i}\mathsf{K}^{A_j}\ q_j\}$. □

Example: For the automata in figure 2 holds: $1\ {}^{N_1}\mathsf{K}^{N_2}\ 1$, $2\ {}^{N_1}\mathsf{K}^{N_2}\ 1$, $3\ {}^{N_1}\mathsf{K}^{N_2}\ 2$ □

The following lemma shows that the K-relation can be used to approximate the global state space with pairs of K-related states. The advantage is that we can work with pairs of local states rather than states of the global state space.

Lemma: In the state space of $A_1 \ \| \ \dots \ \| \ A_n$ a global state $(q_1, \dots, q_n)$ can be reached only if for all A_i and A_j it holds $q_i\ {}^{A_i}\mathsf{K}^{A_j}\ q_j$. □

3.1 The algorithm

States which have to be extended. Suppose automaton A_1 should be extended in the states of Q. In the first step we can calculate for the communication partner for the first communication c_1 of t which states are influenced by the desired extension. Then, we can continue this calculation for each communication of t. The following algorithm calculates step by step the set R_j of K-related states of A_j that have to be extended. A set I collects the automata for which the set of related states is already calculated (necessary restrictions for t are discussed afterwards):

Algorithm for the calculation of influenced states (extension algorithm): [15]
Input: A, A_1, t, Q as declared on the page before.
Output: $R_1 \subseteq States(A_1), \dots, R_n \subseteq States(A_n)$ sets of states of each automaton which are influenced by the desired extension.
Local variable: I is the set of automata which are already observed

```
$R_1 := Q$; $R_2, \dots, R_n := \emptyset$; $I := \{A_1\}$;
for $j = 1$ to $m$ do
    if $A_{snd(c_j)} \notin I$ then ⌈ $R_{snd(c_j)} := \mathsf{K\_related}(R_{rcv(c_j)}, A_{rcv(c_j)}, A_{snd(c_j)})$;
                                     $I := I \cup \{A_{snd(c_j)}\}$ ⌋
    elsif $A_{rcv(c_j)} \notin I$ then ⌈ $R_{rcv(c_j)} := \mathsf{K\_related}(R_{snd(c_j)}, A_{snd(c_j)}, A_{rcv(c_j)})$;
                                     $I := I \cup \{A_{rcv(c_j)}\}$ ⌋ fi fi
od
```

Possible traces for extensions. The extension algorithm assumes for each observed communication that at least one of the sender and receiver is in the set I. The reason is that the new trace describes a new path through the system initiated in certain states of one automaton which is propagated through the system. Therefore we have to guarantee that there is no communication c in $t = t_1.c.t_2$ which is totally independent from the

[3] K for the German word "Kommunikation"

automata influenced by the first part t_1. Such an independent communication would mean that two independent traces one starting in A_1 the other in the sender or receiver c are mixed. Such a mixture is not allowed for our extension algorithm. Two independent traces can be added one after another. The property that t has to fulfil can be formalized as follows:

Definition (traces that can be used for extensions): Let A and t be as described above. A trace t fulfils the *one-path-condition* (abbreviated $opc_A(t)$) for a system A iff

$$c_1 \in \Delta(A_1) \wedge (\forall 2 \leq j \leq m \bullet \{rcv(c_j), snd(c_j)\} \cap (\bigcup_{k=1}^{j-1} \{rcv(c_k), snd(c_k)\}) \neq \emptyset) \square$$

We assume $opc_A(t)$ from now on. Note that $A_{snd(c_j)} \notin I \wedge A_{rcv(c_j)} \notin I$ is always false in the extension algorithm because of $opc_A(t)$.

Extension of Automata. The next definition describes an extension of an automaton, i.e. a new trace is added between a set of initial states for the extension and a final state.

Definition (adding a trace to an automaton): Let $A_i = (\Delta, Q, \delta_{A_i}, q_0)$ be an automaton, $t = c_1 \ldots . c_q$ a trace with $c_j \in \Delta$ $(1 \leq j \leq q)$, $R \subseteq Q$ (the set of initial states) and $f \in Q$ (the final state of the new trace), let $\widetilde{z_1}, \ldots, \widetilde{z_{q-1}}$ be distinct, elements not present in Q (new states used to make t possible). Let the automaton $A'_i = (\Delta, Q \cup \{\widetilde{z_1}, \ldots, \widetilde{z_{q-1}}\}, \delta_{A'_i}, q_0)$ be derived from A_i by adding the following transitions to δ_{A_i} to get $\delta_{A'_i}$:
$\forall r \in R \bullet \ \delta_{A'_i}(r, c_1) = \widetilde{z_1}, \delta_{A'_i}(\widetilde{z_1}, c_2) = \widetilde{z_2}, \delta_{A'_i}(\widetilde{z_2}, c_3) = \widetilde{z_3}, \ldots, \delta_{A'_i}(\widetilde{z_{q-1}}, c_q) = f$
Then A'_i is called *extension* of A_i from R with t to f (short: $A'_i = ext(A_i, R, t, f)$). $\square$

3.2 Problems which can occur during an extension

Now, we analyse under which conditions the calculated states R_i can be extended with the relevant part of the new trace (i.e. $t \downarrow \Delta(A_i)$) such that deadlock-freedom is preserved. At first three restrictions are introduced which are assumed to hold in the basic theorem. Then it is shown that some of these restrictions can be dropped.

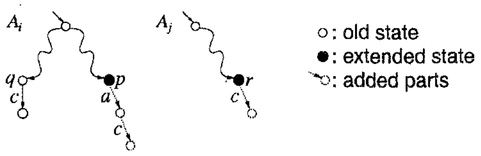


Fig. 4. critical extension which might lead to a deadlock

(i): Figure 4 sketches an extension in which it is calculated that the state p of A_i and the K-related state r of A_j should be extended. The common part of the new trace which is added in A_j is c. Furthermore, the states q and r are also K-related and a communication c is possible in q. It can now be possible after the extension that the automata are in the states q and r after a certain trace. A communication c is now executable in the extended system which might lead to a deadlock because old parts and new parts of the automata are used in a mixture in an undesired way. It must be ensured that such states like q do not exist.

Therefore the following *uniqueness condition* must be fulfilled for all connected automata A_i and A_j and the calculated sets R_i and R_j for the extension ($first(t)$ denotes the first communication of a trace):

$$\{q \in States(A_i) \mid \exists r \in R_j \bullet\ r\, {}^{A_j}\mathsf{K}^{A_i}\, q \\ \wedge\ \delta_{A_i}(q, first(t \downarrow (\Delta(A_i) \cap \Delta(A_j)))) \text{ defined}\} - R_i = \emptyset$$

Though technical, in our case studies this restriction was always fulfilled.

(ii): It might happen that a first communication of a new trace is already possible in a state which should be extended ($\exists q \in R_i \bullet\ first(t \downarrow \Delta(A_i)) \in next(q)$). In this case it is impossible to use this extension rule directly because only deterministic automata are supported. Therefore we assume that this situation never appears.

(iii): Another problem is that the result of a projection of the trace which is used for the extension might be the empty trace ($t \downarrow \Delta(A_i) = \varepsilon$). Because ε-transitions are not allowed in deterministic automata it is assumed that this situation never occurs.

3.3 Main result

Basic Extension Theorem: Let $A, t, A_1, Q, (f_1, \dots, f_n)$ be as declared above, R_j ($1 \leq j \leq n$) be the sets of states computed by the extension algorithm such that the restrictions (i), (ii) and (iii) are fulfilled, $A'_j = ext(A_j, R_j, t \downarrow \Delta(A_j), f_j)$ ($1 \leq j \leq n$), $A' = A'_1 \parallel A'_2 \parallel \dots \parallel A'_n$. Then the following holds:

(I) If A is deadlock-free then A' is, too.

(II) If a trace t' is possible in A then it is possible in A', too.

(III) If a state $q \in Q$ is reached then the trace t can be executed next

Proof:(sketch) (I): If the new trace is initiated in A_1 then each related automaton will get the possibility to work off the new trace because all K-related states are extended and therefore the information about the new trace is propagated to the whole system. No mixture of old traces and new traces is possible because of the deterministic extension and the uniqueness condition. The one-path-condition guarantees that no other automaton than A_1 can start the new trace. Therefore no new deadlocks are possible in A'. (II): Traces are only added to the old automata therefore all traces of A are possible in A'. (III): If automaton A_1 reaches a state of Q then the other automata are in K-related states. Each of these states are extended such that t can follow. □

Possible generalisations. Restriction (i) can only be slightly weakened, this possibility is omitted here due to lack of space for several necessary technical definitions.

The non-determinism in the restriction (ii) can easily be removed. Assume that a trace $c'_1.\dots.c'_k$ should be added in a state q of A_j and that c'_1 is already possible in q before the extension. Then, the trace $c'_2.\dots.c'_k$ is added in the state q' which is reached after executing c'_1 in q ($q' = \delta_{A_j}(q, c'_1)$). The only other change that is needed is that the uniqueness condition must now be checked for q' and the communication c'_2. If c'_2 is possible in q' then the removal of non-determinism is applied again. The only remaining restriction is that if the trace is reduced to ε and the desired final state f_j is not reached then the following approach for the removal of ε-transitions must be applied.

Restriction (iii) can also be weakened. We can calculate the set of influenced states independent from the trace which is used for the extension, starting with a connected automaton for which the set of influenced states is already known. If q is a state in which an ε-transition to a state f_j should be added then a transition $\delta(q, c) = p$ is added for each possible transition from the state f_j ($\delta(f_j, c) = p$). If this extension of δ

would lead to nondeterminism, then this nondeterminism has to be solved in the same way. The uniqueness condition must be checked for the state q and all possible next communications of f_j. Only if this check fails then the extension is not possible.

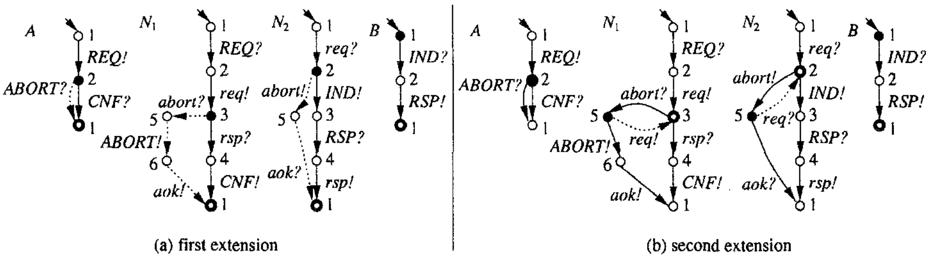


Fig. 5. Two extensions of a protocol

Example: Figure 5 presents two extensions of the protocol described in figure 2. The first extension (a) is that N_2 can answer a request with an abort communication which is transmitted to A. The new trace is $abort.ABORT.aok$ with the start state 2 of N_2, with $\{abort, aok\} \subset \Delta(N_1) \cap \Delta(N_2)$ and $ABORT \in \Delta(A) \cap \Delta(N_1)$. The reachable final state is (1,1,1,1). The influenced states are calculated in the following way: the state 3 of N_1 and the state 2 of A are calculated with the extension algorithm, the influenced state 1 of B is calculated directly from the influenced state in N_2. All side conditions are fulfilled and figure (a) presents the extensions as dotted parts.

The second extension in figure (b) is that N_1 can retry to send the information after receiving an abort communication. The added trace is req, the start state is 5 of N_1, the global reachable state is (2,3,2,1). The influenced states are state 5 of N_2, state 2 of A and state 1 of B. All side conditions are fulfilled and the figure (b) presents the resulting extensions as dotted parts. □

3.4 Optimisations

Usually one is interested in small automata which fulfil certain requirements. Therefore it is necessary to keep the number of extended states as small as possible. One optimisation is mentioned in [15] by which the number of states can be decreased for a certain process structure. More general, the sets of possibly influenced states can often be calculated in several ways. One can start with the initial automaton and can choose an arbitrary connected automaton next. If an automaton has several neighbours then it is possible to calculate the set of potentially influenced states using different paths. It is then easy to prove that the disjunction of all calculated sets of states can be used for the extension such that the extension theorem still holds.

The extension algorithm can also be applied to add automata to an existing system instead of traces. Traces are used here because case studies have shown that they are powerful enough to describe the desired new features.

Another restriction that can be dropped is that all communications must belong to two automata. A communication which belongs only to one automaton is called a *local* communication. Local communications need not to synchronize with any other automata. The only restriction which is needed is that the first communication of a new trace w.r.t. all automata $first(t \downarrow \Delta(A_j))$, $(1 \leq j \leq n)$ must not be a local communication. Otherwise it is possible to use this new path without informing the other automata and to introduce deadlocks. Multisynchronization (i.e. a communication with more than one sender and/or receiver) can be allowed without any further restriction, calculations have to be done for all involved automata for a communication.

4 Automata with state variables

The previous section described a development method based on finite automata. Although regular languages are often powerful enough to describe the typical (i.e. regular) behaviour of many communication protocols, in general complex distributed systems have to store and manipulate data. Therefore local variables are added to the finite automata to get the expressive power of Turing machines.

This section introduces automata with local variables and presents how deadlock-freedom can be proven for them using the knowledge of the previous section. A first simple condition is developed such that deadlock-freedom can be guaranteed. Then, this condition is weakened using the K-relation with the final result that the absence of deadlocks can be proven for a large range of distributed systems specified with automata with local variables.

The specification language SL [20, 21] developed in the project ProCoS can be seen as a language based on finite automata extended with local variables. In this paper only the part of SL (called here *SL-automata*) is introduced which is needed for the following calculations. Variables are declared with the following syntax:

`var` $<$ *variable_name* $>$ `of` $<$ *variable_type* $>$ `[init` $<$ *initial_value* $>$`]`

The optional part `[init` $<$ *initial_value* $>$`]` denotes the initial value of the variable.

Variables are manipulated by the execution of communications. A communication can happen only if a certain condition over the variables (the *enable predicate*) is fulfilled. After the execution of a communication a new condition (the *effect predicate*) is established. Primed variables refer to the values of the variables after the execution (similar to the Z-Notation [23]), e.g. the effect predicate $(y \neq 0 \rightarrow x' = x + 2) \wedge (y = 0 \rightarrow x' = x)$ specifies that the value of x is incremented by 2 if y is not equal to 0, otherwise the value of x is not changed.

Altogether a communication can happen if and only if (a) it is a next possible communication of the automata to which it belongs, and (b) the enable predicate of this communication is fulfilled in each automata to which it belongs.

Thus, the automata describe supersets of possible traces which are further restricted by the communication assertions.

The enable and effect predicates of a communication are summarized in a *communication assertion* of the following form:

`com` $<$ *communication_name* $>$
`when` $<$ *enable_predicate* $>$ `then` $<$ *effect_predicate* $>$

We assume that there exists only one communication assertion for each communication c and abbreviate the enable and effect predicates by $when_c$ and $then_c$.

Definition (SL-Automata): A triple $A = (A_0, Var, CA)$ is called *SL-automata* if A_0 is a finite automaton, Var is a set of variables, and CA is a set of communication assertions. The set CA contains one communication assertion for each element of $\Delta(A_0)$. The free variables of the enable predicates range over Var and the free variables of the effect predicates range over $Var \cup Var'$ with $Var' = \{v' | \ v \in Var\}$. A_0 is called the *underlying automaton* of A, $States(A) = States(A_0)$, $\Delta(A) = \Delta(A_0)$, $Var(A) = Var$. □

Additional definitions for a "well formed" SL-automaton involving type checks in the predicates are omitted here: see [21, 24]. It is assumed that for parallel compositions $A_1 \parallel \ldots \parallel A_n$ the variables are local for each component, i.e.

$$\forall 1 \leq i < j \leq n \bullet \ Var(A_i) \cap Var(A_j) = \emptyset.$$

Example: It is possible in the specification in figure 5(b) that an infinite loop of *abort.req* communications happens. Now it should be guaranteed that only three trials are made to deliver the message. A local retry counter r is introduced as a local variable in N_1, the following parts are added:

```
var r of {0,1,2,3}
com REQ then r' = 3
com req when r > 0 then r' = r − 1
com ABORT when r = 0
```

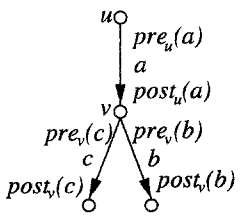
The other enable predicates are assumed to be *true* and the effect predicates are assumed to be *skip*. An effect predicate *skip* means that no value is changed. □

Our task can now be formulated as: (a) describe a method such that in each SL-automaton A at least one communication can follow after each possible trace of A (while ignoring the connected automata) and (b) guarantee that the parallel composition of the SL-automata has no deadlock.

We will see that we can use the background information that the composition of the underlying automata is deadlock-free and the K-relation to solve this task in such a way that we have to do proofs only for each connected pair of automata and not for the whole composed system.

The method explained here uses Hoare-triples of the form $\{pre\} < action > \{post\}$ as known from program verification [1, 22]. As usual the informal meaning of such a triple is "if the condition *pre* is fulfilled and $< action >$ is executed and terminates then it is guaranteed that the condition *post* is fulfilled". In [1] Hoare-triples are also used to show deadlock-freedom for distributed systems, but the calculations need information of each component. In another approach in [1] programs written in Dijkstra's guarded command language are transformed first into nondeterministic programs. This makes those results hardly applicable for complex systems.

In our approach, each communication of each automaton has to be labelled with *pre* and *post* conditions ranging over the variables of the SL-automaton and other auxiliary variables which do not occur in any other automaton. An example for such labels is presented on the right-hand side. Labels for the same communication c may differ for different occurrences of c and therefore the conditions are indexed with the name of the outgoing state.



Approaches based on the calculations of *strongest post conditions* or *weakest pre* *conditions* [10] can be applied to solve the most complicated part to get the right pre and post conditions for the automata. The relation which has to be proven between the pre and post conditions and the enable and effect predicates is, for each communication and each pre and post condition:

$$(pre_x(c) \rightarrow when_c) \quad \wedge \quad (\{pre_x(c)\}\ then_c\ \{post_x(c)\})$$

Definition: Let A be an SL-automaton and $p \in States(A)$. The set of conditions before p (denoted $\circ p$) is the set of all post conditions of communications on transitions leading to p and the set of conditions after p (denoted $p\circ$) is the set of all pre conditions of communications that can follow in p. Formally:

$$\begin{aligned} \circ p &= \{post_q(c) \mid \exists q \in States(A) \bullet \delta_A(q,c) = p\} \\ p\circ &= \{pre_p(c) \mid \exists q \in States(A) \bullet \delta_A(p,c) = q\}. \end{aligned}$$

Let SP be a set of predicates. Then $\bigvee SP$ denotes the disjunction of all predicates in SP. □

The first problem with the introduction of local variables is that if the enable predicates of an SL-automaton are too restrictive (e.g. when $false$) then it might happen that this automaton runs into a deadlock on its own. This can be avoided if each post condition before a state ensures that at least one pre condition after the state is fulfilled. This condition is formalized in the following theorem.

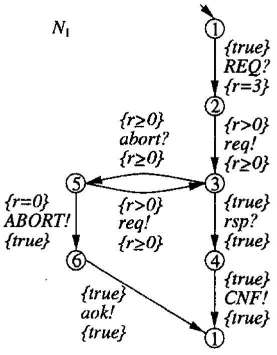
Theorem: A single SL-automaton A with initial state q_0 has no deadlock if the following two conditions are fulfilled:
(i) For the predicate $Init$ describing the initial values of the variables it holds:

$$Init \rightarrow \bigvee q_0\circ$$

(ii) For each state $p \in States(A)$ it holds: $\bigvee \circ p \rightarrow \bigvee p\circ$.

Proof:(sketch) Condition (i) guarantees that an initial communication is possible and condition (ii) guarantees that for each trace which leads to an arbitrary state p there is at least one communication which can follow. □

Example: The SL-automata A, N_2, B of the previous example have no communication assertions, therefore the pre and post conditions can be chosen as $true$. The pre and post conditions for N_1 are shown on the right-hand side. Effect predicates then *skip* do not influence the values of the variables. The only non-trivial implications which have to hold are for the state 2: $r = 3 \rightarrow r > 0$ and for the state 5: $r \geq 0 \rightarrow r = 0 \vee r > 0$. □



If two SL-automata without deadlocks are composed it might happen that the composition has deadlocks even if the composition of the underlying automata is deadlock-free. The reason is that in alternatives different communications might be enabled. Our solution uses the K-relation to introduce criteria which guarantee that the composition is deadlock-free. The advantage of these criteria is that only pairs of connected SL-automata have to be checked instead of the whole parallel composition. Note that there are deadlock-free parallel compositions that do not fulfil the criteria.

The following theorem states that a parallel composition of SL-automata is deadlock-free if the composition of the underlying automata is deadlock-free and if it is guaranteed for two K-related states of connected automata that at least one common communication is enabled.

Theorem: Let A be a parallel composition of SL-automata with the underlying automata $A_1, \ldots, A_n$. Let the parallel composition of the underlying automata be deadlock-free. Then A is deadlock-free if for all connected automata A_i and A_j and for all states $p \in States(A_i)$ and $q \in States(A_j)$ with $p \, {}^{A_i}\mathsf{K}^{A_j} \, q$ the following holds:

$$\text{(C)} \quad \exists c \in \Delta(A_i) \cap \Delta(A_j) \bullet \; ((\bigvee \circ p \rightarrow pre_p(c)) \;\wedge\; (\bigvee \circ q \rightarrow pre_q(c))) \qquad \square$$

The condition (C) is sufficient, but not necessary, as it can be seen for the composition in figure 5(b). It is impossible to fulfil (C) for the state 2 of A because the condition is not fulfilled for the K-related state 2 of N_1, i.e. no common communication is enabled.

Note that the information that the composition of the underlying automata is deadlock-free is not used in the condition (C). An improvement is that condition (C) needs to be proven only for K-related states which fulfil the following condition:

$$(\alpha) \qquad next_{A_i}(p) \cap next_{A_j}(q) \neq \emptyset$$

If in a certain K-related state no communication with the connected automaton can follow then the next communication is completely dependent on the other automata. Therefore (α) ensures that (C) must only hold if a dependency between A_i and A_j w.r.t. p and q is given. The check that (α) holds can be done fully automatically without any knowledge of the pre and post conditions. The previous theorem can now be refined in the following way.

Theorem: Let A be a parallel composition of SL-automata and let the parallel composition $A_1 \parallel \ldots \parallel A_n$ of the underlying automata be deadlock-free. Then A is deadlock-free if the following holds:

$$\forall i, j \; \forall p \in States(A_i), q \in States(A_j) \bullet$$
$$(i \neq j \;\wedge\; \Delta(A_i) \cap \Delta(A_j) \neq \emptyset \;\wedge\; p \, {}^{A_i}\mathsf{K}^{A_j} \, q \;\wedge\; (\alpha)) \;\rightarrow\; (C)$$

Proof: Assume that there is a deadlock in a global state $(p_1, \ldots, p_n)$. If we take an arbitrary automaton A_k then (α) (w.r.t. $p_k \in States(A_k)$) does not hold for any automaton which is connected with A_k, otherwise (C) would guarantee that a communication can follow. Therefore, we can conclude that there is a deadlock in the composition of the underlying automata in p_k which is a contradiction to the assumption. $\square$

Several calculations in a case study for a protocol for a multiuser multimedia system [17] have shown that the new condition is powerful enough for real applications. It is also possible to prove the absence of deadlocks for the small protocol presented in this paper.

If the extension of the automata as described in section 3 is used for SL-automata then it is possible to reuse the pre and post conditions. Additional conditions only have to be found for the states of the added trace. A more detailed theory about the reuse of conditions is under development.

5 Conclusion and final remarks

The previous sections present an approach for the development of large deadlock-free systems in small steps. The verification that a system is deadlock-free needs to be done only for the initial system. If this system is then extended with the above extension rules deadlock-freedom is preserved. It is shown how deadlock-free automata can be used to come to deadlock-free SL-automata with arbitrary local variables. The advantage of the approach is that no knowledge of the global state space is needed to guarantee deadlock-freedom, only calculations for pairs of systems (connected automata) have to be done.

Commercially available tools like STATEMATE [12] for statecharts [11] and SDT [26] for the asynchronous language SDL [2] have to investigate the complete state space to guarantee deadlock-freedom. This paper proposes that such a property should be guaranteed throughout the development process instead of investigating the complete state space.

The property of deadlock-freedom is emphasized in this paper because its hard to prove and, more important, once deadlock-freedom is guaranteed then the developer can concentrate on the real tasks, i.e. obtaining the desired functionality of the system.

The method of labelling communications in SL-automata with local pre and post conditions can also be useful for the verification of other properties. This is a topic for further research. Another topic is the exchange of values with communications. The specification language SL allows that values of a fixed type can be transmitted during a communication. The value is referred to as $@c$ for a communication c. These communication variables can also be variables of the pre and post conditions and used for the calculations described in section 4.

The described technique has been successfully applied by hand to some communication protocols in CoCoN [15, 18]. A feasibility study with a prototype implementation of this approach in PROLOG has shown that the necessary calculations for the extension of automata can be done very efficiently. An important aspect is that the K-relation only needs to be calculated once and can then be stored in a database. Each extension adds some informations to this database, no information needs to be changed. The whole extension approach with optimisations will be implemented in a graphical workbench [19] for the verification and validation of distributed systems.

Acknowledgments. The author thanks H. Tjabben of Philips Research Laboratories Aachen and E.-R. Olderog and the other members of the 'semantics' group in Oldenburg for helpful discussions.

References

1. K.R. Apt, E.-R. Olderog, Verification of Sequential and Concurrent Programs, Springer, New York, 1991
2. F. Belina, D. Hogrefe, The CCITT-Specification and Description Language SDL, Computer Networks and ISDN Systems 16 (1988/89) 311-341, North-Holland
3. D. Bjørner, H. Langmaack, C.A.R. Hoare, ProCoS I Final Deliverable, ProCoS Technical Report ID/DTH db 13/1, January 1993
4. D. Bjørner et al., A ProCoS project description: ESPRIT BRA 3104, Bulletin of the EATCS, 39, 1989
5. J.Bowen et al., Developing Correct Systems, 5th EuroMicro Workshop on Real-Time Systems, Oulu, Finland, 1993, IEEE Computer Society Press
6. K.M.Chandy, J. Misra, Parallel Program Design, Addison-Wesley, 1988

7. K.E. Cheng, Towards a Formal Model for Incremental Service Specification and Interaction Management Support, in L.G. Bouma, H. Veltheuijsen (Eds.), Feature Interactions in Telecommunications Systems, IOS Press, 1994
8. D. Y. Chao, D. T. Wang, An Interactive Tool for Design, Simulation, Verification, and Synthesis of Protocols, Software - Practice and Experience, Vol. 24(8), 1994
9. P. Collette, E. Knapp, Logical Foundations for Compositional Verification and Development of Concurrent Programs in UNITY, in V.S. Alagar, M. Nivat (Eds.), Proc. Algebraic Methodology and Software Technology '95, LNCS 936 (Springer), 1995
10. E.W. Dijkstra, Guarded Commands, Nondeterminacy and Formal Derivation of Programs, Communications of the ACM, 18:453-457, 1975
11. D. Harel, Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming 8, 1987
12. D. Harel et al., STATEMATE: A Working Environment for the Development of Complex Reactive Systems, IEEE Transactions on Software Engineering, Vol. 16, No. 4, 1990
13. C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, London, 1985
14. A. Khoumsi, Detection and Resolution of Interactions between Services of Telephone Networks, internal report IRO 1037, University of Montreal, 1996
15. S. Kleuker, A Gentle Introduction to Specification Engineering Using a Case Study in Telecommunications, in P. D. Mosses, M. Nielsen, M. I. Schwartzbach, eds., Proc. TAPSOFT '95, LNCS 915 (Springer), 621-636,1995
16. S. Kleuker, H. Tjabben, The Incremental Development of Correct Specifications for Distributed Systems, in M.-C. Gaudel, J. Woodcock (eds.), Proc. FME '96, LNCS 1051 (Springer), 1996
17. S. Kleuker, H. Tjabben, A Formal Approach to the Development of Reliable Multi-User Multimedia Applications, Philips Research Laboratories Aachen, Technical Report, 1168/96, ftp://ftp.informatik.uni-oldenburg.de/pub/procos/cocon/mumu.ps.Z
18. S. Kleuker, Using Formal Methods in the Development of Protocols for Multi-user Multimedia Systems, in R. Gotzhein und J. Bredereke (eds.), Proc. of FORTE/PSTV'96, Chapman & Hall, 1996
19. B. Krieg-Brückner, J. Peleska, E.-R. Olderog, D. Balzer, A. Baer, UniForM: Universal Formal Methods Workbench, in Statusseminar des BMBF, Softwaretechnologie, Berlin, March 1996
20. E.-R. Olderog, Towards a Design Calculus for Communicating Programs, LNCS 527 (Springer), 61-77, 1991
21. E.-R. Olderog et al., ProCoS at Oldenburg: The Interface between Specification Language and OCCAM-like Programming Language. Technical Report, Bericht 3/92, Univ. Oldenburg, Fachbereich Informatik, 1992
22. S. Owicki, D.Gries, An Axiomatic Proof Technique for Parallel Programs, Acta Informatica, 6:319-340, 1976
23. J.M. Spivey, The Z Notation: A Reference Manual, Prentice Hall International Series in Computer Science (2nd edition), 1992
24. S. Rössig, A Transformational Approach to the Design of Communicating Systems, PhD thesis, University of Oldenburg, 1994
25. S. Rössig, M. Schenke, Specification and Stepwise Development of Communicating Systems, LNCS 551 (Springer), 1991
26. Telelogic AB, Malmo, Sweden, SDT 3.01: Users' Guide, 1995
27. M. Weske, Deadlocks in Computersystemen (in German), International Thomson Publishing, 1995
28. P. Zafiropulo et al., Towards Analyzing and Synthesizing Protocols, IEEE Transactions on Communications, Vol COM-28, No. 4, April 1980