P3

Verification of Liveness Properties Using Compositional Reachability Analysis

Shing Chi Cheung[†] **Dimitra Giannakopoulou**[‡] **Jeff Kramer**[‡]

[†]Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong.

[‡]Department of Computing, Imperial College of Science, Technology and Medicine, London SW7 2BZ, UK.

Email: scc@cs.ust.hk, {dg1, jk}@doc.ic.ac.uk

ABSTRACT

The software architecture of a distributed program can be represented by a hierarchical composition of subsystems, with interacting processes at the leaves of the hierarchy. Compositional reachability analysis (CRA) is a promising state reduction technique which can be automated and used to derive in stages the overall behaviour of a distributed program based on its architecture. Conventional CRA however has a limitation. The properties available for analysis after composition and reduction are constrained by the set of actions that remain globally observable. The liveness properties which involve internal actions of subsystems may therefore not be analysed. In this paper, we extend compositional reachability analysis to check liveness properties which may involve actions that are not globally observable. In particular, our approach permits the hiding of actions independently of the liveness properties that are to be verified in the final graph. In addition, it supports the simultaneous checking of multiple properties (both liveness and safety), and identifies those properties that are violated. The effectiveness of the extended technique is illustrated using a case study of a Reliable Multicast Transport Protocol (RMTP) with over 96,000 states and 660,000 transitions.

Keywords

Reachability analysis, compositional verification, distributed computing systems, labelled transition systems, Büchi automata, liveness properties.

1 Introduction

Distributed processing is widely used to provide computing support for diverse applications. Many of these applications are complex and critical; an error can have catastrophic consequences. Behaviour analysis is a useful technique that can help discover defects and check if a program performs as intended.

Static analysis techniques for concurrent and distributed programs can be used to verify two classes of property: *safety* and *liveness.* A *safety* property asserts that the program never enters an undesirable state [1]. For example, mutual exclusion is a safety property which specifies the absence of a program state where a common resource is simultaneously accessed by more than one client. A *liveness* property asserts that a program eventually enters a desirable state [1]. For example, the assertion that a program will eventually close a file after opening it is a liveness property.

In this paper, we focus our discussion on liveness properties which can be specified in terms of *Büchi automata* - finite state machines that accept infinite words [2]. These machines will be referred to as *property automata.* Each property automaton specifies the set of acceptable execution sequences in terms of actions that correspond to a liveness property of interest. For example, the property automaton in Figure 1 asserts the liveness property that a write request will eventually be granted. This is because the automaton accepts an infinite word w if and only if the execution of the automaton on w contains the accepting state 0 an infinite number of times. Therefore, this automaton accepts the language (write_request* request_grant)$^{\omega}$, where juxtaposition represents concatenation, and the operators * and $^{\omega}$ denote finite and infinite repetition accordingly.

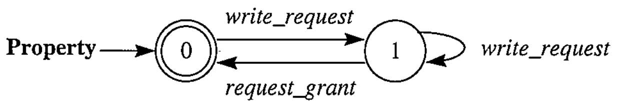

Figure 1: A Property Automaton

However, distributed programs are generally complex to analyse. Even for small programs, analysis of their behaviour is tedious without the support of an *effective automated* technique. One approach is to perform analysis in a compositional manner, thus exploiting the design structure (software architecture) of the distributed program [3]. This can be represented by a hierarchical composition of subsystems, with interacting (primitive) processes at the leaves of the hierarchy. Behaviour of a primitive process can be modelled as a *state machine* whose transitions are labelled by the activities it can perform. *Composite processes* appear at the nodes of the hierarchy. Each composite process is a subsystem formed by a collection of processes that can be either primitive or composite. The behaviour of a composite process is derived by composing the behaviours of its immediate children in the hierarchy. Details of the subsystem that are internal to it are then hidden. A minimal state-machine is generated for its abstracted behaviour, corresponding to the behaviour of the subsystem visible to its environment. The global system behaviour is obtained in this way, and can be used for verification.

Promising results have been reported from the use of *Compositional Reachability Analysis (CRA)* to generate a state space graph for a well-structured distributed program [4-7]. Yeh [8] described several case studies which suggested similar performance between a technique of compositional reachability analysis and that of constraint expressions [9]. Sabnani et al. [6] described an experiment applying compositional reachability analysis to the Q.931 protocol. They found that the intermediate state space graphs generated never exceeded 1,000 states although the global state space graph given by traditional reachability analysis of the protocol contained over 60,000 states. Similar observations have also been made by Tai and Koppol [7]. CRA is particularly suitable for analysing properties of programs which are likely to evolve. It helps localise the effect of change. When changes are applied to a program, only the properties involving those subsystems that are affected by the changes need be re-computed.

The CRA technique however has a limitation. The properties available for analysis after minimisation are constrained by the set of actions that remain globally observable. This poses a severe problem if properties to be checked involve a large set of actions that are not globally observable. Previous work [10] has proposed a mechanism to address this problem during the checking of safety properties. In this paper, we extend the CRA technique with a mechanism for the checking of liveness properties. The extension allows multiple liveness properties to be validated simultaneously. Liveness properties are violated when some subsystems, within the context of a distributed program, can perform execution sequences not acceptable to the specified property (Büchi) automata. If no violation of liveness properties is detected, the analysis constructs a global LTS *observationally equivalent* [11] to that constructed using conventional CRA techniques; otherwise it indicates which and how liveness properties are violated.

As mentioned, we have adopted the approach which expresses properties as Büchi automata. Büchi automata can be used to express formulae of linear time temporal logic [12]. Fernandez et al. propose a technique to compose the property automata with the system [13]. Godefroid and Holzmann [14] compute the product automaton of the specifications of the system with a Büchi automaton for the negation of a formula of interest. Verification then reduces to checking if the product automaton accepts only the empty set. A similar approach has been proposed by Aggarwal et al. [15]. Their work extends the selection/resolution (S/R) model with acceptance states, adding to it the expressiveness of Büchi automata. However, the issues of compositionality and hiding of internal actions are not addressed in any of the above works.

Recently Bultan et al. have proposed a method for performing compositional analysis of temporal properties expressed in the branching time logic ∀CTL [16]. The method generates counterexamples using a compositional approach. Branches of the intermediate graphs are pruned if they do not provide potential counterexamples for the property under verification. All actions are assumed to be globally observable in the method. The issues of hiding internal actions and incorporating the checking mechanism into the framework of compositional reachability analysis have not been addressed. Moreover, their method can handle a single property at a time. Every change introduced into the system when a violation is detected requires rechecking of all the system properties one by one.

The rest of this paper is structured as follows. Sections 2 and 3 introduce labelled transition systems and present a reliable multicast transport protocol (RMTP) which is used as a case study in our discussion. Section 4 describes compositional reachability analysis and its limitations. Section 5 proposes a technique to overcome these limitations. The technique detects and locates violation of liveness properties related to subsystems. This is followed by a comparison of experimental results and conclusions in Sections 6 and 7, respectively.

2 Labelled Transition Systems

A *labelled transition system* (LTS) can be used to model the behaviour of a synchronous communicating process in a distributed program. An LTS contains all the states the process may reach and all the transitions it may perform. The model has been widely used in the literature for specifying and analysing distributed programs [17-21]. In the model, communicating processes are synchronised through actions sharing the same labels. For example, let a represent the action in which a machine in a flexible manufacturing system transfers a part to a conveyor belt. The action a occurs only if the machine is ready to hand over the part, and the conveyor belt is simultaneously prepared to receive the part. In terms of LTS, a is modelled as a possible action in the standalone behaviour of both processes. Its execution then requires simultaneous participation from both processes. Formally, an LTS of a process P is a quadruple $< S, A, \Delta, p >$ where

(i) S is a set of states;
(ii) $A = \alpha P \cup \{\tau\}$, where αP is the communicating *alphabet* of P which does not contain the internal action τ;
(iii) $\Delta \subseteq S \times A \times S$, denotes a transition relation that maps from a state and an action onto another state;
(iv) p is a state in S which indicates the initial state of P.

An LTS of $P = < S, A, \Delta, p >$ transits into another LTS of $P' = < S, A, \Delta, p' >$ with an action $a \in A$ if and only if $(p, a, p') \in \Delta$. That is,

$$< S, A, \Delta, p > \xrightarrow{a} < S, A, \Delta, p' > \text{ iff } (p, a, p') \in \Delta.$$

Since there is a one-to-one mapping between a process P and its LTS, we use the terms process and LTS interchangeably. Processes in a distributed program may be composed by operator $\|$, which has similar semantics to those of the composition operator used in CSP [22]. $P1 \| P2$ is the parallel composition of processes $P1$ and $P2$ with synchronisation of the actions common to both of their alphabets and interleaving of the others. Observability of actions in a process can be controlled by a restriction operator $\uparrow$. $P\uparrow L$ represents the process projected from P in which actions in A-L are replaced by the internal action τ.

Finally, a liveness property is expressed as a Büchi automaton $B = < S, A, \Delta, q_0, F >$, where S is a finite set of states, A is a set of observable actions, Δ is a set of transitions, q_0 is its initial state, and F is a set of acceptance states. An execution of B on an infinite word w = $a_1a_2a_3$□over A is an infinite sequence $\sigma = q_0q_1q_2...$ of elements of S, where $(q_{i-1}\ a_i\ q_i) \in \Delta$ for every $i > 0$. An execution of B is accepting if it contains some acceptance state of B an infinite number of times. A word w is accepted by B if there exists an accepting execution of B on w.

3 The Reliable Multicast Transport Protocol (RMTP)

To illustrate our approach, we present a Reliable Multicast Transport Protocol (RMTP) as proposed by Lin and Paul [23]. The protocol is designed for applications that cannot tolerate data loss. It provides sequenced, lossless delivery of data from a sender to a group of receivers, at the expense of delay. Reliability is achieved by a periodic transmission of acknowledgement by the receivers (*ACK* packets) and a selective retransmission mechanism by the sender. Scalability is provided by grouping receivers into a hierarchy of local regions, with a *Designated Receiver (DR)* in each of those regions. Receivers in each local region send their ACKs to the corresponding DR, DRs send their ACKs to the higher level DRs or to the sender (see Figure 2), thereby avoiding the ACK-implosion problem. In addition, DRs cache received data and respond to receivers in their local regions, thus decreasing end-to-end latency. The term *Acknowledgement Processor (AP)* is used to denote either a DR or the sender, when referring to them as entities that receive and process ACKs. Receivers which are not designated receivers are referred to as *ordinary receivers.*

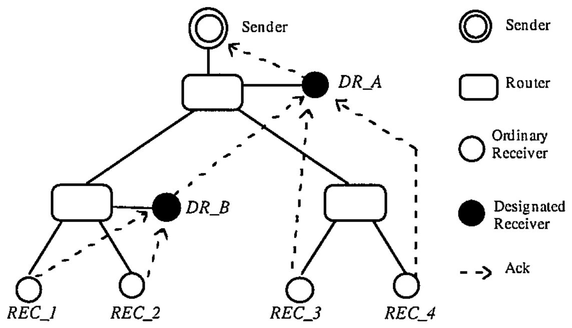

Figure 2: A Multicast Tree of Receivers

To cater for situations where DRs may fail, receivers use a mechanism to dynamically select the nearest operational AP in the multicast tree. This is the part of the RMTP protocol that our case study focuses on. Dynamic selection of APs is achieved in RMTP by the use of a special packet, called the SND_ACK_TOME (SAT) packet. The sender and all DRs periodically advertise themselves (action *adv*) by multicasting SAT packets along their subtrees. The SAT packets are tagged with the same initial *time-to-live* (TTL) values. Routers decrement the TTL value when forwarding packets. Therefore a larger TTL value indicates a closer proximity in the multicast tree. On receiving an SAT packet, a receiver compares the TTL value associated with the incoming packet with that associated with the AP currently selected. The receiver switches to a new AP if the incoming packet has a larger TTL value. When a receiver fails to receive a new SAT packet from the currently selected AP after a certain period of time, it assumes failure of the AP and initiates another selection cycle.

In our case study, we have modelled this part of the protocol for the configuration depicted in Figure 2. Three processes are associated with both ordinary and designated receivers in the multicast tree, namely the Receiver, Channel, and Watch processes.

Let us consider an ordinary receiver REC_1 in Figure 2 as an example. Figure 3 presents the configuration diagram of REC_1 that encapsulates three processes. Communicating actions take place where portals of components (represented as grey dots) are bound together. A portal is an interface instance and has a type that is simply a set of names that refer to actions or events shared between bound components. Interface types in the diagram are defined as follows:

```
interface dr_info {ms_fail; selA; selB; selS;}
interface dr_mes {mesA; mesB; mesS;}
interface fails {failA; failB;}
interface advs {advA; advB; advS;}
```

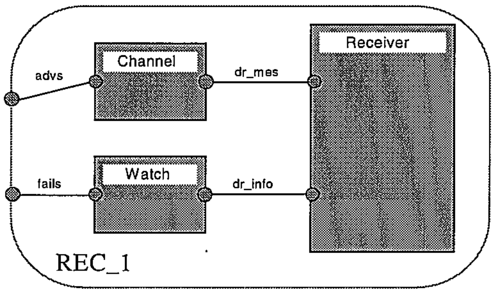

Figure 3: The Configuration Diagram of Subsystem REC_1

As illustrated in the configuration diagram, REC_1 interacts with other entities in the multicast tree through interfaces *advs* and *fails*, consisting of actions *advA*, *advB*, *advS*, *failA*, and *failB*. For convenience, we use *actX/Y/Z* to stand for the set of actions *actX*, *actY* and *actZ* which share the same prefix *act* in their labels. The behaviour of REC_1 is given by the composite behaviour of its three constituent processes described in terms of LTS as in Figure 4.

The Channel process models a lossy channel, which receives advertisements from the APs above the receiver (actions *advA/B/S*), and transmits them to the Receiver process (actions *mesA/B/S*), or loses them (action *lose*). The specification assumes fair execution in the sense that unfair execution sequences where the Channel keeps losing all messages are refused. The Watch process models the time-out associated with the selection of a new AP. It observes all potential APs for the receiver, and when a failure of the selected AP occurs (actions *failA/B*), it informs the Receiver (actions *ms_fail*) so that the selection procedure is initiated. The receiver then selects as its AP (action *selA/B/S*) the AP whose advertisement it receives first. Selections are modified whenever an advertisement is received from a nearer AP than the one currently selected. In the composite behaviour of the Receiver, Channel and Watch processes, only actions *failA/B*, and *advA/B/S* synchronise with the environment of REC_1, so all the remaining actions can be made unobservable, i.e.,

REC_1 = (Channel || Watch || Receiver) $\uparrow$ {*failA/B, advA/B/S*}.

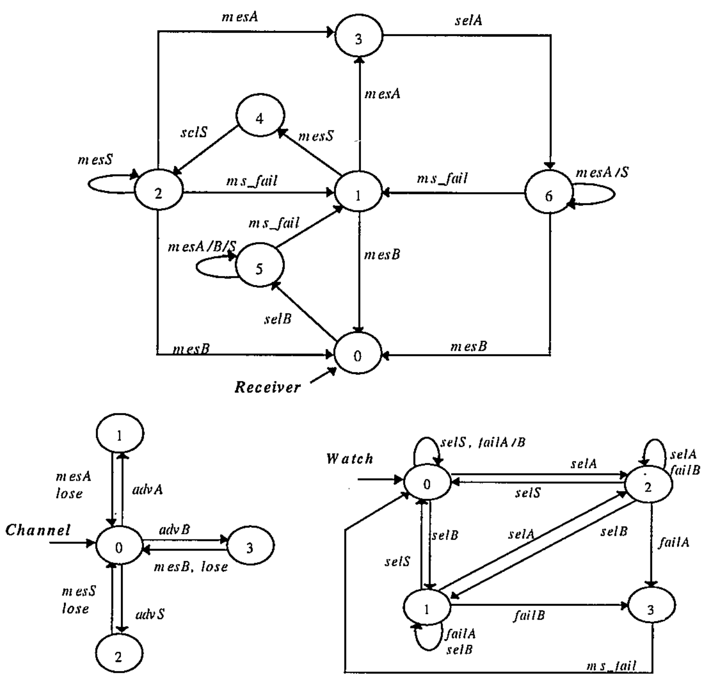

Figure 4: LTS of the Receiver REC_1

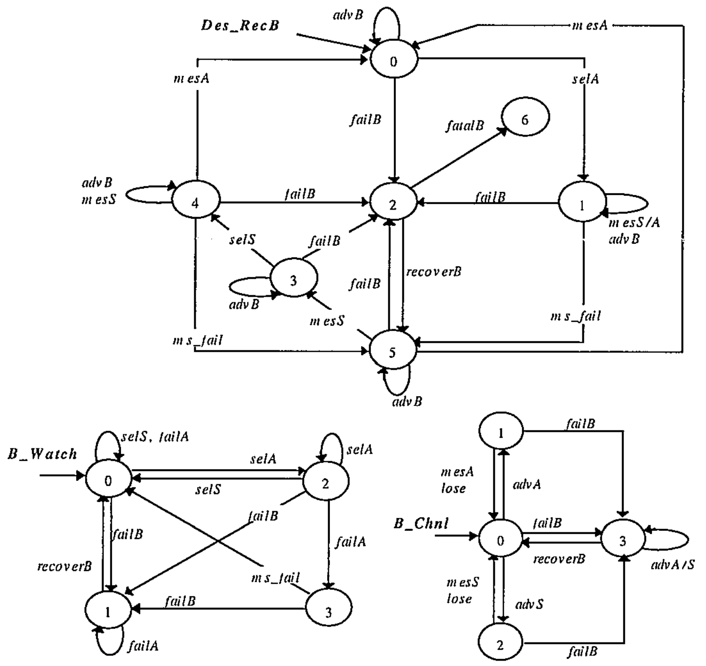

Figure 5: LTS of the Designated Receiver DR_B

In Figure 5 we illustrate the behaviour of designated receiver DR_B. DR_B has also been specified in terms of three components. A DR behaves like a receiver, except that it may fail and that it advertises itself. DR_B may fail at any time *(failB)*, and enter a state where it stops advertising itself. From this state it may either fatally fail *(fatalB)* or recover *(recoverB)*. All actions in DR_B that do not synchronise with its environment can be made unobservable, i.e.,

$$DR\_B = (B\_Chnl \parallel B\_Watch \parallel Des\_RecB) \uparrow \{\mathit{failA/B},\ \mathit{advA/B/S}\}.$$

Note that we have not modelled failure for ordinary receivers and the sender. If the sender fails, the multicast session is cancelled, in which case RMTP need not fulfil its objectives. Properties on the receivers are not expected to hold when they fail. Moreover, failures of ordinary receivers do not affect the behaviour of their environment, and may therefore be ignored.

Routers have not been specified as separate processes because our model directly supports multicast by the synchronisation of actions common in the process alphabets. Finally, in our experiments we have not taken into account the behaviour of receivers REC_2 and REC_4, since their behaviour for this part of the protocol is identical to the behaviour of REC_1 and REC_3, respectively.

We have used this case study to verify two liveness properties (Figure 6). Property Liv_LocRegl concerns the local region that has DR_B as its designated receiver. This property is intended to check that the dynamic selection mechanism of the protocol ensures that whenever DR_B recovers from failure *(recoverB)* and does not fail again *(failB)*, it will eventually be the selected AP of all receivers in its local region. In our case study, the latter corresponds to checking that *(selB)* will eventually be performed by REC_1. Since we want to be able to verify this property for the local region, we need to postpone the hiding of actions *selB* and *recoverB* of components REC_1 and DR_B respectively until the next level in the compositional hierarchy, as seen in Figure 10. Property LivRec3 refers to REC_3, and asserts that whenever DR_A is its selected AP and DR_A fails (action *ms_fail* synchronised with its corresponding watch process - Watch3 in Figure 10 - reflects this fact), REC_3 will eventually select a new AP.

We will now proceed to show how the above liveness properties of the protocol can be effectively validated using an enhanced compositional reachability analysis technique.

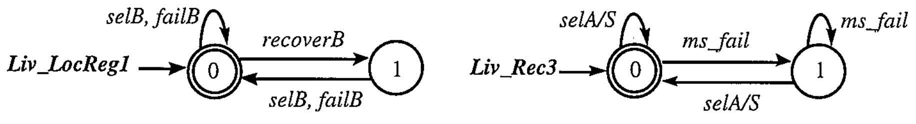

Figure 6: Liveness Properties as Property Automata

4 Compositional Reachability Analysis and Its Limitations

Promising results have been reported in the literature on the use of a compositional approach to derive the overall system behaviour using reachability analysis [5-7]. In compositional reachability analysis (CRA) techniques, the model of the target system is given as an LTS that describes an abstraction of the system behaviour, according to the requirements of the user. Figure 7 gives an LTS prescribing the abstracted behaviour of designated receiver DR_A[1]. Action *A.selS* in the figure represents the *selS* in designated receiver DR_A. The LTS indicates that the Sender is the only AP selected by DR_A. The selection is performed voluntarily by DR_A upon its recovery from failure.

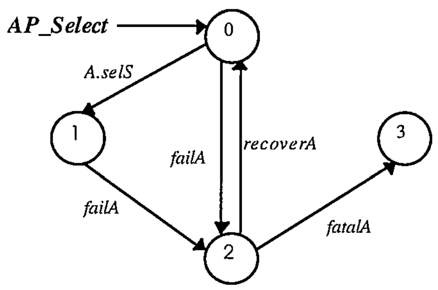

Figure 7: A Global LTS of RMTP

The analysis is performed in two steps. Firstly the RMTP protocol is decomposed into a hierarchy of subsystems that mirrors its multicast tree. Secondly the LTS of the overall system is composed step by step from those of its subsystems in a bottom-up manner. In each intermediate step, the LTS of a subsystem is simplified by hiding internal actions that are not of interest to the global view of the protocol.

The key to the success of CRA techniques is to employ modular software architecture and hide as many internal actions as possible in each subsystem. The observable behaviour of a subsystem where internal actions have been hidden can in general be represented by a simpler LTS. However, the properties that are available for reasoning in the analysis are then constrained by the set of remaining globally observable actions. For instance, consider the case of liveness property Liv_Rec3 in Figure 6. This property involves actions that are internal to subsystem REC_1. Actions *selA, selS,* and *ms_fail* would therefore need to be exposed in the global graph of the system for verifying property Liv_Rec3. However, this compromises the CRA approach. One of the main advantages of CRA is that it offers to the users the possibility of abstracting from the behaviour of subsystems those details in which they are not interested. This should obviously not be made at the expense of the effectiveness of analysis.

[1] This LTS has been constructed automatically by using the TRACTA tool [3].

In our previous work [10] we describe a technique for making the verification of safety properties independent from the actions that are observable at the global state graph of the system. In this paper, we provide a way of achieving the same goal for the case of liveness properties. Our method achieves this without reducing the advantages that CRA exhibits as compared to traditional reachability analysis. Our experimental results presented in Section 6 demonstrate and confirm this.

5 Validation of Liveness Properties

5.1 Specification of Properties

We have incorporated in CRA a mechanism for checking liveness properties. The method exhibits three main desirable features. *Firstly*, it finds a way of making the hiding of actions independent of the liveness properties that are to be checked in the final graph. *Secondly*, it checks simultaneously multiple properties, specifically identifies the violated ones and generates the overall system behaviour. *Thirdly*, it avoids keeping specific information on states. Instead states are differentiated in terms of the actions that can be performed at them.

To achieve the above features, we have introduced a mapping between a given property automaton and its associated liveness property LTS, as in Definition A.

Definition A: A property automaton $P = < S, A, \Delta, q, F>$ is mapped into a liveness property LTS $P' = < S, A \cup \{acc\}, \Delta', q>$ by adding a new globally unique action *acc* and new transitions such that:

(i) $acc \notin \alpha A$; and

(ii) $\Delta' = \Delta \cup \{s \xrightarrow{acc} s \mid s \in F\}$.

Applying the Definition A to the property automata in Figure 6 results in the LTS depicted in Figure 8.

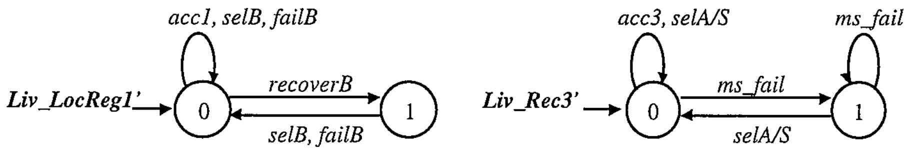

Figure 8: Liveness Properties in ECRA

The mapping identifies the acceptance state of Liv_LocReg1 with its ability to perform the action *acc1* in Liv_LocReg1'. Transitions, which identify acceptance states, are referred to as acceptance transitions. The use of acceptance transitions removes the need for modelling acceptance states in the LTS model. Storing acceptance states as special states in the analysis process would have required the introduction of specific rules for the minimisation procedure. With the use of acceptance transitions, any two states *s* and *s'* of a subsystem *Sys* are considered behaviourally equivalent, if and only if *s* and *s'* (or the respective states to which they can unobservably transit) represent the same acceptance status for liveness properties that have been introduced in the subtree rooted at *Sys*. Thus, in our checking mechanism described in the following section, an LTS violates a liveness property iff its minimised equivalent does.

5.2 Checking Properties and Locating Violations

In our Extended Compositional Reachability Analysis (ECRA) technique, every property automaton *B* is mapped to an LTS *B'* as described in Definition A. Each *B'* is included in the compositional hierarchy for composition with the (sub)system for which it expresses some liveness property. CRA is then used to compute the global graph for the system. In our case study for example, the liveness property LTS Liv_Rec3' has been included in the subtree of REC_3 (see Figure 10). REC_3 thus becomes:

(Channel3 || Watch3 || Receiver3 || Liv_Rec3') $\uparrow$ {*failA, advA/S, acc3*}.

In ECRA, a process *P* satisfies the liveness property expressed as a property automaton *B* if and only if all cycles in *P*||*B'* contain a transition labelled by the acceptance action of *B'*.[2] For simplicity, the technique assumes fair selection and fair process execution in the modelled systems[3]. For example in a communicating channel, the assumption ignores unfair execution sequences that keep losing messages. It also ignores those situations where a ready process never fires. Under the stated assumption, satisfaction of property *B* can be reduced to checking the existence of acceptance transitions at *terminal* sets of states of *P*||*B'*. A set of states *C* in an LTS $< S, A, \Delta, p >$ is said to be *terminal* if and only if:

- *C* is a strongly connected component; and
- *C* is closed under Δ, i.e., $\forall\ s \in C,\ (s, a, s') \in \Delta \Rightarrow s' \in C$.

The computation of terminal sets of states in a graph can be performed with complexity linear to the size of a graph [24]. ECRA also keeps track of all acceptance actions that have been introduced in the analysis. Let *a* be an acceptance action introduced to identify the acceptance states of a property automaton *B*. Our method concludes that the property *B* is not satisfied by a system *S* if *S*||*B'* contains terminal sets of states where *a* cannot be executed. Since the action *a* uniquely identifies a property automaton, ECRA specifically indicates which properties cannot be satisfied by the system under analysis.

When analysis uncovers property violations, a useful kind of diagnostic information is to provide the user with a detailed path leading to the violation. In a compositional technique where actions have been hidden at intermediate phases of

---

[2] This is a mechanism adapted from that described by Gribomont et al [12] and Fernandez et al [13].

[3] Our analysis technique additionally uses a liveness checking mechanism that applies to cases where the fairness assumption is too restrictive. Assuming fairness is thus provided as an option in our analysis tools. It is beyond the scope of this paper to describe the latter mechanism.

analysis, abstracted information can be recovered by using hierarchical tracing [25]. When no violation is detected, ECRA removes acceptance transitions from the global state-graph and then minimises it. The minimisation results in an LTS observationally equivalent to the one that would have been obtained if the liveness properties had not been included in the analysis.

5.3 Checking the RMTP Protocol

The RMTP protocol as described in section 3 has been used for comparing our method with both traditional Reachability Analysis (RA) and CRA. In the case study, we have assumed that the user wishes to globally expose actions *failA, A.selS, recoverA* and *fatalA,* and therefore observe only part of the behaviour of component DR_A within the system. All remaining actions have been hidden as soon as they were made internal to subsystems.

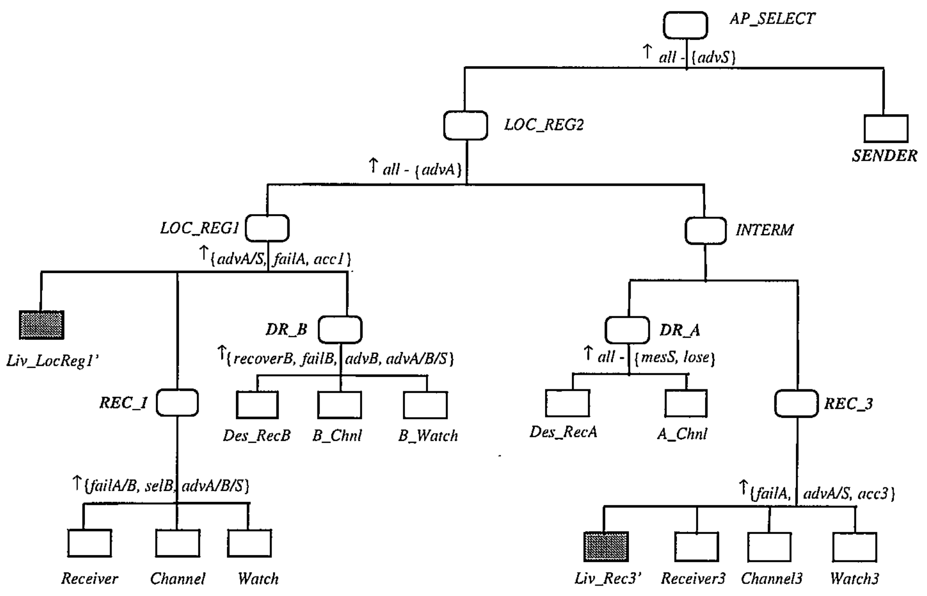

Figure 10: The compositional hierarchy for the RMTP

The ECRA technique has been applied to the compositional hierarchy that mirrors the RMTP multicast tree of Definition A, where LTS Liv_LocRegl' and Liv_Rec3' have been introduced as described in section 5.2 (see Figure 10). The global graph thus constructed for AP_SELECT contains 344 states and 2,626 transitions. However, the graph contains terminal sets of states where *acc1* cannot be executed. As such, ECRA has identified that the system modelled by our specifications violates Liv_LocRegl, and has returned a trace in the global graph, that leads to a terminal set of states in which *acc1* cannot be performed.

This trace together with the intermediate subsystems obtained have been used for building up the debugging trace «*selB, failB, advA, τ, τ, failA, recoverB, advB, τ*» on the graph of subsystem LOC_REG1 before minimisation. This trace leads LOC_REG1 to a non-acceptance state that forms a terminal singleton of states where the only actions that can be performed are *advA/B/S*. Mapping this trace to REC_1 we obtained trace «*selB, failB, ms_fail, advA, mesA, selA, failA, advB, mesB*» which drives components (Receiver, Channel, Watch) to state (0, 0, 3). In this state of REC_1, both the Receiver and the Watch processes are deadlocked, and REC_1 can only perform non-progress cycles where the Channel keeps receiving and losing advertisements.. At this stage, it has been relatively easy to track down the problem to an omission in the specification of process Watch. When Process Watch is in state 3, it is ready to inform the Receiver about the failure of its AP, but is not ready to record a new AP selection by the Receiver. However, the trace obtained illustrates that the Receiver may be at the stage of selecting DRB due to proximity in the multicast tree, in which case it is no longer interested in failures of its current AP.

After addition of the transition (3, *selB,* 1) to process Watch, we have performed ECRA to the corrected version of the RMTP, obtaining the global graph shown in Figure 11. Having used a compositional approach, we have re-computed only those subsystems affected by the change in the specifications. No violation of liveness properties was detected this time. Acceptance transitions may therefore be removed from the global graph, resulting in an LTS that reflects the behaviour of component DR_A in the multicast tree. This LTS may be used to check further behavioural properties, such as the one which asserts that whenever DR_A recovers from failure, it can always select an AP.

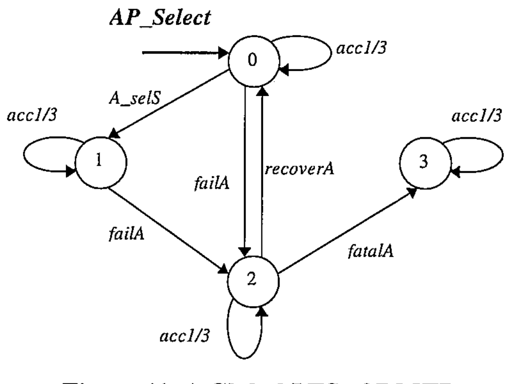

Figure 11: A Global LTS of RMTP

6 Experimental Results

We have performed the RMTP case study using ECRA, CRA and RA for both the cases of incorrect and correct specifications. We have then compared the three techniques in terms of the size of the graphs that they have generated. The experiments were conducted using TRACTA - an environment for analysing behaviour of distributed systems [3]. The results are summarised in Tables 1 and 2.

Incorrect Specification	**ECRA**		**CRA**		**Traditional RA**	
	#states	#trans.	#states	#trans.	#states	#trans.
Largest subsystem	90	370	91	305	not applicable	
Global system	344	2,626	1,291	7,586	96,528	664,416

Table 1: Results for Incorrect Specifications of RMTP

Specification After Correction	**ECRA**		**CRA**		**Traditional RA**	
	#states	#trans.	#states	#trans.	#states	#trans.
Largest subsystem	90	370	91	305	not applicable	
Global system	4	13	1,371	8,035	96,528	672,588

Table 2: Results for Specifications of RMTP After Correction

The size of the graph generated by traditional RA shows that even the part of the RMTP protocol presented is nontrivial to analyse. The results were obtained by excluding from the analysis components REC_2 and REC_4 which exhibit behaviour identical to that of components REC_1 and REC_3, respectively. In the experiments, CRA was found to be more efficient than traditional RA, even with the global exposition of actions involved in the liveness properties of interest. The largest graph generated by CRA is smaller than that by traditional RA by 70 times. This justifies the use of CRA for this verification. However, the advantages that CRA exhibits as compared with traditional RA gradually disappear as the number of actions that need to be globally observable increases.

ECRA, on the other hand, performs better in both cases. In the case where specification is correct, it reduces the global graph generated by CRA by 300 times, and the one generated by RA by 24,000 times. Moreover, it returns a graph that exposes concisely the system behaviour of interest to the developer. We have to mention here that, in ECRA, although the largest intermediate subsystem in the correct case has the same size as the one in the incorrect case, the size for most of the intermediate subsystems was reduced in the former case.

We have made the following observation in our experiments with ECRA and CRA techniques. Consider subtrees of the compositional hierarchy containing liveness LTSs that involve actions in some set *Actions*. In most cases, for all subsystems in which all members of *Actions* have been exposed by CRA, ECRA performs better or, in the worst case, equally to CRA. An informal explanation for this is that ECRA is a technique that achieves *selective* minimisation when liveness properties are included in the analysis. It inhibits, in the minimisation process, the merging of states that could result in hiding violations in the global graph for the system. It is therefore not expected to increase the size of a graph where observable actions can be used to detect the violation. In the absence of violations, it allows minimisation to proceed to its full effects, as has been shown in Table 2.

7 Conclusion and Future Work

In this paper, we have extended compositional reachability analysis with a mechanism for verifying multiple liveness properties that may involve globally unobservable actions. The mechanism does not require modification of the well-known formalism of LTS, and can be readily integrated in the existing framework of CRA. The integration preserves the existing composition and minimisation procedures of CRA. This has been achieved by avoiding special treatment of acceptance states. Acceptance states are identified with transitions labelled by actions globally unique to the system under analysis. A further advantage is that the approach is complementary to our approach for checking safety properties [10]. Safety properties are specified as property automata which can be composed directly with those specifying liveness properties under the CRA framework. As mentioned, to the best of our knowledge, no similar work provides the possibility of simultaneously checking multiple properties in the framework of CRA. This is particularly so in the presence of action hiding. Solutions that have been proposed compromise one or more desirable features of CRA.

Note that we have chosen not to use automata for the negation of properties, a method proposed in other work [12, 14, 15]. We have found it counter-intuitive and often difficult to express the automaton describing the negation of a given property. Moreover, for a number of properties that we have examined, the automaton for their negation has often contained more states than the one for the positive property. It is always possible to algorithmically construct a Büchi automaton for any formula in linear temporal logic, but the algorithm is not optimal. Thus it could generate much larger automata than the ones generated manually [12].

A case study of a reliable multicast transport protocol of over 96,000 states and 660,000 transitions has been used to illustrate our technique and compare it to alternative ways of verifying liveness properties. Promising results have been obtained. The mechanism preserves key desirable features of CRA while enhancing its verification capabilities.

Further work is needed to provide guidance as to which actions to hide and at which point in the compositional hierarchy to compose the properties to be checked. This is both a logical decision as to which is the most sensible, and an efficiency decision as to which aids the minimisation automation. We are currently working on further optimisations to the proposed mechanism and towards extending it with contextual constraints [26]. We are also investigating a method that. after recording the violation on subsystem states, prunes from those subsystems the transitions originating from those states.

References

[1] G. R. Andrews, *Concurrent Programming - Principles and Practice*: The Benjamin / Cummings Publishing Company Ltd., 91.

[2] B. Alpern and F. B. Schneider, "Verifying Temporal Properties without Temporal Logic," *ACM Transactions on Programming Languages and Systems*, vol. 11, pp. 147-167, 89.

[3] D. Giannakopoulou, J. Kramer, and S. C. Cheung, "TRACTA: An Environment for Analysing the Behaviour of Distributed Systems," presented at ACM SIGPLAN Workshop on Automated Analysis of Software, Paris, 97.

[4] J. Malhotra, S. A. Smolka, A. Giacalone, and R. Shapiro, "A Tool for Hierarchical Design and Simulation of Concurrent Systems," presented at BCS-FACS Workshop on Specification and Verification of Concurrent Systems, Stirling, Scotland, 88.

[5] W. J. Yeh and M. Young, "Compositional Reachability Analysis Using Process Algebra," presented at Symposium on Testing, Analysis, and Verification (TAV4), Victoria, British Columbia, 91.

[6] K. K. Sabnani, A. M. Lapone, and M. Ü. Uyar, "An Algorithmic Procedure for Checking Safety Properties of Protocols," *IEEE Transactions on Communications*, vol. 37, pp. 940-948, 89.

[7] K. C. Tai and P. V. Koppol, "Hierarchy-Based Incremental Reachability Analysis of Communication Protocols," presented at IEEE International Conference on Network Protocols, San Francisco, California, 93.

[8] W. J. Yeh, "Controlling State Explosion in Reachability Analysis," : SERC, Purdue University, 93.

[9] G. S. Avrunin, U. A. Buy, J. C. Corbett, L. K. Dillon, and J. C. Wileden, "Automated Analysis of Concurrent Systems with the Constrained Expression Toolset," *IEEE Transactions on Software Engineering*, vol. 17, pp. 1204-1222, 91.

[10] S. C. Cheung and J. Kramer, "Checking Subsystem Safety Properties in Compositional Reachability Analysis," presented at 18th International Conference on Software Engineering, Berlin, Germany, 96.

[11] R. Milner, *Communication and Concurrency*: Prentice-Hall, 89.

[12] P. Gribomont and P. Wolper, "Temporal Logic," in *From Modal Logic to Deductive Databases*, A. Thayse, Ed.: John Wiley and Sons, 89.

[13] H.-C. Fernandez, L. Mounier, C. Jard, and T. Jéron, "On-the-fly Verification of Finite Transition Systems," in *Computer-Aided Verification*, R. Kurshan, Ed.: Kluwer Academic Publishers, 93.

[14] P. Godefroid and G. J. Holzmann, "On the Verification of Temporal Properties," presented at 13th IFIP WG 6.1 International Symposium, on Protocol Specification, Testing, and Verification, 93.

[15] S. Aggarwal, C. Courcoubetis, and P. Wolper, "Adding Liveness Properties to Coupled Finite-State Machines," *ACM Transactions on Programming Languages and Systems*, vol. 12, 90.

[16] T. Bultan, J. Fischer, and R. Gerber, "Compositional Verification by Model Checking for Counter-Examples," presented at International Symposium on Software Testing and Analysis, San Diego, California, 96.

[17] J. Kemppainen, M. Levanto, A. Valmari, and M. Clegg, ""ARA" Puts Advanced Reachability Analysis Techniques Together," presented at 5th Nordic Workshop on Programming Environment Research, Tampere, Finland, 92.

[18] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering, Chapter 6*: Prentice-Hall, Inc., 91.

[19] A. Rabinovich, "Checking Equivalences Between Concurrent Systems of Finite Agents," presented at 19th International Colloquium on Automata, Languages and Programming, Wien, Austria, 92.

[20] A. Valmari, "Alleviating State Explosion during Verification of Behavioural Equivalence," : Department of Computer Science, University of Helsinki, Finland, 92.

[21] E. M. Clarke, D. E. Long, and K. L. McMillan, "Compositional Model Checking," presented at 4th Annual Symposium on Logic in Computer Science, Pacific Grove, California, 89.

[22] C. A. R. Hoare, *Communicating Sequential Processes*: Prentice-Hall, 85.

[23] J. C. Lin and S. Paul, "RMTP: A Reliable Multicast Transport Protocol," presented at IEEE INFOCOMM'96, San Francisco, California, 96.

[24] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*: Addison-Wesley, 83.

[25] W. J. Yeh and M. Young, "Hierarchical Tracing of Concurrent Programs," presented at 3rd Irvine Software Symposium (ISS'93), Irvine, California, 93.

[26] S. C. Cheung and J. Kramer, "Context Constraints for Compositional Reachability Analysis," *ACM Transactions on Software Engineering and Methodology*, 96.