Incorporating Verification of Liveness Properties in Compositional Reachability Analysis

Shing-Chi Cheung†, Dimitra Giannakopoulou†, and Jeff Kramer†

Technical Report HKUST-CS96-36
October 1996

* HKUST
Department of Computer Science
University of Science & Technology
Clear Water Bay, Hong Kong
scc@cs.ust.hk

† Imperial College of Science, Technology & Medicine
Department of Computing
London SW7 2BZ, UK
{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 automated technique which can be used in stages to derive the overall behaviour of a distributed program based on its architecture. CRA is particularly suitable for the analysis of programs which are subject to evolutionary change. When a program evolves, only behaviours of those subsystems affected by the change need be re-evaluated. The technique however has a limitation. The properties available for analysis are constrained by the set of actions that remain globally observable. The liveness properties of subsystems which involve hidden actions may therefore not be analysed. In this paper, we extend compositional reachability analysis to check liveness properties of subsystems which may involve actions that are not globally observable. 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.
Incorporating Verification of Liveness Properties in Compositional Reachability Analysis

Shing Chi Cheung
Department of Computer Science
Hong Kong University of Science and Technology,
Clear Water Bay, Hong Kong
Fax: (+852) 2358 1477
Email: scc@cs.ust.hk

Dimitra Giannakopoulou, Jeff Kramer
Department of Computing
Imperial College of Science, Technology and Medicine, London SW7 2BZ, UK
Fax: (+44-171) 581 8024
Email: {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 automated technique which can be used in stages to derive the overall behaviour of a distributed program based on its architecture. CRA is particularly suitable for the analysis of programs which are subject to evolutionary change. When a program evolves, only behaviours of those subsystems affected by the change need be re-evaluated. The technique however has a limitation. The properties available for analysis are constrained by the set of actions that remain globally observable. The liveness properties of subsystems which involve hidden actions may therefore not be analysed. In this paper, we extend compositional reachability analysis to check liveness properties of subsystems which may involve actions that are not globally observable. 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
Static analysis, distributed systems, labelled transition systems, Büchi automata, liveness properties, compositional reachability analysis.

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 [3]. 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 [3]. 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. These machines will be referred to as property automata. Each property automaton specifies the set of feasible execution sequences over the 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.

Figure 1: A Property Automaton

However, distributed programs are generally very complex to analyse. Even for small programs, analysis of their behaviour is impractical without the support of an effective automated technique. One approach is to exploit the design structure (software architecture) of the distributed program. 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; these processes can be either primitive or composite.

A common approach for the analysis of distributed programs is to construct a semantically equivalent representation of the global system. However, the search
space involved generally increases dramatically with the number of parallel processes. Great effort has been made to avoid this state explosion problem by not having to construct the complete state graph. Roughly, the proposed methods can be classified into two categories: reduction by partial ordering and reduction by compositional minimisation.

In the former category, reduction is achieved by avoiding the generation of all paths formed by the interleaving of the same set of transitions [12, 14, 24]. In the latter category, reduction is achieved by intermediate simplification of subsystems [17, 21, 22, 27]. Techniques in this category are known as compositional reachability analysis (CRA).

They were originally proposed to remedy the problem of traditional reachability analysis techniques [19, 23] which compose the global system representation in a single step. Promising results have been reported. Yeh [26] described several case studies which suggested similar performance between a technique of compositional reachability analysis and that of constraint expressions [4]. Sabnani et al. [21] 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 observation has also been made by Tai and Koppol [22]. Furthermore, CRA techniques are particularly suitable to analyse programs which are likely to evolve. The techniques help localise the effect of change. When changes are applied to a program, only the state machines of those subsystems that are affected by the changes need be re-computed.

Although CRA techniques have advantages over traditional techniques of reachability analysis, the system representation generated cannot be utilised to validate behavioural properties involving actions that are not globally observable. Verification has to be restricted to those properties formed by the globally observable actions. This can severely limit the applicability of CRA techniques to complex programs.

Previous work [6] has described a technique for making the verification of subsystem safety properties independent from the actions that are observable at the global state graph of the system. In this paper, we incorporate into the CRA techniques a mechanism to validate liveness properties of subsystems without making the involved actions globally observable. These properties are violated when some subsystems, within the context of a distributed application, can perform execution sequences not acceptable to the specified property (Büchi) automata. The validation can be carried out in the enhanced mechanism of CRA. If no violation of liveness properties is detected, the analysis constructs a global LTS observationally equivalent [18] to that constructed using conventional CRA techniques; otherwise it indicates which and how liveness properties are violated.

The problem of handling liveness properties has attracted a lot of attention in the literature. However, 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.

As mentioned, we have adopted the approach which expresses properties as Büchi automata. The system satisfies a property if all sequences that the system generates are acceptable by the specified Büchi automaton. Büchi automata can be used to express formulae of linear time temporal logic [13]. Fernandez et al. propose a technique to compose the property automata with the system [9]. Godefroid and Holzmann in [11] 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. [1]. 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 $\forall$CTL [5]. 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 has not been addressed. Moreover, the final graph obtained reflects only the system behaviour generating counterexamples in the case of violations. The final graph is thus empty when no violation is detected.

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 comparison 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 [8, 10, 15, 20, 25]. 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 \( < S, A, \Delta, q > \) where

(i) \( S \) is a set of states;

(ii) \( A = A' \cup \{ \tau \} \), where \( A' \) is the communicating alphabet of \( P \) which does not contain the internal action \( \tau \);

(iii) \( \Delta \subseteq S \times A \times S \), denotes a transition relation that maps from a state and an action onto another state;

(iv) \( q \) is a state in \( S \) which indicates the initial state of \( P \).

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

\[
< S, A, \Delta, q > \xrightarrow{a} < S, A, \Delta, q' > \text{ iff } (q, a, q') \in \Delta.
\]

Since there is a one-to-one mapping between a process \( P \) and its LTS, we use the terms process and LTS interchangeably.

Observability of actions in a process can be controlled by a restriction operator \( \uparrow \). \( P \uparrow \Delta \) represents the process projected from \( P \) in which actions in \( A \setminus \Delta \) are replaced by the internal action \( \tau \). Concurrent LTSs are synchronised via transitions labelled with the same communicating actions. A set of states \( C \) in an LTS \( < S, A, \Delta, q > \) is said to be terminal if and only if:

a) \( C \) is a strongly connected component; and

b) \( C \) is closed under \( \Delta \), i.e.,

\[
\forall s \in C, (s, a, s') \in \Delta \Rightarrow s' \in C.
\]

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, \( \Delta \) is a set of transitions, \( q_0 \) is its initial state, and \( F \) is a set of acceptance states. An infinite word \( a_0 a_1 a_2 ... \) over \( A \) is accepted by \( B \) if and only if there exists an infinite execution \( q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow ... \) of \( B \) such that \( q_i \in F \) for infinitely many \( i \)’s.

3 MODELLING THE RELIABLE MULTICAST TRANSPORT PROTOCOL (RMTP)

To illustrate our approach, we present a Reliable Multicast Transport Protocol (RMTP) as proposed by Lin and Paul [16]. 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 status 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. Receivers in each local region send their ACKs to the corresponding DR, which does not contain 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.

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 \( \text{adv} \)) by multicasting SAT packets along their subtrees. All SAT packets have identical initial time-to-live (TTL) values in the header. Routers decrement the TTL value when forwarding packets. Therefore a larger TTL is equivalent to a closer proximity in the multicast tree. A receiver stores the address and TTL value of its selected AP, and selects a new one as soon as a SAT packet with a higher TTL is received. To recover from the failure of the selected AP, a time-out is also used to initiate the selection mechanism.

In our case study, we have modelled this part of the protocol for the configuration depicted in Figure 2. Three processes are associated with every simple receiver in the multicast tree, namely the Receiver, Channel, and Watch processes. For example, the behaviour of REC_1 in Figure 2, is given by the composite behaviour of the Receiver, Channel and Watch processes, specified as in Figure 3.
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 $ms\_failA/B$) 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 are made unobservable, i.e.,

$$REC_1 = (Channel \parallel Watch \parallel Receiver) \uparrow$$
In Figure 4 we illustrate the behaviour of designated receiver B (DR_B in the multicast tree). 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 (fail_B), and enter a state where it stops advertising itself. From this state it may either fatally fail (fatal_B) or recover (recover_B). All actions in DR_B that do not synchronise with its environment are made unobservable, i.e.,

$$\text{DR}_B = (B\_Chnl || B\_Watch || \text{Des}_\text{RecB}) \uparrow \{\text{failA/B, advA/B/S}\}.$$  

Note that we have not modelled failure for simple receivers and the sender. If the sender fails, the multicast session is cancelled, in which case RMTP does not need to fulfil its objectives. Properties on the receivers are not expected to hold when they fail. Moreover, failures of simple 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 5). Property LivRec_1 refers to REC_1, and asserts that whenever DR_B, its nearest AP, recovers from failure (recover_B) and does not fail again (fail_B), REC_1 will eventually select B as its AP (selB).

Property LivRec_3 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 9- 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.

### 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 [21, 22, 27]. In compositional reachability analysis (CRA) techniques, the model of the target system is given as an LTS which describes an abstraction of the system behaviour, according to the requirements of the user. In our case study, the abstracted behaviour of designated receiver DR_A is as shown in Figure 6 (acceptance transitions are ignored). 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.

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.

![Figure 6: A Global LTS of RMTP](image)

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.
Applying the Definition A to the property automata in
with its ability to perform the action
The mapping identifies the acceptance state of LivRec_1
acc
A
is mapped into a liveness property LTS
removes the need for modelling acceptance states in the
as acceptance transitions. The use of acceptance transitions
Transitions which identify acceptance states are referred to
the analysis process, would have required the introduction
LTS model. Storing acceptance states as special states in
of specific rules for the minimisation procedure. With the
use of acceptance transitions, any two states
of a

Figure 7: Liveness Properties in ECRA

In our previous work [6] we describe a technique for
making the verification of subsystem 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 special
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 the
following mapping between a given property automaton
and the associated liveness property LTS:

Definition A: A property automaton \( P = \langle S, A, \Delta, q, F \rangle \)
is mapped into a liveness property LTS \( P' = \langle S, \Delta', q \rangle \)
by adding a new globally unique action \( \text{acc} \) and new transitions such that:

(i) \( \text{acc} \notin \Delta \); and

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

Applying the Definition A to the property automata in
Figure 5 results in the LTS depicted in Figure 7.

The mapping identifies the acceptance state of LivRec_1
with its ability to perform the action \( \text{acc} \) in LivRec_1'.
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

Figure 8: A Terminal Set of States

subsystem \( S_{sys} \) 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 \( S_{sys} \). 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 subsystem for which it expresses some liveness
property. CRA is then used to compute the global graph
for the system. For example, it includes the liveness
property LivRec_1 in the subsystem REC_1 in Definition
A. After the inclusion, REC_1 becomes:

\[
(\text{Channel} \parallel \text{Watch} \parallel \text{Receiver} \parallel \text{LiveRec}_1) \uparrow \{\text{failA/B, advA/B/S, recoverB, acc1}\}.
\]

In ECRA, a process \( P \) satisfies the liveness property
expressed by a property automaton \( B \) if and only if all
cycles in \( P \downarrow B' \) contain a transition labelled by the
acceptance action of \( B' \). \(^1\) For simplicity, the technique
assumes fair selection and fair process execution in the
modelled systems. \(^2\) 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 has never been fired. Under the
stated assumption, satisfaction of property \( B \) can be
reduced to checking the existence of acceptance states at
terminal sets of states. This can be computed in a
complexity linear to the size of a graph [2].

ECRA keeps track of all acceptance actions that have been
introduced in the analysis. Suppose \( a \) is an acceptance
action introduced to identify the acceptance states of a
property automaton \( B \). We conclude that the property \( B \)

\(^1\) This is a mechanism adapted from that described in [9] and [13].

\(^2\) The method proposed under the fairness assumption can be easily
adapted to handle cases where the assumption is not made. However,
this is beyond the scope of this paper.
cannot be satisfied by a system if its corresponding LTS 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.

Let us consider a terminal set of states in subsystem REC_1 as shown in Figure 8a. According to the compositional hierarchy (Figure 9), actions \( \text{advA/B/S} \) are to be hidden from the global state-graph. The underlying minimisation procedure of ECRA eventually reduces the set of states to a singleton set of states in the global state-graph as shown in Figure 8b. This singleton set contains no state that may execute a transition labelled by \( \text{acc1} \) which is the acceptance action of LivRec_1'. As a result, it represents a situation where the liveness property expressed by LivRec_1 is violated.

Counter examples can be provided by returning a path leading to the problematic terminal set of states where the violation occurs. This is achieved by recovering abstracted information at intermediate graphs of subsystems [28]. 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 had liveness properties not been included in the analysis.

5.3 Checking the RMTP Protocol

The RMTP protocol as described in section 3 was 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 in the system. All remaining actions have been hidden as soon as they were made internal to subsystems.

The ECRA technique was applied to the compositional hierarchy that mirrors the RMTP multicast tree of Definition A, with LTS LivRec_1' and LivRec_3' being introduced as described in section 5.2 (see Figure 9). ECRA identified that the system modelled by our specifications violates LivRec_1', and returned a trace in the global graph, that leads to a terminal set of states in which \( \text{acc1} \) cannot be performed.

Since LivRec_1 is a property that refers to REC_1, we have used the state-graphs obtained for intermediate subsystems on the left subtree of the multicast tree to build-up trace (\( \text{selB, failB, ms_fails, advA, mesA, selA, failA, recoverB, advB, mesB} \)) for process REC_1. This trace leads to a terminal set of states where advertisements from APs above REC_1 in the multicast tree are received and lost by the Channel (Figure 8a). The trace drives components (Receiver, Channel, Watch, LivRec_1') to state \((0, 0, 2, 1)\) from which neither \( \text{selB, failB} \) can occur due to synchronisation problems. At this stage, it was relatively easy to detect a problem in our specifications for process Watch of REC_1. Process Watch was modelled to record the selected AP for the Receiver, and observe and transmit failures of this AP to the Receiver. However, according to Figure 3, when process Watch enters state 2, it cannot record the selection of DR_B by the Receiver. This is obviously due to an omitted transition in the specification of Watch.

After addition of the transition \((2, \text{selB}, 3)\) to process Watch, we repeat ECRA to the corrected version of the RMTP, obtaining the global graph shown in Figure 6. Having used a compositional approach, we need only re-compute 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

![Figure 9: The compositional hierarchy for the RMTP](image-url)
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.

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 results are summarised in Tables 1 and 2.

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 components 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. Then 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 Figure 6 and Table 2.

7 CONCLUSION AND FUTURE WORK
In this paper, we have extended compositional reachability analysis with a mechanism for verifying multiple liveness properties of subsystems 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 giving special treatment to 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 [6]. Safety properties are specified as property automata which can be composed directly with those specifying liveness properties under the CRA framework.

Note that we have chosen not to use automata for the negation of properties, a method proposed in other works [1, 11, 13]. 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

<table>
<thead>
<tr>
<th>Specification</th>
<th>ECRA</th>
<th>CRA</th>
<th>Traditional RA</th>
</tr>
</thead>
<tbody>
<tr>
<td>#states</td>
<td>#trans.</td>
<td>#states</td>
<td>#trans.</td>
</tr>
<tr>
<td>Largest subsystem</td>
<td>90</td>
<td>370</td>
<td>91</td>
</tr>
<tr>
<td>Global system</td>
<td>344</td>
<td>2,626</td>
<td>1,291</td>
</tr>
</tbody>
</table>

Table 1: Results for Incorrect Specifications of RMTP

<table>
<thead>
<tr>
<th>Specification After Correction</th>
<th>ECRA</th>
<th>CRA</th>
<th>Traditional RA</th>
</tr>
</thead>
<tbody>
<tr>
<td>#states</td>
<td>#trans.</td>
<td>#states</td>
<td>#trans.</td>
</tr>
<tr>
<td>Largest subsystem</td>
<td>90</td>
<td>370</td>
<td>91</td>
</tr>
<tr>
<td>Global system</td>
<td>4</td>
<td>13</td>
<td>1,371</td>
</tr>
</tbody>
</table>

Table 2: Results for Specifications of RMTP After Correction
generated manually [13]. However we accept that using
the negation approach may be more efficient in the case
where the model for processes themselves supports
acceptance states. It is easier to check that the result of the
composition accepts the empty set, than to reflect our
checking mechanism on the acceptable executions of the
model for the system.

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.

We are currently working on further optimisations to the
proposed mechanism and extend it with contextual
constraints [7]. We are also investigating a method that
prunes from subsystems those transitions originating from
states that are potential sources of violations.

REFERENCES
[1] S. Aggarwal, C. Courcoubetis, and P. Wolper,
“Adding Liveness Properties to Coupled Finite-State
Machines,” ACM Transactions on Programming
Principles and Practice: The Benjamin / Cummings
Dillon, and J. C. Wileden, “Automated Analysis of
Concurrent Systems with the Constrained Expression
Toolset,” IEEE Transactions on Software
“Compositional Verification by Model Checking for
Counter-Examples,” presented at International
Symposium on Software Testing and Analysis, San
Diego, California, January 1996.
Safety Properties in Compositional Reachability
Analysis,” presented at 18th International Conference
on Software Engineering, Berlin, Germany, March
1996.
for Compositional Reachability Analysis,” ACM
Transactions on Software Engineering and
Methodology, (to appear).
“Compositional Model Checking,” presented at 4th
Annual Symposium on Logic in Computer Science,
Pacific Grove, California, June 1989.
“On-the-fly Verification of Finite Transition
Systems,” in Computer-Aided Verification, R.
[10] C. Ghezzi, M. Jazayeri, and D. Mandrioli,
Fundamentals of Software Engineering, Chapter 6:
Verification of Temporal Properties,” presented at
13th IFIP WG 6.1 International Symposium, on
for the Efficient Verification of Deadlock Freedom and
Safety Properties,” presented at 3rd International
Conference on Computer Aided Verification (CAV),
Aalborg, Denmark, July 1991.
From Modal Logic to Deductive Databases, Chapter
[14] G. Holzmann, P. Godefroid, and D. Pirotti,
“Coverage Preserving Reduction Strategies for
Reachability Analysis,” presented at 12th
International Symposium on Protocol Specification,
Testing, and Verification, Lake Buena Vista, Florida,
Clegg, “ARA” Puts Advanced Reachability
Analysis Techniques Together,” presented at 5th
Nordic Workshop on Programming Environment
Multicast Transport Protocol,” presented at IEEE
INFOCOM’96, San Francisco, California, March
1996.
Shapiro, “A Tool for Hierarchical Design and
Simulation of Concurrent Systems,” presented at
BCS-FACS Workshop on Specification and
Verification of Concurrent Systems, Stirling,
[18] R. Milner, Communication and Concurrency:
Concurrent Systems of Finite Agents,” presented at
19th International Colloquium on Automata,
Languages and Programming, Wien, Austria, July


