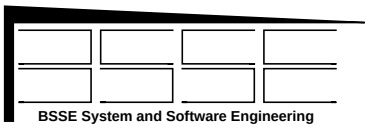

Some Hints about How to Reduce Size of State Space

R. Gerlich

BSSE System and Software Engineering, Auf dem Ruhbuehl 181, D-88090 Immenstaad, Germany
Phone: +49/7545/91.12.58, Fax: +49/7545/91.12.40, e-mail: gerlich@t-online.de

Keywords: SDL, state space reduction, state exploration, exhaustive simulation, state explosion

Abstract:

During set up of the generic modelling environment EaSySim II it was recognised (as already described in [1]) that an unreasonable high number of system states was reported for a rather simple system (as described in [2] and Fig. 5-1 below) which prevented termination of state exploration. This was the starting point for a number of tests which discovered the drivers for the huge state space. In consequence, a reduction from more than one million states down to about 200 states could be achieved by taking only a subset of SDL and applying an appropriate modelling style. The underlying idea is that it should be better to impose certain implementation rules to succeed with state exploration than to compromise the overall success by allowing an arbitrary modelling style. Most of such rules could be supported by tools or by additional SDL directives.

1. Introduction

It is well known that SDL provides powerful means for verification of a system's behaviour based on its formal representation by Finite State Machines (FSM) and the formal description of data flow by Message Sequence Charts (MSC). This allows to perform systematic exploration of the state space by exhaustive simulation.

However, in practice it is very difficult to succeed with exhaustive simulation for real systems, even for rather simple examples like the one given by [2] and Fig. 5-1 respectively. To succeed techniques like filtering, static exploration of state space or limitation of the number of instances are applied.

Filtering may compromise the representativity of a verification result because some negative side effects may be excluded from the actually investigated state space and hence may not be identified: by filtering only a certain property of a system can be confirmed, but the proof for absence of faults or bugs cannot be given.

In case of static exploration of state space] without consideration of data and without execution of code the result will only represent a logical view, but will not give a feedback from an executing system.

(C) Copyright Dr. Rainer Gerlich BSSE System and Software Engineering 1999. All Rights Reserved.

Taking a few instances only instead of the maximum number during exhaustive simulation is reasonable, if the system's behaviour is really independent of the number of instances. However, this is not true if consumption of resources is taken into account [1].

In consequence, no satisfying solution exists which allows to investigate the full state space of a system in a representative manner when aiming to demonstrate absence of bugs or taking into account shared resources.

When looking on the amount of states generated during exhaustive simulation for a rather simple system, the impression was that such a high number of states may have been generated by the modelling style, possibly by the verifier itself and by lack of directives by which an engineer can tell a verifier what is important or not.

In fact, in case of filtering, static state exploration or minimisation of instances we are already telling a verifier what is important or not by means which are equivalent to directives but on a rather coarse level. The directives we have in mind shall apply on a finer granularity level: they address better organisation of the source code and of state exploration and allow to express knowledge about the application. Nevertheless, the goal is the same as for the other techniques: it is reduction of the state space as seen by the verifier.

In [1] a first few hints on how to reduce the state space have briefly been mentionned. This paper will provide more background information and will provide with details about the proposed modelling approach.

The hints are divided into two classes: (1) hints which are valid independently of impact of resource consumption, and (2) hints which apply especially if sharing of resources impacts a system's behaviour. Some of the recommendations may be considered as work-arounds because they are just needed to prevent a verifier from doing something which is not desired, but cannot be turned off by the verifier itself. So tools may adopt the idea and then provide better support.

We need to distinguish between two types of systems and their verification: (A) systems for which their behaviour does not depend on shared resources and timing, (B) systems which require consideration of shared resources and of timing constraints. While for type (A) iteration of input sequences during exhaustive simulation is adequate, it is not for type (B).

Firstly, for type (B) systems such iterations are not desired. E.g. a hardware element like a bus prevents arrival of more than one input at a certain instant, and if more than one inputs are pending in an input queue rules exist (like priorisation) which allow only one possible processing sequence. Secondly, when considering sharing of resources the state space may look different from the one spawned by iterations [1].

Therefore techniques like filtering, static exploration or limitation of number of instances cannot be applied for systems of type (B): for exploration of their state space the full number of processes is required.

Hence, in case of type (B) combinatorial iterations will expand the state space to irrelevant domains and will prevent to simplify state exploration. On the other side, while supporting iterations, verfiers do not well support resource consumption and variation of consumption within given limits.

(C) Copyright Dr. Rainer Gerlich BSSE System and Software Engineering 1999. All Rights Reserved

Resource consumption can be introduced by means of SDL timers. However, if a user tries to do it he may increase unnecessarily the state space due to frequent use of SDL timers with their heavy impact on size of state space.

The following chapters shall help users who want to verify systems of type (A) or (B) to reduce the state space without compromising system verification. Chapter 2 makes some principal remarks about what may increase significantly the state space and chapter 3 provides hints: "class 1 hints" apply to both system types, while "class 2 hints" are mainly related to systems of type (B). Finally, chapter 4 disccusses how tools could help and chapter 5 presents first results.

2. Potential Significant Contributors to State Space

In case of "exhaustive simulation" or "exhaustive exploration" the states of the FSM's, the data and possibly the parameters of SDL procedures contribute to the state space. State exploration is performed by iteration of input queue states, recursive execution of the code, and saving of the corresponding state vectors. If for the given scenario (which may not necessarily be complete) no new state vector is identified the verification procedure terminates.

However, the state exploration only terminates if computer resources or available machine time are not exeeceded. Otherwise the exploration process needs to be aborted. The lengthe of a state vector depends on (a) the amount of data declared within a SDL process, (b) the nesting level of procedures and the number of their parameters, (c) the amount of states and (d) the number of SDL processes included in the system.

To reduce the state space we have to look on the contributions from all four drivers of size of state space, but we will concentrate now on (a), (c) and (d).

It is obvious that unlimited or (quasi-continuous) data types like INTEGER or REAL immediately lead to state explosion if the full spectrum of values is allowed.

Remote procedures, VIEW/REVEAL and EXPORT/IMPORT[1] and timers increase the traffic between SDL processes and extend a system's state space. Although unvisible on SDL level the FSM of a process is internally extended to manage the data exchange and time distribution.

A "remote procedure" is treated like a SDL process. Therefore communication with a remote procedure impacts the inputs of a state and the input queue, which all together extends the state space. Same is true for VIEW/REVEAL and EXPORT/IMPORT. To access remote data messages are implicitly exchanged and the FSM's are expanded, too.

The same procedure applies for exchange of timing information. Due to the nature of event-driven simulation, progress of time in *one* process has to be forwarded to *all* other SDL processes which include a SDL timer whenever a timer expires. Evidently, this causes a lot of additional, unvisible traffic.

[1] VIEW/REVEAL or IMPORT/EXPORT may be implemented by means of shared memory, but this is implemetation dependent. It has to be checked for each toolset how they are implemented.

(C) Copyright Dr. Rainer Gerlich BSSE System and Software Engineering 1999. All Rights Reserved

Moreover, such traffic increases the number of queue elements so that the number of possible combinations grows as well and extends the state space.

A user of a SDL toolset easily may check such impact by setting up simple examples and observing the size of the state space and the contribution by each such SDL construct.

Finally, it is quite obvious that the size of the overall system state space is (approximately) the product of the size of the state spaces of all its processes [1]. Therefore the reduction of SDL processes and instances will also lead to a significant reduction of state space.

3. Hints to Reduce the State Space

The known solutions to master state explosion have already been discussed in the introduction. In addition, consideration of resource consumption in case of systems of type (B) alos leads to reduction of state space due to shorter input queues [1].

The hints as given by this chapter approach the problem of state space reduction by the following steps:

1. by getting rid of major contributions by applying a SDL subset only and tuning of the modelling style,
2. by introduction of work-arounds to guide the verifiers the right way

This provides a high potential to decrease the state space and allows to succeed with exhaustive simulation without limiting scope of verification. Compared to static state exploration exhaustive simulation has the advantage of code execution, so that the code is also tested during state exploration. And this is another major motivation to look for other means for reduction of state space than by limiting to static exploration.

3.1 "Class 1 Hints" to Reduce the State Space

These recommendations are valid in general for both types of systems (A and B).

3.1.1 Reducing the Data Traffic

A tool does not know about how to tune the data traffic and a user cannot tune when he does not know about the heavy traffic.

To tune the system do not apply - whenever possible - "heavy" SDL elements like timers, VIEW/REVEAL or EXPORT/IMPORT. If you need to access data from another process it might be better to organise this by yourself than to leave it up to the tool.

If some data change sporadically, only, it is more efficient to forward the updates once than to continuously access the data implicitly via the tool during a state transition.

The same remark is valid for timers. However, timers generate even more traffic: while in case of VIEW/REVEAL or EXPORT/IMPORT only two processes are involved, for update of time all processes need to be involved which include one timer at least. So this could possibly cause traffic to all SDL processes. Hence, a user should be careful when introducing timers. If timers are needed a better organisation of distribution of timing information will allow to drop all timers except for one like it has already been implemented by the EaSySim II environment [3].

EaSySim II optimises distribution of time due to limitation to two-process-communication:

(C) Copyright Dr. Rainer Gerlich BSSE System and Software Engineering 1999. All Rights Reserved

- a server process owns the only timer in the system and receives from all other processes time requests

  i.e. it runs "virtual timers" for all its clients

- if such a virtual timer expires it sends a message to its client, only.

Hence, only two processes are involved when the next virtual timer expires: the server and the client process. This approach is better than the usual approach with dedicated timers per process if more than two processes need a timer.

3.1.2 Reducing the Data States

Large data ranges or continuous data types may contribute significantly to the size of a state space when the processed values are spread fully over the allowed range. However, to succeed with state exploration we need a finite or small number of data values like in case of enumeration types.

So how can we reduce the number of different data values without compromising verification? In fact, we need to make a trade-off:

- shall we make everything visible to the verifier,

  then having a good chance to fail

  or

- shall we restrict the scope of verification to what is important and shall leave minor things outside,

  then getting a good chance to succeed.

Obviously, the second option is the better choice. Now, what does it mean? The following example may help to understand.

Consider the case where in a system a decision true/false may have to be derived by comparing two data of type REAL and this result is needed by another process.

Then it is better to evaluate locally the result and to send only data of type BOOLEAN than to send the data of type REAL to the other process.

Similarly, if data are acquired via an interface which is not part of SDL (e.g. a file or RS232 interface), do not import the REAL data into SDL, but make the decision in C and import only the enumerated or boolean value.

If the behaviour of the system only depends on the boolean value it is not needed to expose the verifier with the full spectrum of REAL data types. In consequence, the state space is reduced by moving from the REAL data type to the enumeration or boolean type.

Also, data should be declared as constants whenever possible. Then the verifier should not add[2] them to the state vector. In case a user needs to assign a value to a variable which is

[2] A user should check if this is true or not for his toolset

(C) Copyright Dr. Rainer Gerlich BSSE System and Software Engineering 1999. All Rights Reserved

never changed again, this variable could be declared as "imported constant" [4] and the assignment could be done via a C operator.

3.2 Guiding the Verifier the "Right Way"

The recommendation given here may only apply to a system of type B. It requires some effort to implement a work-around needed to guide the verfier the "right way". So it is not a solution which may be immediately accepted by everybody. To ease implementation of the idea EaSySim II [3] provides support for its implementation on top of existing SDL tools.

To guide a verifier a user has to organise his SDL system such that the verifier executes and sees the full system, but does not spawn the full state space. Due to lack of background information a verifier may not be able to optimise the verification process. But a user can help the tool to do it!

The specific case addressed here concerns the total state space as a function of the state space of each process. If a number of instances are included in the system then all their state spaces are equivalent. Hence, it is sufficient to consider only one or a few instances of the same process type for systems of type (A). This remains true for systems of type (B) for which - however - we need to consider the full number of instances due to side effects by resource sharing.

Now, we can apply the following optimisation:

1. reduction of instances as seen by the verifier for state exploration
2. consideration of all instances concerning the data flow and generation of Message Sequence Charts (MSC) and consumption of resources.

As the current tools do not support this feature we need to apply a trick: we introduce a "reentrant SDL process" similar to what is already well known in real-time computing. A piece of code is loaded/included only once into memory, but it may serve for several clients due to use of an own data vector for each client. This yields a "high fan-in" in object-oriented terminology and a higher test coverage for the only instance.

The verifier only sees the one instance, while the resources are consumed by the actual number of instances: in consequence, the contribution to the state space is reduced from S^n to S if S is the size of the process' state space.

Depending on the toolset we have to look for approprioate solutions. E.g. we have to analyse whether a toolset allows a process to directly communicate with itself and to save the process' status. Moreover, we have to add an instance number to the signals so that the (server) process knows on reception of a signal for which of its clients it shall process the data.

A general solution which will work for every toolset is supported by EaSySim II [1,3]: to solve the communication problem a system is divided into two parts:

(C) Copyright Dr. Rainer Gerlich BSSE System and Software Engineering 1999. All Rights Reserved

a. a computational part which represents the specific data processing functionality of the system,

   and

b. a communication part which represents the system's network.

In this case a process always communicates with a process of the other part, but never with a process of its own part.

After verification by simulation the network part may disappear and be replaced by hardware. In this case the communication part should be automatically removed and the real number of instances should be put into the system by a utility. However, if a number of instances run on the same processor there is no need to instantiate the real number of processes and the reentrant approach can still be kept.

To save and restore a state, operators are provided by EaSySim II so that on reception of a signal the correct state of an instance can be entered.

By this approach the verifier is advised to map the different state spaces onto a single state space: instead of a product of state spaces we will get an enlargened and more complete linear state space only.

The more instances we will have in our system the more we will gain. Therefore we should drive our design such that we will get a lot of instances derived from a few classes / process types only.

4. Potential Issues

The recommendations for state space reduction represent the best approach for the time being. In future, there could be better support by tools or by SDL standards.

4.1 Tool Issues

We have identified a number of sources which increase the number of states to an undesired high value. To get rid of such undesired contributions we have to change our modelling style which in most cases does not require much motivation. However, to master the "reentrant process" some experience is needed if no supporting environment like EaSySim II is available.

But tools could do much more than they are currently doing regarding reduction of state space. E.g. they could distinguish between user states and tool internal states in case of traffic generated by timers, VIEW/REVEAL or EXPORT/IMPORT.

Furthermore, they could allow a user to specify by directives what shall not be added to the state vector. Then data of type REAL still could be left in SDL and there is no need to hide them by C code and to import an enumerated result instead from C to SDL.

Last but not least they could support the concept of "reentrant processes" and open the door to apply SDL verifiers to systems of type (B) without limitations.

(C) Copyright Dr. Rainer Gerlich BSSE System and Software Engineering 1999. All Rights Reserved

4.2 Standardisation Issues

More support could also come from SDL standardisation by defining more standards by which other languages could interface with SDL verifiers. This idea is driven by

1. that SDL seems to be tailored to systems of type (A),
2. but the formalisation of behaviour by FSM's and formal description of data flow by MSC's is also useful for systems of type (B) which usually are implemented by other languages.

E.g. in languages like C or Ada the concept of reentrant processes is well known and supported. So the rather artificial work-arounds as described in chapter 3.1.2 would not apply.

What is needed - if we do not want to invent the wheel again - is a standardised interface to the verifiers. Then the output which is currently produced during exhaustive simulation could be produced by other languages as well and be feeded into the SDL verifiers or other tools like Aldebaran [5] processing this output.

In case of data flow we have already the MSC standard. Therefore it is already possible today to generate MSC files from other languages like C, an activity which was performed in the CRISYS project[6].

Also, the directives mentionned in chapter 4.1 could be a matter of standardisation.

5. Results

When the system as shown by Fig. 5-1 was implemented as a "warm-up" exercise a rather high number of states was found (Fig. 5-2, improvement step 1) which caused an abort of exhaustive simulation. As this high number of states for a rather simple system was not understood we started to find the reason. So major contributors as described in chapter 3 were identified one after the other. This lead to a reduction of states from more than one million down to about 200 which is a resonable number. Simulation time could be decreased from about 1 hour down to less than one second at the end.

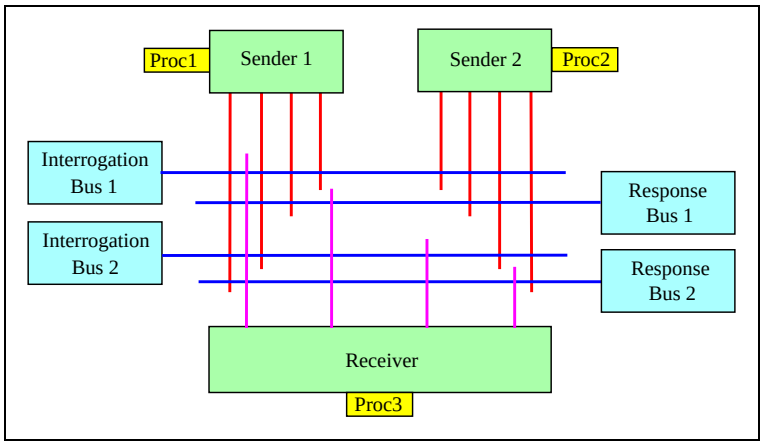

Fig. 5-1: The Sample System

(C) Copyright Dr. Rainer Gerlich BSSE System and Software Engineering 1999. All Rights Reserved

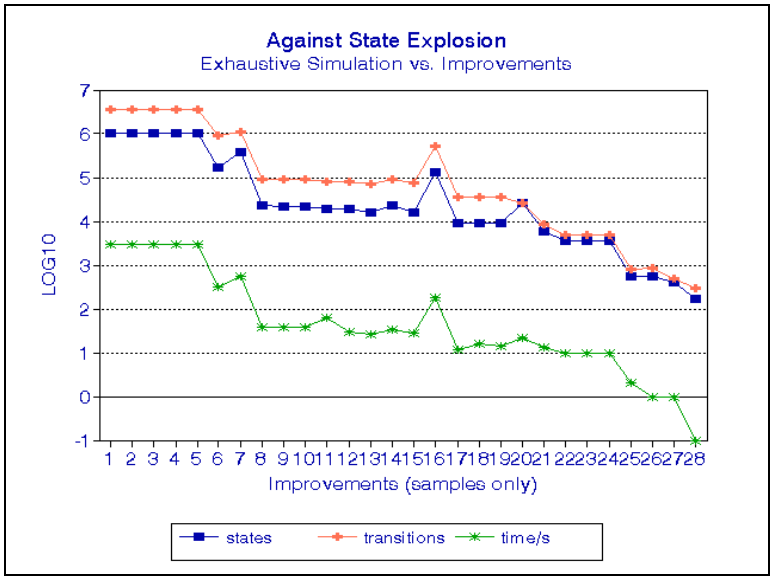

Fig. 5-2: Reduction of System States and Transitions for the Sample Application

For an air traffic control (ATC) application [7] two ATC centres have been modelled each consisting of three components (SDL processes) as shown by Fig. 5-3 and guiding up to one hundred aircrafts. Subject was the verification of the protocol for exchange of flight data. As this system was of type (B) such a high number of aircrafts needed to be considered.

Exhaustive simulation did terminate within about 1 second. The whole system was represented by four reentrant SDL processes only: three for each ATC component and one for each aircraft.

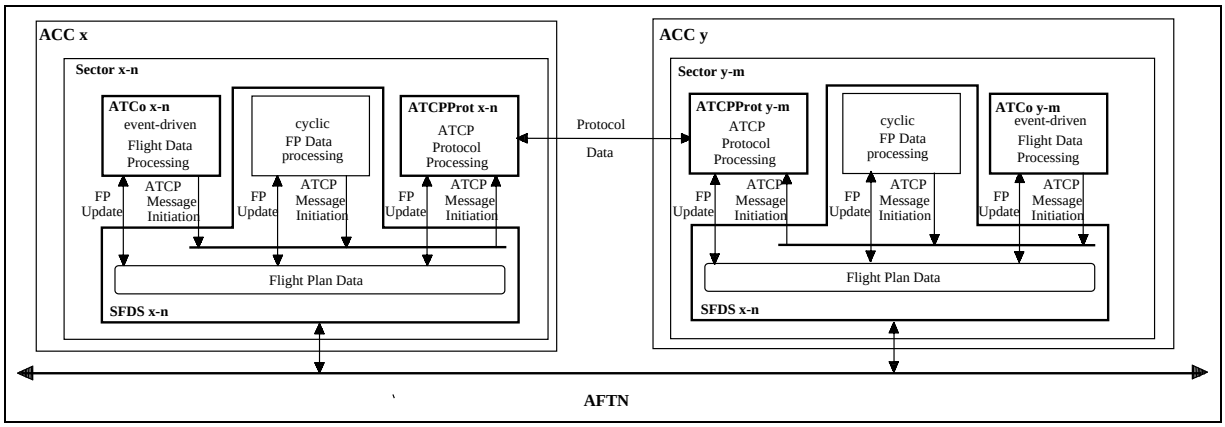

Fig. 5-3: An ATC Application

6. Conclusions

A number of major contributors to state space have been identified and recommendations have been provided how such undesired contributions can be avoided. A certain modelling style has to be accepted by a user when following such recommendations in order to succeed

(C) Copyright Dr. Rainer Gerlich BSSE System and Software Engineering 1999. All Rights Reserved

with reduction of state space. Some of the recommendations are easy to follow, at least one recommendation requires some experience with SDL and some mental motivation.

The recommendations have been divided into two classes: one class is valid for all systems, the second class addresses system verification for which shared resources have to be considered.

Toolsets could support in future such recommendations. The EaSySim II environment extending the SDL toolset from Verilog already supports these recommendations.

Standardisation of interfaces to verifier tools could open the current SDL/MSC verification means for other languages as well. This would allow for application specific verification environments and would avoid to get one big toolset which is a superset of all specific environments.

The guidelines have been applied to a sample system for which continuously the number of states could be decreased when applying the recommendations one after the other and to an ATC example for which exhaustive simulation terminated for more than one hundred "virtual" processes within 1 second on a UltraSparc I/143.

It is planned to apply the recommendations for a space on-board application right now.

References:

[1] R. Gerlich: Tuning Development of Distributed Real-time Systems with SDL and MSC: Current Experience and Future Issues
in A. Cavalli, A.Sarma (editors): SDL'97: Time for Testing - SDL, MSC and Trends

[2] R.Gerlich: EaSySim II SDL Extensions for Performance Simulation
Workshop on Performance and Time in SDL and MSC,
February 17-19, 1998, University of Erlangen, Germany

R.Gerlich: Accuracy of Simulation
Workshop on Performance and Time in SDL and MSC,
February 17-19, 1998, University of Erlangen, Germany

[3] EaSySim II: The Enhanced Environment for System Validation, R. Gerlich BSSE, Auf dem Ruhbuehl 181, D-88090 Immenstaad, Germany

[4] Remark by Rick Reed during SDL Forum '97, Evry, France

[5] Aldebaran, Verimag

[6] ESPRIT project EP 25514 "CRISYS", Project Review #1, December 15, 1998, Brussels, Belgium

[7] OPAL, Protocol Evaluation, 1997, unpublished

(C) Copyright Dr. Rainer Gerlich BSSE System and Software Engineering 1999. All Rights Reserved

.