Difference compression in SPIN

Benoît PARREAUX

Laboratoire d'Informatique de Besançon
Université de Frache-Comté
Route de Gray
25030 Besançon Cedex

E-mail: parreaux@lib.univ-fcomte.fr

SPIN 98 WORKSHOP

Summary: *One of main problems induced by the technique of model checking is state explosion. Work aims to improve the rate of covering the graph of accessibility. Different approaches have been proposed: bitstate hashing [Hol 95], partial order [God 96a], collapse mode [Hol 97], methods based on BDD [God 96b] [Mil 93] and compressing state vectors [Hol 92] [Vis 96].*

Our work originates from this problem. We propose a new compression method using a hash-table. This technique comes from methods for processing animated images. The state compression is done in two steps. Given a hash-code, the first vector using it is chosen as a reference vector. First, we compute the difference between the state vector and the reference vector. Second, we compress the resulting vector using a RLE-like technique. If the generated vector is more sized, the reference vector is enlarged with additional bits. Modifying the reference vector this way preserves the method's reversibility since any vector is always compressed with respect to the same part of the reference vector and always yields to the same compression even if the reference size is increased. This reversible technique can be viewed as an exhaustive way to put model checking into action.

We obtain compression rates close to other classical methods with a quicker graph traversal. In addition, we sketch three possible improvements. Significant improvement can be made with a hash function that chains elements including identical subparts. Second, a dynamic choice of the reference vector would result in increasing the compression rate. A third improvement would be to modify the implementation so that it becomes compatible with the collapse mode.

Key words: Model Checking, Compression and memory storage.

1.INTRODUCTION

The technique of model checking [Ger 95] allows us to verify the communicating process behavior by realizing all possible rendered executions by the model and by verifying properties on these states. During this verification a graph, called graph of accessibility, is created. It contains all states already encountered. This kind of exhaustive verification requires a lot resource memory [Cou 92]. This memory should be used as well as possible to obtain a good rate of cover. Many research works that aim to increase the number of accessible states have been put in action [Hol 97]. Especially a large part of these works have proposed to decrease the size of the representation of states in memory. For this open problem many specific techniques as well as traditional ones have been presented and experimented. We register our called technique compression by difference in this last.

Different techniques of compression have been used to reduce the size of the representation of states in memory. Some of these techniques, e.g. as the bitstate hashing [Hol 95] [Maj 96], are not reversible. These methods are very efficient in term of compression because their non-reversibility is the consequence of the loss of a more or less important part of the information contained in the state vector. When using this kind of compression, we cannot know whether or not the accessibility graph has been reached. Besides, unless an error occurs, we cannot infer anything. Other methods do not

have this drawback and are therefore reversible although less efficient. Traditional methods like the byte masking, compression RLE or Huffman, have been put into action, as well as specific ones like the Collapse. These methods, in addition to reversibility, have all a property in common: it is necessary that the compressed representation of a state remains the even all along the execution, to allow the comparison of states. Techniques based on dictionaries, like Huffman, are not very efficient due to the difficulty to realize an optimum dictionary for all problems whence the emergence of two pass methods. The first pass create the dictionary, the second uses the dictionary to encode stats vectors, and performs the verification. An alternative to traditional methods consists of using the Collapse mode. This technique implemented in SPIN gives very good results. In the next section we are going to outline methods based on reversible compression. We explain guidelines of these algorithms and conclude with advantages and drawbacks of these techniques. In the third section our proposal appears. We begin by presenting underlying ideas and reasons that us have elaborate to develop this technique. We give some implementation details of this technique. The fourth section concludes about results obtained and opens to future direction for this work.

2. EXISTENT COMPRESSION TECHNIQUES

Two ways have been investigated about the choice of compression algorithms. The first has been implemented of already known general algorithms. We will cite three techniques: the bit masking, the Run-Length encoding and the method of static Huffman. We will end with the mode Collapse introduced in the version 2.7.4 of SPIN. This technique is interesting because it is specific to properties of state vectors generated by model checking. It exploits specificity of state vectors to decrease the size of their representation.

The simplest technique, the byte masking [Hol 92] has belonged to the distributed version of SPIN for several years. This method of compression localizes states that are supposed to contain constant values, within a state vector. We can cite for example:

- bytes filled in by the compiler in order that the size of data structures are a multiple of a given word,
- structures of data used to implement Rendez-Vous channels. Constant parts are marked with a byte Mask.

When vectors of state are memorized, only non-marked alone bytes not are copied and others are ignored. To warn the confusion states during the comparison, the original length of each vector of state is prefixed with its compressed version. In the case where the vector of state would be less great than (resp. greater than) 2^{16} bytes one adds two (resp. four) bytes at most to the compressed state vector. These operations "masking" are reversible what guarantees that the compression is without loss. This method allows a moderated reduction of the place used in memory for a weak cost about the time of the execution.

Other solution is to apply the technique of the Run-Length Encoding [Hol 92] on the vector to compress. In all experiences proposed, the size of the vector to store is a multiple of a byte's length. In the method of the Run-Length Encoding, the vector of state is coded as a sequence of byte pairs. The first part of the pair gives how many times the value of the byte is repeated. The second part gives the original value of the byte. Clearly this method is efficient only if on average each byte appears twice at least consecutively in vectors of state. If this is not the case this method can increasing and even doubling the size of the memory necessary to store the vector of state. This method results in increasing significantly the time of execution without for as much give good results to the level of the economy of memory.

A last solution consists of use a method of compression of static Huffman [Hol 92] [Knu 73]. To apply this method a statistical study of the composition of state vectors is necessary. These values will

be used to create the dictionary of the method of coding by attributing the smallest codes to the most used bytes. Then bytes are stocked in a vector whose length is varying. Some simulations showed us that results are better and less time-consuming in comparison with RLE method. The drawback of this method is that results are average since the dictionary is static and define before the execution.

From September 95 in the version 2.7.4 of SPIN, a new mode of experimental compression has been introduced to exploit recursive indexation methods [Hol 96] [Vis 97] for storing state vectors. The method is based on the verification that the complexity of the asynchronous system verification comes mainly from the non-determinism during the execution of process. Each process and each datum can be assigned only a relatively small number of different values. The number of combinations that we can be realized from these local states gets the very great number of accessible states of the system. Therefore repeating the complete description of local components for each global vector is therefore very expensive in memory and time. In the first mode of compression Collapse, each local component storing in the vector, is realized by separately encoding local states represented by each process or channel in hash tables. Each part is be assigned a unique index number. Numbers constitute the global state vector.

A drawback in the first implementation of this method is that the user has to define a possibly maximum index for components of state vectors. The limit is used to determine if indices of components of the vector can be stored on one, two, three or four bytes. The necessity to give an estimation of the size of the vector of state prevents to use the vector as SPIN compression method by default. A solution to avoid the necessity of specify a upper bound for the number of bytes used by indexing to store the state vector components is to store the number of components and the number of bytes used for each component of the index. These numbers can now be authorized to change according to vectors of state, because the exact compression method or uncompression can be always uniquely determined for each vector. The number of bytes used to encode an index never is smaller than a byte and greater than four bytes. To store this information, only two bits per component are necessary. In this new implementation the table of index sizes is added to the global variable component. This vector now contains:

- a local index, from one to four bytes, for each active process, that identifies components that contains this state, including all these variable places, but excluding locally declared channels;
- a global index, from one to four bytes, for the unique component that contains all shared variables, all channels of message and the table of index weights of all local indices;
- a byte that specifies the number of bytes used for the global index.

This technique is called recursive indexation. Traditional techniques as RLE and Huffman are not very efficient so regarding of the time spent for the compression during the verification. A solution to improve results is the introduction of a first pass in the verification, to constituting the dictionary for the compression, then a second pass that uses this dictionary to realize the compression itself. The collapse mode gives good results without increasing the time of computation excessively. This technique mainly uses a property of state vectors: they can be grouped into parts of identical state vectors.

The technique we have developed and we present in next section uses this property.

3. OUR PROPOSAL

It is well known that a technique specific to a problem is more efficient than a general one. In our case it is necessary to study state vectors to try to find properties. A characteristic of state vectors is that two consecutive vectors on a path of execution are in general slightly different. So it would save

memory space if we code a vector according to its difference with the previous vector during the verification. This type of coding is not possible because the same vector of state could have several memory representations according to the way it has been reached. This approach is not suitable for us: each time that a state is generated, we would need to know if it already exists, so we would have to be able to construct all the states found already. That would result in an unacceptable increase of the time of execution. So a method should associate a unique representation with each vector.

States are encoded one according to other in chain of collision of the hash table. Therefore a state is encoded by unique manner since the hash code of the state is unique.

3.1. Presentation

To encode state quickest, we use a vector per collision chain – so-called reference vector – with that will be made differences with all the other vectors. Reference vectors are grouped into a basis. The base is constituted under way of execution and these elements once known does not have to change value. If a vector of the basis changes, we have to recompute the representation of each state vector belonging to the corresponding collision. Each vector of the graph of state will be encoded according to its difference to a – unique – vector of the basis. The rate of compression will be related to the composition of collision chains of the hash table. More similar the elements of these chains better results obtained in term of compression. Likewise there must be denoting at least an element in the chain of collision in order that the compression works. Now we are simply explaining what we mean by coding a state by difference. We are going to begin with computing its difference bit-to-bit between the vector and the state of reference. The first state met in a collision chain will be the state of reference. This state will be stored without compression as the first element of the hash chain. This element must be constant during the whole of the verification, so states could be encoded according to the same process.

As soon as bit-to-bit the difference is done we packed continuations of '0' obtained to reduce the size of the vector of states. Several verifications can be suitable:

- It will be necessary to have a clamping of more of an element in the collision chain so that the method has results.
- Longer the clamping, less efficient the research of elements.
- The function computing the difference will have, to get good performance, to be related to the hash function.
- The function of the bit-to-bit difference calculation is reversible.

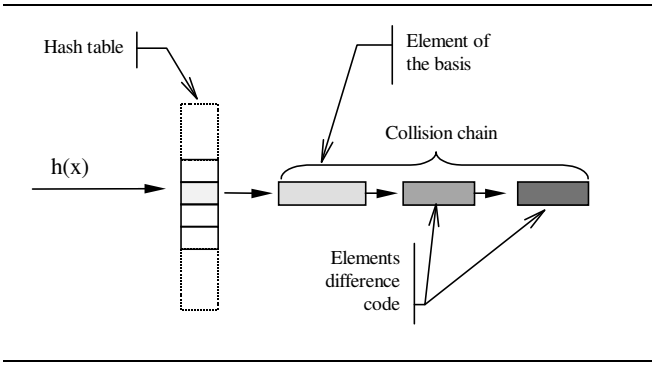

Figure 1 : Compression by difference.

Therefore we have studied hash functions to find out which are the most adapted to our problem. To do that an hash algorithm has been implement in C and will allow us to test the different possible solutions. Different functions with different values of parameters have been tested. Tests have been

realized using a version of SPIN, and have allowed us to find out if our idea was valid. Results have been very convincing, even without modifying the initial hash function of SPIN. Therefore this conclusion has encouraged us to go on with our experience. A more elaborate study of functions usable as hash functions could have improved results. We did not succeed in studying that sufficiently long, and we did not find any function giving straightforwardly good results in comparison with others. We have tested different classic functions and results were equivalent. Of course some were more effective for some problems but less in others. However we think than a more thorough study of the definition of these functions would allow improving the technique again.

One of the problems of this technique is its incompatibility with the Collapse mode. More precisely if this technique is used it is impossible then to realize the Collapse. Indeed, if the difference preserves the different parts of the vector, the compression of the '0' realized afterwards no longer allows to store on the one hand the local part and the other the global part. However it could have be envisaged to realize the difference independently of each part of the vector of state and therefore after having realized the Collapse.

We go to finish to study how the implementation of this technique has been realized in practice and results obtained.

3.2. Implementation and results

We have chosen to implement this method in SPIN. Our choice is related with several factors: the availability of sources, a precis documentation of these sources [Hol a], a certain control of the tools established during different works (for example [Jul 98a]). Besides this platform has been used as a basis to put different techniques into action in order to save memory as much as possible. That allows us more easily to verify efficiency of our method by testing it on problems used to realize other performance studies.

A hash function is a function that has for image area an interval of integer. This value represents then the position in a table of bits. To insure that the function returns a value belonging to a given interval as result, the mathematical function mod(N) is usually considered. Any function is suitable, since its image is a finished domain corresponding to what is expected. Other function is generally used just before this last in order to distribute elements in the table as efficiently as possible. Often a polynomial function is used, but other kinds of function could be used. It is elsewhere on this last that we have attempted to intervene to increase the efficiency of the method.

The implantation of the method has been realized by Stéphane Michaut studying in DESS. We have studied the code of SPIN in detail to find the function in charge of the detection of collisions and the addition of the new states vectors in the hash table. As soon as the function realizing that was found, it has been necessary to modify the code in order to realize the difference and the compression of '0' chain. We have written a function realizing the bit-to-bit difference between the vector running and the vector of basis. This difference is realized by the utilization of a XOR between the two vectors. Another function compresses '0' chain by using a coding similar to that used for RLE but simpler and more rapid. It has then been necessary to verify that these two-chained operations were reversible. Therefore we have compressed and uncompressed different vectors to verify that they were not modified. These operations being stated reversible, we have begun to study what results would be without changing the internal functioning of SPIN truly. Therefore we have computed the size of vectors before and after compression with the size of vectors of the basis. We have recorded good rate of compression (from 20 to 60%) for slightly different times of execution. However we have stated that when the size of the hash table decreased the rate of compression also decreased. After a small study we observed that the fundamental state vector we chose was too small. The difference with other large-sized vectors did not induce a sufficient number of '0' so this technique was not effective. It also seemed to us that the same problem could have occurred when we would generate a great space of states. The solution that we have found to this problem is to increase the size of the vector of basis

when a greater state vector is found. Then we add the additional bits of the state vector running to the vector of basis.

The compression remains reversible because for a given a vector, it will have the same representation, since the bit-to-bit difference operates on a part of the basic vector, and the lengths of this part and one vector are the same. With this modification, results are almost the same in the favourable cases, otherwise they go bad with the ratio Number of states / hash table size but very slowly.

4. CONCLUSIONS AND PERSPECTIVES

We have obtained some results. The obtained gain fluctuates between 20 and 60% according to problems processed. These results have been obtained with the original hash function. More important the number of states, best the rate of compression. This is related to the number of collisions, what is not surprising. When the number of states is not very important a solution is to reduce the size of the hash table. That has indeed a very good influence on results obtained so often the gain is important enough. However, except for some very specific problems the obtained maximum gains are around 50%.

It should be interesting to work on the hash function used. We think indeed that improving this function would result in better performance. It can also seem interesting to implement the variant compatible with the collapse mode. First, a possible solution is to divide the vector into parts so that the compression by difference is separately applied to each part. A slight adjustment of the method induces this result. However it was not possible to change the code of SPIN in order to be able to experiment this option and to compare results obtained to these of the other methods.

Others works are led in this direction. It seems indeed that a function of very simple compression working on preprocessed data results in better compression time, rather than obtained gains.

5. BIBLIOGRAPHY

[Cou 92] C. COURCOUBETIS, Mr. VARDI, P. WOLPER, Mr. YANNAKAKIS, "Memory efficient algorithms for the verification of temporal properties", Formal Methods in System Design I, pp 275-288, 1992.

[Ger 95] R. GERTH, D. PELED, Mr. VARDI, P. WOLPER," Simple one - fly automatic verification of linear temporal logic", Proc. PSTV95 Conference, Warsaw, Poland, June 1995.

[God 93] P. GODEFROID and G.J. HOLZMANN, "One the verification of temporal properties", Proc. IFIP / WG6.1 Symp. One Protocols Specification, Testing and Verification, PSTV93, Liege, Belgium, June 1993

[God 96a] P. GODEFROID, "Partial order Methods for the verification of the Concurrent Systems", LNCS Vol. 1032, Springer Verlag, 1996.

[God 96b] P. GODEFROID, "Symbolic protocol verification with Queue BDDs", Proc. Logic in Computer science, Rutgers Univ., New Brunswick, New Jersey, July 1996.

[Hol 91] G.J. HOLZMANN, "Design and validation of protocols", Prentice entrance software series, 1991.

[Hol 92] G.J. HOLZMANN, P. GODEFROID and D. PIROTTIN, "Coverage preserving reduction strategies for reachability analysis", Proc. 12th Int. Conf. On Protocol Specification, Testing and verification, PSTV 92, Orlando, FI., 1992.

[Hol 95] G.J. HOLZMANN, "An analysis of bit state hashing", Proc. IFIP/WG6.1 Symp. on Protocol Specification, Testing, and Verification, PSTV 95, Warsaw, poland.

[Hol 96] G.J. HOLZMANN, "The model checker SPIN", IEEE Trans. One Software Engineering, Flight. 23, n° 5, 1996.

[Hol 97] G.J. HOLZMANN, "State Compression in Spin", Proc. Third Spin Workshop, Twente University The Netherlands, April 1997.

[Hol a] G.J. HOLZMANN, "Basic Spin manual", Bell laboratories - Murray hill, NJ 07974.

[Hol b] G.J. HOLZMANN, "Overview of the Spin model checker", Technical Transfer, Computing Principles Research Department, Bell Laboratories.

[Jul 98a] J. JULLIAND, B. LEGEARD, T. MACHICOANE, B. PARREAUX, B. TATIBOUET, "Specification of year integrated circuits card protocol application using B and Temporal Linear Logic", 2nd Conference one the B method, Montpellier, April 1998.

[Knu 73] D.E. KNUTH, "The Art of Computer Programming", Vol. 1 Addison-Wesley, 1973.

[Maj 96] B.S. MAJEWSKI, N.C. WORMALD, G. HAVAS and Z.J. CZECH, "A family of perfect hashing methods", The computer Journal, Vol. 39, No. 6, pp 547-554.

[Mil 93] K. McMILLAN, "Symbolic model checking", Kluwer Academic Publishers, 1993.

[Vis 96] W. VISSER, "Memory efficient storage in SPIN", Proc. 2nd SPIN Workshop, DIMACS, Rutgers Univ., New Brunswick, New Jersey, July 1996.