Electronic Notes in Theoretical Computer Science 89 No. 1 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume89.html 16 pages

Compressed and Distributed File Formats for Labeled Transition Systems

Stefan Blom [1,3] Izak van Langevelde [3] Bert Lisser [2]

Department of Software Engineering
CWI
Amsterdam
The Netherlands

Abstract

With model checking techniques growing towards maturity, the availability for standardised file formats for labeled transition systems is more important than ever. A number of requirements for file formats are introduced, based on requirements for software, databases and compression. Two candidate formats, *SVC I* and *SVC II* are introduced, with the former emphasising compression and the latter focusing on distributed access. The two formats are compared with existing file formats.

1 Introduction

The maturation of action based explicit state model checking goes hand in hand with a number of trends, each of which could benefit from the development of compressed and distributed file formats for the models under scrutiny. First, the continuous growth of available computing resources acts as both a blessing and a curse, in that the complexity of both analysing systems and analysed systems grows beyond limits, and although disk space is cheap, having to handle and store extremely large files seriously hampers the applicability of modern checking methods. Second, with an increasing number of tool sets (e.g. [1,6]) related to model checking, the need for open and standardised interfaces between these grows. Third, the scope of the tools extends beyond pure model checking to cover related fields like testing, simulation and visualisation. Fourth and final, the development of distributed tools ([9,3,2]) calls for a format to be used for communicating models between the distributed components of the tool.

[1] Email: sccblom@cwi.nl
[2] Email: bertl@cwi.nl
[3] Research was carried out with financial support of the "Systems Validation Centre".

The issue of file formats for models has been addressed before, but so far no file format has been developed which satisfies the requirements of being an open standard, allowing a compact representation and being specifically tailored towards model checking. The *fc2* format [14] is well-documented, and the *Aldébaran* format [6] is self-explanatory, but both are text-based, without any compression. The *binary coded graph* (BCG) format [6] includes decent compression, but both the file format and the compression algorithm are proprietary. Also, well-known compression schemes like *Gnu Zip* [8] are general-purpose and lack provisions to read or write transitions one at a time.

This paper makes an inventory of the requirements a suitable file format for model checking should satisfy, and presents two suitable candidate formats. The organisation of the paper is as follows. Section 2 proposes the requirements. Section 3 presents the SVC I format as a candidate. Section 4 presents the more sophisticated SVC II format for distributed settings, whose use is illustrated in Section 5. Section 6 evaluates the two formats and Section 7 takes stock.

2 Requirements

The requirements that can be imposed on compact file formats for labeled transition systems fall into three categories. First, there are *sound software* requirements that apply to software in general. Second, there are *database requirements* which apply to the structured data aspect. Third and final, there are specific *labeled transition requirements*.

The sound software requirements spring from both model checking and the efficient storage of models being young and dynamic research fields. The file format, its application programming interface (api) and its implementation will be subject of discussion and, thus, should accommodate future change. The best guarantee for this is openness, in that format, interface and implementation are well-documented and, possibly, open source, for instance by bringing it under a public license from the GNU family [7].

The database requirements focus on efficient, concurrent and reliable access to the elements of a labeled transition system. As for *efficiency*, the common queries of what transitions start from or end at a given state should be readily answerable, without reverting to first reading the full transition system into working memory. Also, the format must be scalable, in the number of processes that have concurrent access, as well as in the size of the transition system. As for concurrency, multiple processes may simultaneously access a labeled transition system without unnecessarily interfering with other processes' actions. As for reliability, in no case should the integrity in the data be destroyed by, say, hard- or software failures.

The requirement of compactness is motivated from the observation that labeled transition systems typically cover more disk space than can be comfortably handled. It should be noted that this requirement loses some of its

weight, since modern distributed file systems, such as PVFS (see [5]), can handle very large file systems efficiently. Also, it should be stressed that compactness is at odds with the other database requirements. Compression is usually at the cost of performance. Also, the nature of compression as removing redundancy has in it the risk that minute mutilations in a compressed file destroy it beyond recovery.

The most specific requirements are about the nature of labeled transition systems. The ideal format should allow a great deal of freedom when it comes to the information on states, action labels, transitions and the whole system. While the process-algebraic approach to model checking typically abstracts away the data contained in states, leaving these as simple integers, other approaches are attached to *state vectors*; as a result, suitable file formats should not rely on any fixed representation. The same holds for transitions, which are in process-algebraic approaches labeled by actions, left unlabeled in other approaches, and sometimes parametrised by other data, like the λ parameter from the exponential distribution in stochastic models. Finally, it should be possible to include descriptive information, like the name of the creating tool, a description of the system and the version number of the format.

3 SVC I

The SVC I format [13] was developed in the scope of the Systems Validation Centre, which lends its name to the format. Its intended use is the compact storage of labeled transition systems. The associated library with documentation can be downloaded from `ftp://ftp.cwi.nl/pub/izak/svc`.

3.1 Contents

An SVC I file consists of a *directory*, a *header*, a *body* and *trailer*. The directory consists of three parts. First, the indexing of the file is given, which indicates whether the states are numerical (*indexed*) or not (*non-indexed*). Second, the format version number is included to facilitate backward compatibility with future version of the file format. Third, the positions in the file of the header, body and trailer are given.

The file header contains the file's metadata, which consists of the following fields:

name

creation date

version number

subtype a clue as to how the contents of the file should be interpreted, e.g. *stochastic*, *probabilistic*.

creator the name of the tool that generated the file.

initial state

comments

number of states the number of distinct states (*read only*)

number of transitions the number of transitions (*read only*)

number of labels the number of distinct transactions (*read only*)

number of parameters the number of distinct action parameters (*read only*)

The file body consists of zero or more transitions, each of which consists of the following:

source the source state of the transition.

action the action label of the transition.

destination the destination state of the transition.

parameter the parameter of the transition, e.g. the parameter of the reverse exponential distribution used in Markov chains.

The file trailer contains a checksum computed over the bytes of the file.
All data, i.e. states, action labels and transition parameters, are terms as defined by the ATerm library [4]. This includes classical function applications like $f(a, g(c))$, but also integers, strings and binary objects, which are efficiently stored by the ATerm library, exploiting maximal sharing of subterms. The current SVC I library uses ATerms as the native data format in its application programming interface; the ATerm sharing scheme is not exploited.

3.2 Compression scheme

All state labels and action labels are compressed using a two-level compression scheme that depends on the file being indexed or not. In the former case, the compression is a combination of *differencing* [10] and *dynamic Huffman encoding* [12], while in the latter case it is a combination of a Lempel-Ziv variation *LZSS* [16] and dynamic Huffman.

Differencing is an operation which is a useful preprocessing step for compression of sequences of integers where the difference between two consecutive integers shows a good deal of similarity. For instance, although the sequence 34, 36, 37, 36, 38, 40, 41, 42 does not show a great deal of similarity among consecutive elements, the differences between consecutive elements, i.e. 2, 1, -1, 2, 2, 1, 1 does show this regularity which lends itself well to compression.

The Lempel Ziv family of algorithms builds on the assumption that a stream of bytes to be compressed consists to a large extent of recurring patterns. Thus, the compressed input is output as a series of *tokens* containing an *offset*, a *length* and a *next byte*. A token is interpreted as the *length* bytes that were encountered at a position *offset* bytes back, followed by the new *next byte*. A byte not previously encountered is output as a token with both offset and length equal to 0. The members of the family differ in the numbers of bytes allocated for the tokens, the data structure used for storing the history of compressed bytes and the algorithm used for searching this history. The

specific member of the family used by SVC I is known as LZSS.

Dynamic Huffman encoding relies on the assumption that some data occurs more frequently than other; the more frequent data gets assigned a shorter code. This encoding was originally designed for situations where the frequency distribution of the data to be compressed is known beforehand. Later, a dynamic algorithm was formulated [12], where this frequency distribution is learned and used on-the-fly, and where each first occurrence of a datum is output unencoded.

The compression for indexed files works as follows. For each of the transitions, the difference between the current and the previous source state, and the difference between the current and the previous destination state are computed. Then, the source difference is compressed and output through dynamic Huffman encoding, the action label is compressed and output through dynamic Huffman, using LZSS for every first occurrence, and the transition parameter is compressed through dynamic Huffman, using LZSS for every first occurrence. For action labels and transition parameters, different LZSS search buffers are used.

The compression for non-indexed files works as follows. First, the source state is compressed through dynamic Huffman, using LZSS for every first occurrence. Second, the action label is compressed through dynamic Huffman using LZSS. Third, the destination state is compressed through dynamic Huffman using LZSS, and fourth, the transition parameter is compressed through dynamic Huffman using LZSS. States, action labels and transition parameters are compressed through three different LZSS search buffers.

4 SVC II

Both SVC I and BCG [6] were developed as compact storage formats, without facilitating distributed access. Although these formats can be used to obtain scalability and memory efficiency, by storing a transition system across multiple files, this does not provide selective access. For example, the workers of the distributed strong bisimulation reduction tool [2] need access to the destination states of their incoming transitions and to the source states and labels of their outgoing transitions. As SVC I and BCG store transitions as triples, getting efficient access to part of the triple is difficult. In order to overcome these limitations, the development of SVC II was initiated.

4.1 Contents

The transition storage of the SVC II format is a distributed version of the transition storage of SVC I. The distribution is based on dividing the set of states into N segments. The list of transitions is split into N^2 sub-lists, one sub-list for each pair of segments. Each sub-list of transitions is divided into a sub-list of source states, a sub-list of action labels and a sub-list of destination

states. Transition parameters are not yet available in the SVC II format, but adding them is trivial. We have chosen to divide the transitions into $3N^2$ sub-lists, because this allows distributed model checkers and state space reduction tools to access the information they need without having to extract it from more complicated data-structures. The current implementation stores all $3N^2$ sub-lists as files in a directory.

Using a directory and a lot of files simplifies efficient distributed access and experimentation with compression methods. However, for large clusters this approach would lead to an extremely large number of files (several millions for a thousand node cluster). Existing file systems are not capable of dealing with that many files if they need to be open at the same time. This problem can be solved in a number of ways. The simple solution is to store multiple logical streams within a limited number of real files in the file system. A more complicated solution is to develop a file system, which is capable of dealing with this many open files. This may seem too complicated given the easy solution, but it has the additional advantange that the file system can be implemented in such a way that the workers within a distributed application cooperate to get their data rather than that they compete for it.

Another advantage of using a directory is that it is easy to store additional information in the SVC II directory. Apart from the files with transition information the SVC II format has two additional mandatory files: an info file containing (among others) initial state and transition counts and a file containing a index of action labels.

State information can be stored in both SVC formats. For each state, the SVC I format can store a single term and the SVC II format can store a fixed

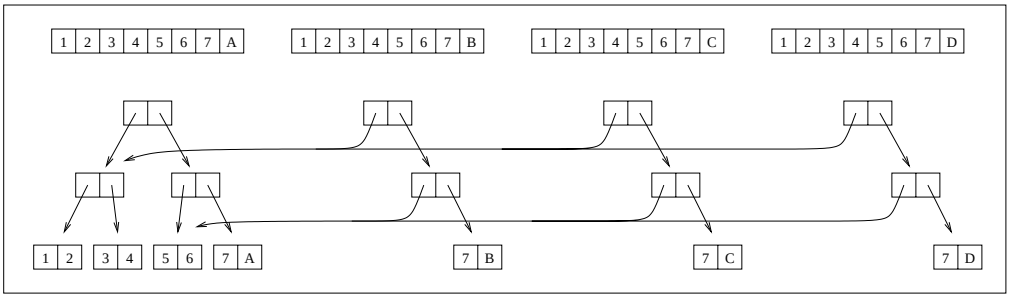


Fig. 1. Shared vector storage

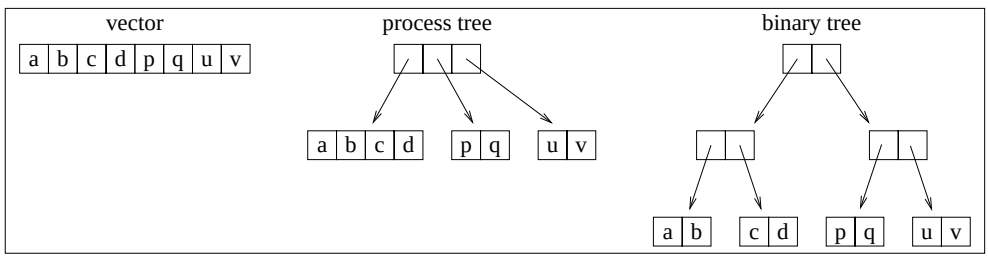


Fig. 2. State representations for the system $X(a, b, c, d) \parallel Y(p, q) \parallel Z(u, v)$.

length vector of terms. To store vectors the SVC II format represents vectors as trees and shares sub-trees as is done in the ATerm library [4]. The basic idea behind this state representation has existed for a long time. For example, the Xesar tool for large state spaces described in [11] exploits what we call a process tree. That is, the vector of all variables is divided once into sub-vectors of the variables belonging to a process. The instantiator of the μCRL toolset takes the idea one step further and recursively divides the vector in two parts, yielding a binary tree. We have illustrated the difference between the vector, process tree and binary tree representations of a state in Fig. 2. Sharing is illustrated in Fig. 1: On the top of the picture there are four state vectors. Each vector uses 8 cells of memory. On the bottom of the picture, we have the same vectors in the shared format. In this case the first vector uses 14 cells and every additional vector uses 6 cells. For large state spaces, the performance is excellent. For example, a state space with 33.9 million states and vectors of length 78 uses only 35.4×8 million bytes to store the state vectors. That is less than a bit of space for every vector element.

In Fig. 3 we have drawn a small state space. The circles are states, the arrows are transitions and the rectangles indicate segments. The numbers inside the circles are state numbers relative to the segment. In the same figure, we have shown how this state space is stored in the SVC II format.

4.2 Compression scheme

The implementation of the SVC II format which is used in the distributed tools of the μCRL toolset does not include compression. However, it is easy to apply differencing and standard compression tools to the separate files in an SVC II directory, so we are able to compress files for long term storage. These existing tools (gzip [8], bzip2 [15]) deliver pretty good compression. By using the streams interface offered by the compression libraries that are related to these tools (zlib, bzlib) it is very easy to integrate these compression methods into the SVC II library.

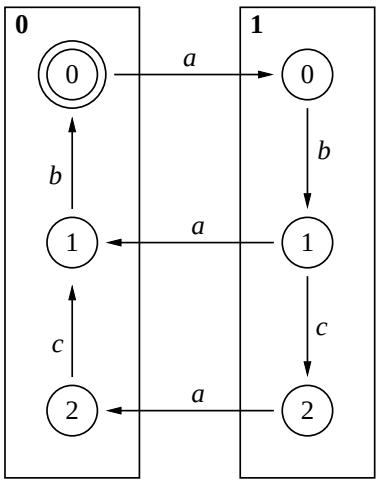
However, these standard tools do not allow random access to compressed files and because they only work on a single file they cannot exploit similarities between collections of files (such as all files containing action labels). Hence, as a part of future work we propose to investigate the following compression scheme for a collection of files containing integers.

(i) Count the number of occurrences of every integer in all files.

(ii) Build a Huffman compression table based on these counts.

(iii) Compress the files in the collection in blocks of for example 1000 integers and write pointers to the begin of each block.

The result is a compression scheme which exploits similarities between collections of files and we can seek in the compressed file with a reasonable efficiency.



description	SVC II data	
header	segments:	2
	root:	(**0**, 0)
transitions from	src-**0**-**0**:	[1, 2]
segment **0**	label-**0**-**0**:	[*1*, *2*]
to segment **0**	dest-**0**-**0**:	[0, 1]
transitions from	src-**0**-**1**:	[0]
segment **0**	label-**0**-**1**:	[*0*]
to segment **1**	dest-**0**-**1**:	[0]
transitions from	src-**1**-**0**:	[1, 2]
segment **1**	label-**1**-**0**:	[*0*,*0*]
to segment **0**	dest-**1**-**0**:	[1, 2]
transitions from	src-**1**-**1**:	[0, 1]
segment **1**	label-**1**-**1**:	[*1*, *2*]
to segment **1**	dest-**1**-**1**:	[1, 2]
	index:	[*a*, *b*, *c*]

Fig. 3. State space

5 An application of SVC II

The μCRL tool set facilitates the distributed generation of state spaces from μCRL specifications. The generation of state spaces is designed to be crash proof, which means that after a crash the run can be restarted from the last dumped checkpoint.

The distributed system consists of N machines, a joint file system (nfs) mounted on each machine, and each machine running the μCRL tool set. Each machine runs a single-threaded state space generator, called *instantiator*, and a database server, which provides access to a local database; it is efficient to combine the instantiator and the database server into one process. There is one *manager* process running. In the context of this explanation it is sufficient to consider an instantiator as a black box which receives a set of states and sends a set of transitions, and to consider a database server as a black box which receives transitions, writes transitions in SVC II format, and sends back new states to be explored. This configuration is illustrated in Figure 4.

The set of states is partitioned into N segments. A hash function on the

data components of a state (modulo N) defines the segment a state belongs to. The local database on machine k contains the states of segment k. A *state* is represented by a data vector followed by a pair *(segment, index)*, and a *transition* is represented by a triple *(source, action label, destination)*, where *source* is a state and *destination* is a data vector.

The distributed state space generation process works as follows. The manager successively:

(i) Sends each state received from the database servers (or the initial state) to one of the instantiators.

(ii) Waits until all instantiators are finished.

(iii) Sends each transition received to one of the N distributed database servers, as determined by its destination state.

(iv) Waits until all database servers are finished.

(v) Dumps check point information.

This schedule is not optimal, but bypasses complicated deadlock situations. Generating the state space is breadth first. The generated states are divided in levels. Level 0 is the set which exists of the initial state, level $n+1$ is the set of destination states of all transitions whose source state is member of level n.

When all states of a certain level are processed by the database servers there is an opportunity for dumping a checkpoint. Dumping a checkpoint includes dumping all databases and adding a line to the checkpoint file with the highest indices occuring in the databases before and after the states are added by the servers, the indices of the last explored states, N^2 integers pointing to the end of the $3N^2$ transition files, and an identifier of the checkpoint.

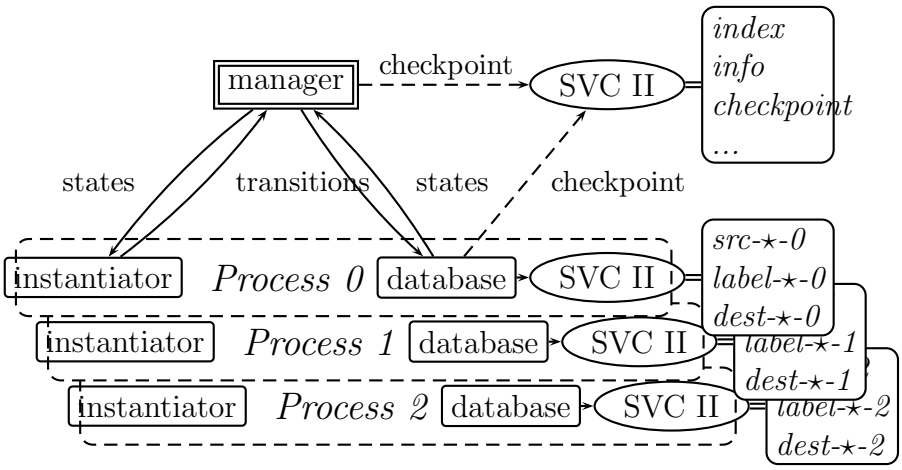


Fig. 4. Distributed state space generation

6 Comparison with other work

For evaluating the compression ratios we used 40 state spaces from the first release of the CWI/VASY test collection[4], ranging in size from 1,224 to 165,318,222 transitions.

We have compared the SVC formats with the Aldébaran format and the BCG format. In this comparison, we have used SVC II with differencing applied to both source and destination states of transitions (SD). The SVC II files in this comparison had one segment and were packed into a tar file. In order to get the smallest possible files, we applied bzip2 (with -9 option) to all files. The raw file size data is included as Appendix A.

In Fig. 5, we have plotted the sizes of the various files in BCG format. In order to allow comparison of files of various sizes, we plotted the number of bytes per transitions rather than the absolute file size. In Fig. 6, we have plotted the actual comparison. That is, for each file format, we have divided its number of bytes per transition by the number of bytes per transition for BCG. The lines for SVC I and Aldébaran (AUT) are nearly always above the lines for SVC II and BCG. This means that those two formats are clearly not as efficient as SVC II or BCG. However, the line for SVC II keeps crossing that for BCG. So from the picture, we cannot conclude which format is more efficient. The sum of the files sizes is 475,572,976 for BCG and 429,142,694 for SVC II. This seems to suggest that SVC II is more efficient. However, the last example counts for roughly half the total size and if we omit that particular example then the sums are 229,370,029 and 228,275,174 respectively. This is a negligible difference.

Using the largest example (33.9 million states and 165 million transitions), we also tested the effect of segmentation on compression. The results are shown in Fig. 7. Not surprisingly, compression is less effective if there are more segments. However, the effect of difference encoding gets bigger if the segment count increases.

7 Conclusions and future research

Two file formats for labeled transition systems were proposed, each of which with a slightly different focus. The SVC I format was developed with compact storage in mind, implementing an efficient compression scheme. The younger SVC II format was designed for distributed access, segmenting the system into segments which can be independently accessed.

For both of the formats proposed their compression performance is compared with existing formats. The comparison shows that for relatively small systems of less than 10^6 transitions the SVC I compression scheme is the most

[4] VASY is the systems validation group at INRIA Rhône-Alpes, which also maintains the CADP toolset. The test collection is accessible through the following URL: `http://www.inrialpes.fr/vasy/cadp/resources/benchmark_bcg.html`.

Fig. 5. File size vs LTS size.

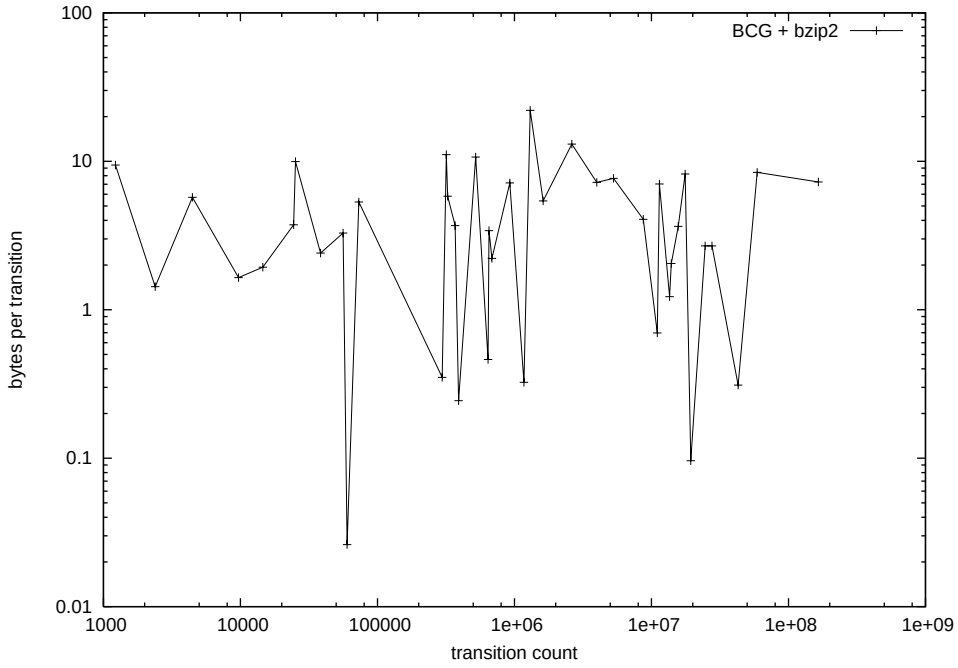
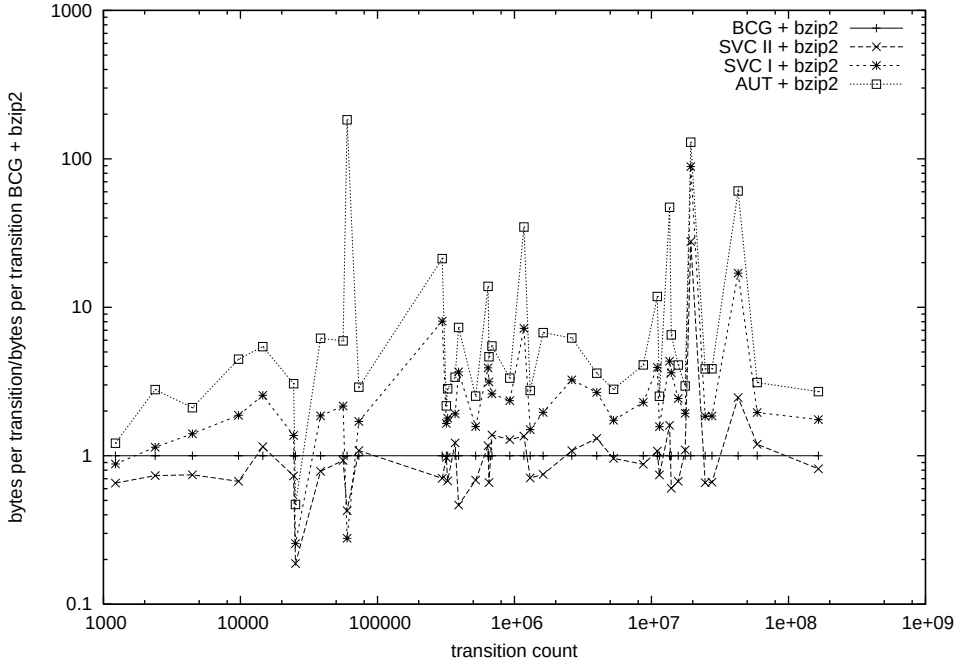
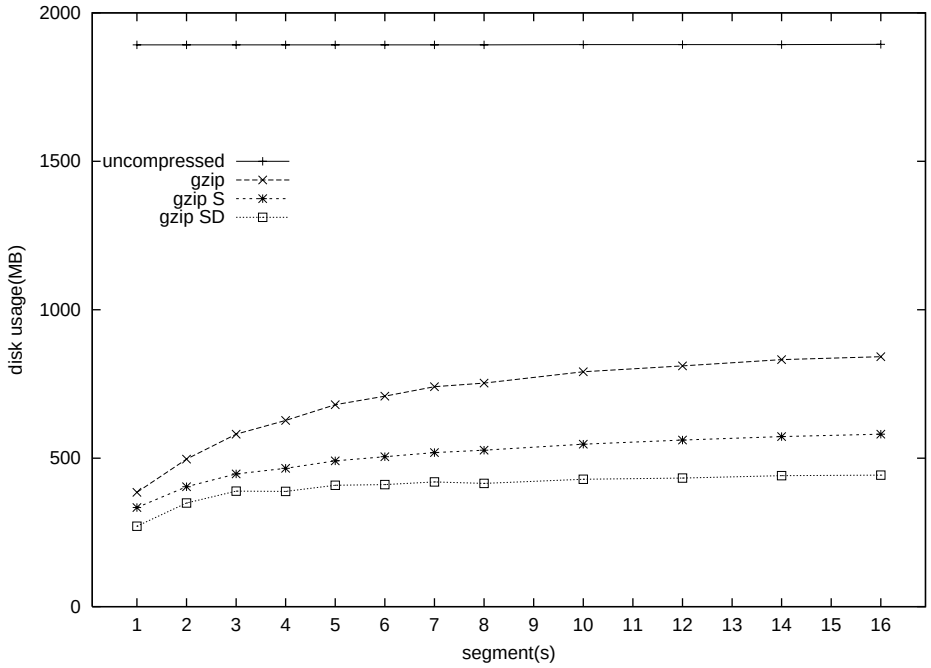


Fig. 6. A comparison of file formats.



effective one, based on the amount of redundency left (gzip and bzip2 have almost no effect). However, BCG files with additional gzip or bzip2 compression are both better than all others. For systems with more transitions, the performance of BCG with additional compression is very good, but for both

Fig. 7. The effect of segmentation on compression.



gzip and bzip2 compression the performance of the best variant of SVC II is better. As the best variant is always the same one (differencing applied to both source and destination states), this will be the variant that will be implemented.

Future research will concentrate on the further development of SVC II, benefiting from the lessons learned form the SVC I compression scheme. With disk space growing cheaper and distributed processing being the new trend in model checking, it is definitely worth to emphasise concurrent access over compactness. However, quite often bandwidth is a serious problem as well, so the optimal solution may well be a compression algorithm which is optimized for speed rather than compactness. Although experiments are promising, there is still work to be done in incorporating compression into the SVC II implementation. Also, a well-documented application programming interface is to be completed.

So far, this paper has concentrated on technicalities. However, what inspired the development of SVC I and SVC II more than the need for compactness was the need for openness. We believe that the field of model checking could be pushed towards new horizons through the use of standardised file formats through well-defined interfaces, which allow the rich variety of powerful model checking tools to cooperate in fruitful unison. The SVC formats are only the beginning.

Acknowledgement

We thank Jaco van de Pol who read and commented upon an earlier version of this text. We also thank anonymous referees for their extensive comments which were instrumental in improving the paper and provided some directions for future work.

References

[1] Blom, S. C. C., W. J. Fokkink, J. F. Groote, I. A. v. Langevelde, B. Lisser and J. C. v. d. Pol, *μCRL: A toolset for analysing algebraic specifications*, in: G. Berry, H. Comon and A. Finkel, editors, *Computer Aided Verification (CAV 2001)*, lncs **2102**, 2001, pp. 250–254.

[2] Blom, S. C. C. and S. M. Orzan, *A distributed algorithm for strong bisimulation reduction of state spaces*, in: *Proceedings of the International Workshop on Parallel and Distributed Model Checking (PDMC 2002)*, Electronic Notes in Theoretical Computer Science **68** (2002).

[3] Bollig, B., M. Leucker and M. Weber, *Local parallel model checking for the alternation-free μ-calculus*, in: *Proceedings of the 9th International SPIN Workshop on Model Checking of Software (SPIN '02)*, Lecture Notes in Computer Science **2318** (2002).

[4] Brand, M. G. J. v. d., H. A. d. Jong, P. Klint and P. A. Olivier, *Efficient annotated terms*, Software – Practice & Experience (2000), pp. 259–291.

[5] Carns, P. H., W. B. L. III, R. B. Ross and R. Thakur, *PVFS: A Parallel File System For Linux Clusters*, in: *Proceedings of the 4th Annual Linux Showcase and Conference*, 2000, pp. 317–327.

[6] Fernandez, J.-C., H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier and M. Sighireanu, Cadp *(Cæsar/Aldébaran development package): A protocol validation and verification toolbox*, in: R. Alur and T. A. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification*, Lecture Notes in Computer Science **1102** (1996), pp. 437–440.

[7] Free Software Foundation, `http://www.gnu.org`, "GNU's not Unix – the GNU Project and the Free Software Foundation (FSF)," .

[8] Gailly, J.-l., "The gzip home page," `http://www.gzip.org`.

[9] Garavel, H., R. Mateescu and I. Smarandache, *Parallel state space construction for model-checking*, in: M. B. Dwyer, editor, *Proceedings of the 8th International SPIN Workshop on Model Checking of Software SPIN'2001*, LNCS **2057** (2001), pp. 217–234.

[10] Gottlieb, D., S. A. Hagerth, P. G. H. Lehot and H. S. Rabinowitz, *A classification of compression methods and their usefulness for a large data processing center*, , **44**, 1975, pp. 453–458.

[11] Graf, S., J.-L. Richier, C. Rodriguez and J. Voiron, *What are the limits of model checking methods for the verification of real life protocols?*, in: J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science **407** (1990), pp. 275–285.

[12] Knuth, D. E., *Dynamic Huffman coding*, Journal of Algorithms **6** (1985), pp. 163–180.

[13] Langevelde, I. A. v., *A compact file format for labeled transition systems*, Technical Report SEN-R0102, CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands (2001).

[14] Madelaine, E., *Verification tools from the Concur project*, EATCS Bulletin **47** (1992).

[15] Seward, J., "The bzip2 and libbzip2 home page," `http://www.bzip2.com`.

[16] Storer, J. and T. Szymanski, *Data compression via textual substitution*, Journal of the ACM **29** (1982), pp. 928–951.

A Disk usage statistics

In table A.1 we list the sizes of the files in various formats. In each case we used bzip -9 for extra compression. The SVC II format used differencing for both source and destionation states.

Table A.1
Raw data for file formats with bzip2 compression.

problem	transitions	states	BCG	SVC II (SD)	SVC I	Aldébaran
vasy_0_1	1,224	289	2,725	1,784	2,399	3,310
cwi_1_2	2,387	1,952	2,791	2,049	3,185	7,785
vasy_1_4	4,464	1,183	6,768	5,039	9,525	14,264
vasy_5_9	9,676	5,486	9,039	6,073	16,922	40,394
cwi_3_14	14,552	3,996	7,732	8,898	19,746	42,019
vasy_8_24	24,411	8,879	33,165	24,272	45,420	101,418
vasy_25_25	25,216	25,217	250,981	47,145	64,255	117,980
vasy_8_38	38,424	8,921	21,507	16,895	39,945	133,198
vasy_10_56	56,156	10,849	35,631	33,139	77,061	211,549
vasy_40_60	60,007	40,006	1,047	446	291	192,023
vasy_18_73	73,043	18,746	99,696	10,8171	169,834	288,791
vasy_157_297	297,000	157,604	55,255	38,929	445,551	1,174,791
vasy_52_318	318,126	52,268	580,142	564,968	956,733	1,258,834
vasy_83_325	325,584	83,436	485,395	328,274	870,380	1,371,117
vasy_116_368	368,569	116,456	429,858	524,769	823,192	1,455,143
vasy_720_390	390,999	720,247	175,745	81,816	645,695	1,285,280
vasy_69_520	520,633	69,754	745,465	510,990	1,173,754	1,878,390
cwi_371_641	641,565	371,804	171,979	200,657	672,021	2,380,098
vasy_166_651	651,168	166,464	567,699	374,297	1,786,144	2,637,617
cwi_214_684	684,419	214,202	475,201	656,697	1,241,569	2,609,445
cwi_142_925	925,429	142,472	1,018,182	1,308,262	2,394,769	3,391,676
vasy_386_1171	1,171,872	386,496	125,502	170,019	905,735	4,361,385
vasy_66_1302	1,302,664	66,929	1,475,761	1,043,846	2,215,563	4,061,729
vasy_164_1619	1,619,204	164,865	891,178	666,526	1,744,980	6,017,736
vasy_65_2621	2,621,480	65,537	855,425	926,309	2,778,370	5,319,981
cwi_566_3984	3,984,157	566,640	4,084,401	5,345,408	10,902,630	14,688,953
vasy_1112_5290	5,290,860	1,112,490	8,529,969	8,191,776	14,798,721	23,899,068
cwi_2165_8723	8,723,465	2,165,446	8,802,855	7,703,884	20,169,883	36,004,231
vasy_6120_11031	11,031,292	6,120,718	4,270,004	4,588,144	16,800,333	50,474,368
vasy_2581_11442	11,442,382	2,581,374	18,157,616	13,487,879	28,502,525	45,653,842
vasy_574_13561	13,561,040	574,057	703,781	1,129,103	3,053,843	33,220,283
vasy_4220_13944	13,944,372	4,220,790	8,653,748	5,218,503	31,345,987	56,383,512
vasy_4338_15666	15,666,588	4,338,672	15,792,942	10,619,323	38,333,641	64,536,772
cwi_2416_17605	17,605,592	2,416,632	19,864,506	21,711,323	38,319,501	58,635,358
vasy_6020_19353	19,353,474	6,020,550	579,352	16,060,305	51,431,680	74,982,282
vasy_11026_24660	24,660,513	11,026,932	29,713,319	19,579,631	54,766,756	114,184,715
vasy_12323_27667	27,667,803	12,323,703	33,208,409	22,026,956	61,564,880	128,052,854
vasy_8082_42933	42,933,110	8,082,905	2,511,126	6,185,382	42,673,930	152,543,298
cwi_7838_59101	59,101,007	7,838,608	65,974,132	78,777,287	128,898,100	205,495,252
cwi_33949_165318	165,318,222	33,949,609	246,202,947	200,867,520	431,042,570	665,931,000
total			475,572,976	429,142,694	991,708,019	1,765,041,741

In table A.2 we list varaints of SVC II with differencing applied to source states only and no differencing. These variants are compressed with bzip -9. In the same table we list the sizes of BCG and SVC I without the extra compression. The size of an SVC II file without compression is 12 times the number of transitions plus some overhead.

Table A.2
Raw data for SVC II variants with bzip2 and BCG/SVC without compression.

problem	transitions	states	SVC II (S)	SVC II ()	BCG	SVC
vasy_0_1	1,224	289	2,035	2,582	4,538	2,070
cwi_1_2	2,387	1,952	4,037	4,884	8,264	3,089
vasy_1_4	4,464	1,183	6,656	9,219	10,422	9,126
vasy_5_9	9,676	5,486	13,695	17,614	21,485	16,691
cwi_3_14	14,552	3,996	17,693	25,064	14,305	20,563
vasy_8_24	24,411	8,879	38,865	51,182	48,402	45,936
vasy_25_25	25,216	25,217	56,906	57,149	3,537,764	208,303
vasy_8_38	38,424	8,921	30,057	37,894	71,178	55,466
vasy_10_56	56,156	10,849	50,824	77,363	123,761	97,507
vasy_40_60	60,007	40,006	27,714	55,477	82,300	52,620
vasy_18_73	73,043	18,746	117,769	158,983	121,055	169,840
vasy_157_297	297,000	157,604	265,397	356,145	553,033	492,911
vasy_52_318	318,126	52,268	579,877	651,171	712,853	954,405
vasy_83_325	325,584	83,436	528,106	635,158	841,734	870,525
vasy_116_368	368,569	116,456	712,736	855,830	551,591	828,115
vasy_720_390	390,999	720,247	551,741	569,962	1,310,220	832,067
vasy_69_520	520,633	69,754	735,504	846,238	1,163,589	1,200,355
cwi_371_641	641,565	371,804	808,353	1,115,983	965,620	921,971
vasy_166_651	651,168	166,464	991,681	1,191,570	1,549,698	1,802,495
cwi_214_684	684,419	214,202	1,163,774	1,420,224	926,708	1,309,200
cwi_142_925	925,429	142,472	1,700,682	1,937,919	1,174,601	2,395,587
vasy_386_1171	1,171,872	386,496	981,321	1,231,548	1,746,400	1,807,280
vasy_66_1302	1,302,664	66,929	1,153,182	1,297,693	2,680,330	2,545,877
vasy_164_1619	1,619,204	164,865	1,411,067	1,659,982	3,397,963	2,739,023
vasy_65_2621	2,621,480	65,537	1,484,431	1,650,556	5,398,814	4,158,069
cwi_566_3984	3,984,157	566,640	7,687,317	8,598,598	4,856,901	10,938,552
vasy_1112_5290	5,290,860	1,112,490	10,025,115	11,465,772	13,267,003	15,096,205
cwi_2165_8723	8,723,465	2,165,446	14,482,753	17,096,418	15,450,851	21,146,758
vasy_6120_11031	11,031,292	6,120,718	18,965,291	27,539,046	19,718,896	22,183,004
vasy_2581_11442	11,442,382	2,581,374	19,203,128	21,584,107	26,503,580	29,575,152
vasy_574_13561	13,561,040	574,057	4,452,670	5,307,460	22,985,028	19,732,682
vasy_4220_13944	13,944,372	4,220,790	22,042,242	25,748,454	31,562,893	33,603,704
vasy_4338_15666	15,666,588	4,338,672	26,438,386	30,553,650	38,010,113	42,292,635
cwi_2416_17605	17,605,592	2,416,632	29,767,148	33,650,231	22,833,655	39,158,022
vasy_6020_19353	19,353,474	6,020,550	36,684,963	43,629,014	22,602,113	56,351,588
vasy_11026_24660	24,660,513	11,026,932	47,659,341	58,282,716	52,982,347	62,613,627
vasy_12323_27667	27,667,803	12,323,703	53,500,707	65,351,948	59,451,496	70,645,459
vasy_8082_42933	42,933,110	8,082,905	35,117,937	42,966,494	78,511,060	77,715,527
cwi_7838_59101	59,101,007	7,838,608	107,495,422	118,859,935	117,020,662	136,517,110
cwi_33949_165318	165,318,222	33,949,609	280,778,487	322,877,817	319,833,836	449,982,456
total			727,735,010	849,429,050	872,607,062	1,111,091,572