# Crocodile: a Symbolic/Symbolic tool for the analysis of Symmetric Nets with Bag*

M. Colange[1] and S. Baarir[2], F. Kordon[1], and Y. Thierry-Mieg[1]

[1] LIP6, CNRS UMR 7606, Université P. & M. Curie – Paris 6
4, place Jussieu, F-75252 Paris Cedex 05, France
Maximilien.Colange@lip6.fr, Fabrice.Kordon@lip6.fr, Yann.Thierry-Mieg@lip6.fr
[2] LIP6, CNRS UMR 7606 and Université Paris Ouest Nanterre La Défense
200, avenue de la République, F-92001 Nanterre Cedex, France
Souheib.Baarir@lip6.fr

**Abstract.** The use of high-level nets, such as colored Petri nets, is very convenient for modeling complex systems in order to have a compact, readable and structured specification. Symmetric Nets with Bags (SNB) were introduced to cope with this goal without introducing a burden due to the underlying complexity of the state space. The structure of bags allows through exploitation of symmetries to provide a compact quotient state space representation (similarly to the construction proposed in GreatSPN).
In this paper, we present Crocodile, the first implementation of a modeling environment and model checker dedicated to SNB. Its goal is first to be a proof of concept for experimenting the quotient graph techniques together with hierarchical set decision diagrams. A second objective is to enable experimentation of modeling techniques with this new class of Petri nets.

**Keywords**: Symmetric Nets with Bags, Model Checking, Symmetries-based techniques, Hierarchical Set Decision Diagrams.

## 1 Introduction

Symmetric nets with Bags (SNB) [7] are a compact and readable dialect of colored Petri nets allowing structured specification of complex systems. They are based on Symmetric Petri nets (SN), formerly known as Well-Formed Petri Nets [2], a subclass of High-level Petri Nets[3]. Like SN, they allow construction of a quotient state graph representation [8], automatically derived from the specification, that preserves many properties of interest (e.g. LTL), because bags[4] in tokens remain compatible with symmetry-based reduction techniques.

This paper presents the tool Crocodile, which allows creation and analysis of SNB. The definition of SNB is quite recent [7], and this is the first tool that allows their manipulation. It is composed of an Eclipse plugin for front-end modeling (based on the

* Supported by the FEDER Île-de-France/System@tic—free software NEOPPOD project.

[3] “Symmetric Nets” have been chosen in the context of the ISO standardization.

[4] ’bag’ is a synonym for ’multiset’

Coloane editor [9]), and it uses hierarchical Set Decision Diagrams (SDD) [5] in the back-end to support construction of the quotient state graph.

The paper is structured as follows. Section 2 informally presents SNB and their relation to SN. Then, section 3 describes the architecture of the tool as well as its original encoding of the quotient graph. Section 4 provides some information about performances of Crocodile.

## 2 Informal Presentation of Symmetric Nets with Bags (SNB)

This section informally presents SNB. Due to lack of place, we do not introduce the formal definitions that can be found in [7].

**The SaleStore example** Let us present SNB by means of a simple example, the SaleStore (see Figure 1). People enter the sale store through an **airlock** with a capacity of two (of course, only a single person may enter too). Then, people may buy items (at most two but possibly zero if none fits their need) and leave with the acquired items. Let us note that this example has two scalable parameters: **P**, the number of involved people in the system and **G**, the number of available gifts in the warehouse.

```
Class                                  Var
  People is 1..P;                        p in People;
  Gift is 1..G;                          BP in unique BagPeople;
Domain                                   BG in BagGift;
  BagPeople is bag (People);
  BagGift is bag(Gift);
  PeopleBagGift is <People, BagGift>;
```

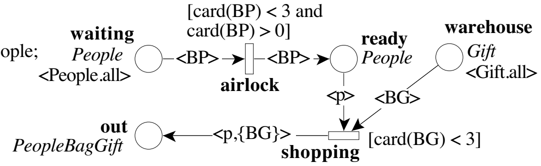


**Fig. 1.** The SaleStore example modelled with a SNB

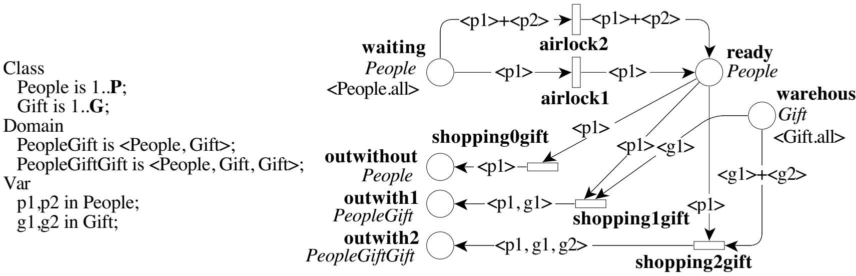
The model in Figure 1 illustrates most of the main features of SNB. First, there are several color types giving the place's domains: simple color types like *People* or *Gift* are called classes, while bags such as *BagPeople* or *BagGift* and cartesian products such as *PeopleBagGift* are built upon basic color classes.

Variables which are formal parameters for transition binding are declared in the *Var* section. A basic variable such as *p* can be bound to represent any element of *People*. A variable such as *BP* represents a multiset (or bag) of *People*; since it is tagged by the `unique` keyword, it can actually only be bound to a subset of *People* (each element in *BP* appears at most once). Variable *BG* is not tagged with `unique` keyword; it could be bound to any multiset of gifts (if the warehouse was configured to contain several instances of each gift for instance).

Transition guards can be used to constrain the cardinality of a Bag variable : the constraint $[card(BP) < 3 \text{ and } card(BP) > 0]$ on **airlock** model its capacity of at most 2 people (the airlock does not operate empty), while the constraint on **shopping** bounds the number of gifts bought in the store by each person.

**Equivalence with Symmetric Nets** SNB have the same expressiveness as Symmetric Nets but allow for more compact modeling. Like colored nets which can be seen as an abbreviation of P/T nets, SNB can also be unfolded into an equivalent SN. Figure 2 shows such an unfolding. Transitions **airlock** and **shopping** are replicated for each possible cardinality of the bag variable they can instantiate. Place **out** in the SNB is unfolded according to the various domains obtained by “flattening” the bag token of the *PeopleBagGift* type. Hence, the greater the bound on the cardinality, the more places in the unfolded SN. Note that modeling anything the customers do once they are "out" with their gifts would be very cumbersome in the SN.

However, designers must be careful when defining guards and initial markings so that the model remains finite if unfolded to SN. For example, if **shopping** had no **warehouse** place in input and no guard, an infinite number of bags could be generated (leading to an infinite unfolding). The main advice for the designers is to always bound the cardinality of the bag-variables.



```
Class
  People is 1..P;
  Gift is 1..G;
Domain
  PeopleGift is <People, Gift>;
  PeopleGiftGift is <People, Gift, Gift>;
Var
  p1,p2 in People;
  g1,g2 in Gift;
```

**Fig. 2.** Unfolding of the SNB presented in Figure 1 into a SN

**Advantages of SNB** From Figures 1 and 2, it is obvious that with bags manipulation, SNB provide a much more compact and natural way to model system than SN.

Another advantage of SNB is to allow production of a more compact quotient reachability graph in number of edges. This is due to the use of bag variables which better express the symmetries of possible bindings of variables to values. For instance, when choosing two *People* from **waiting**, **airlock** in the SNB allows $P \times (P-1)$ bindings (*i.e.* choose two from *P*), while **airlock** in the SN allows $2 \times P \times (P-1)$ because variables $p1$ and $p2$ can independently be bound to any element of *People*. Since computing successors in a quotient graph is a costly operation (due to the canonization procedure), this aspect may heavily impact the performance of analysis tools.

**Issues in Representing the State Space** Implementation of symmetry-based techniques is not a challenge anymore since tools such as GreatSPN [6] or MurPhi [11] have efficiently implemented such algorithms for over 20 years using explicit data structures.

The challenge resides in combining so-called “symbolic” techniques based on symmetries with the so-called “symbolic” techniques based on decision diagrams. Such a “symbolic/symbolic” approach was first experimented in [13] on top of Data Decision Diagrams (DDD) [4]. Here, the hierarchical structure of SNB for both the net structure, the types and the tokens strongly suggests to take benefits of a new decision diagram structure: Hierarchical Set Decision Diagrams (SDD) [5]. The engine developed to generate the state graph is thus fully based on decision diagrams; note that this differs from

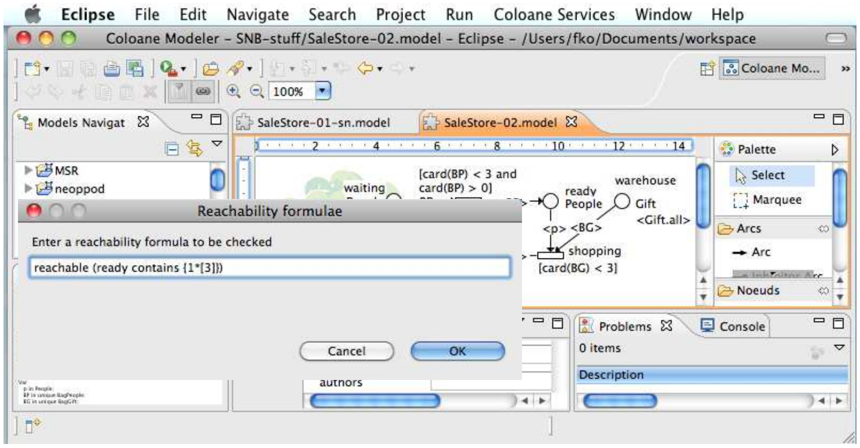
[1] which does not support construction of a quotient graph, but tries to deal with performance evaluation of stochastic symmetric nets using decision diagrams.

Our work aims at showing that the two techniques can be combined and provide, in favorable conditions, added gains.

## 3 Tool Architecture

So far, Crocodile is a "symbolic/symbolic" state space generator for SNB able to compute reachability properties. As already mentioned, its purpose is to provide a first modeling and analysis tool for SNB as well as to merge two well known techniques for efficient state space generation. This section sketches the tool architecture first, and then presents the encoding technique we use for an efficient storage of quotient state graphs.

**Use of the Tool** Crocodile is plugged in the Coloane modeler [9] (see Figure 3). It is written in C++ and uses the `libddd` [10] for SDD manipulation.



Reachability formulae

Enter a reachability formula to be checked

reachable (ready contains {1*[3]})

**Fig. 3.** Crocodile in the Coloane User Interface

Coloane is a generic graph editor in which the concrete syntax of SNB has been plugged. Once the model is designed, it is possible to invoke Crocodile directly from the "Coloane Services" menu. Then, a windows requesting for a reachability formula pops out. If no formula is provided, Crocodile simply generates the state space. We use the syntax proposed for the model checking contest [12] that has been extended to support SNB markings. Various statistics can be displayed: number of symbolic states in the quotient state graph, number of SDD nodes, the number of canonizations that have been computed, etc. Assessment and performances (section 4) are computed using a standalone version of the tool under Unix (also distributed).

**Symbolic/Symbolic Representation of SNB states** Let us first remind the main characteristics of symbolic markings as they were presented in [3]. The main idea is to avoid representing similar behaviors, *i.e.,* identical behaviors with respect to values permutations. To do so, the actual "identity" of values is forgotten and only their distributions among places are stored. Values with the same distribution and belonging to the same color type are grouped into a so-called *dynamic subclass*. A symbolic marking is, then, a cartesian product of dynamic subclasses and will represent a large number of concrete markings (according to the cardinalities of the involved subclasses).

The originality of Crocodile is to encode these symbolic markings by means of Hierarchical decision diagrams. SDD extend DDD by proposing a way to hierarchically encode data. They also inherit the notion of homomorphism that was defined in DDD [4]. Both computation of successor states and their canonization are implemented as homomorphisms.

**Encoding** Roughly, where BDD represent sets of boolean assignments, SDD represent sets of set assignments. The first consequence is that SDD arcs are valued with sets, where BDD arcs are valued with booleans. The second consequence is the hierarchy: since arcs are valued with sets, and SDD themselves represent sets, the arcs of a SDD may refer to another SDD. This increases the sharing capacity of decision diagrams and highlights their hierarchical feature.

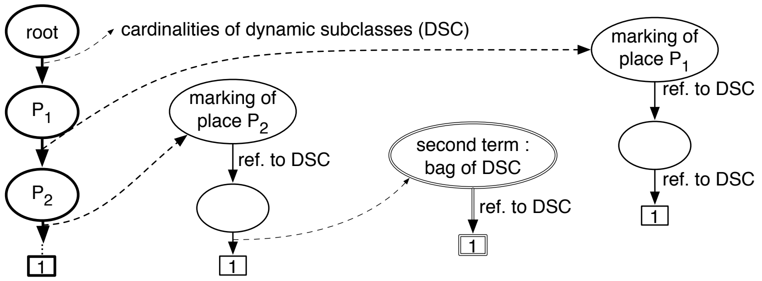


**Fig. 4.** Architecture of Crocodile encoding of a symbolic marking

Let us illustrate our encoding scheme with Figure 4. A symbolic marking is represented by:

- the identification of dynamic subclasses cardinalities,
- the symbolic content of each place (as a cartesian product of dynamic subclasses).

Our encoding presents three levels. The first one (bold) corresponds to the structure of the SNB and lists its places. The second level (thin) is reached from the arcs between the nodes encoding places and describe their symbolic marking. The third level (double thin) stands to encode bag tokens. As both second and third levels represent bags, they may share a given description (its interpretation is then handled by the homomorphism that operate on the structure). Figure 5 shows an example of this capability for two partial markings (only places **waiting**, **warehouse** and **out** are represented) of the SaleStore example:

$$M_1 = waiting(P_1) + warehouse(G_0) + out(\langle P_0, \{G_1\}\rangle)$$
$$M_2 = waiting(P_1) + warehouse(G_1) + out(\langle P_0, \{G_0\}\rangle)$$

We assume that $P_0$ and $P_1$ are dynamic subclasses in *People* while $G_0$, $G_1$ are dynamic subclasses in *Gift*.

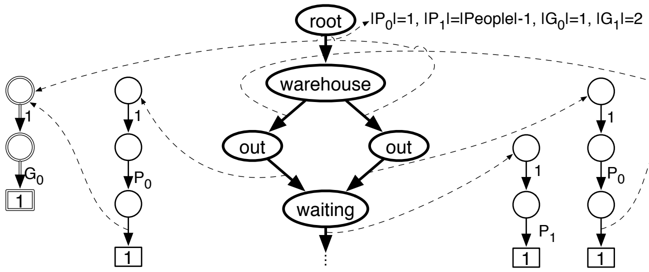


**Fig. 5.** Example of encoding for two markings with common shared parts

This Figure shows the encoding of these two partial markings. The three levels of hierarchy are clearly visible. We also observe that the markings $G_0$ (and $G_1$) are shared at two different levels. The dotted path from the arcs below place **out**, corresponds to a bag inside a bag while, from place **warehouse**, they are simply a bag.

**Symbolic/Symbolic Representation of SNB arcs** In most decision diagram-based state space representations, reachability graph arcs are not explicitly stored in memory. Yet, they may be reconstituted when necessary (*i.e.* when elaborating a counter example) through the firing relation. This is also the case in Crocodile. We thus trade memory against CPU when elaborating the counter example.

**Summary** Our objective is to stack the two so-called “symbolic” mechanisms.

First, symbolic states allow a compact representation by grouping similar states up to permutations thanks to dynamic subclasses that gather structurally equivalent values in color types.

Second, symbolic encoding of such a state representation allows to share common parts of the description, thus saving memory and providing a fast way to compare symbolic markings. The use of the SNB structure (graph, tokens, bags in tokens) even increases the sharing capacity between levels of representation with SDD.

## 4 Assessment and Performances

This section shows how we assessed our tool on SNB by using a comparison with GreatSPN [6] that is now the reference implementation of the quotient state graph for SN. In this case, we only use the colored features of GreatSPN that also handles stochastic nets[5]. We also run our tool on both the SN and SNB models of section 2 with various values for the two scaling parameters (size of types *People* and *Gift*).

[5] Well-Formed Petri Nets [2] introduce stochastic features that are not yet embedded in SNB.

**Assessment** since SNB encompass SN, we use GreatSPN as a comparison when processing SN with Crocodile. In both cases, the size of the state space is the same.

We also observe the same number of states for the quotient state space for both SN and SNB, which is consistent too. In fact, the main difference between the symbolic state space of a SNB and the one of its unfolded SN is the number of symbolic arcs. This will lead to the analysis of less successors for symbolic states and thus, less canonizations (as illustrated in Table 1 and Figure 6).

**Performance** To evaluate performance of state space generation (to first verify safety properties), we use the models presented in section 2. The SNB of Figure 1 was processed by Crocodile while its corresponding SN was processed by both Crocodile and GreatSPN. This enables a separate evaluation of the gain brought by the encoding compared to the one coming from the use of bags in tokens.

We let the number of values in *People* and *Gift* increase progressively. Executions were operated on a 32bits 3.2GHz Intel processor with 3GByte of memory and running Linux. Time was measured with `time` and memory estimated using `memusage`. Table 1 summarizes the collected information. Columns show:

- P, the size of class *People*,
- G, the size of class *Gift*,
- the size (number of nodes) for both the quotient state graph and the state space (*i.e.* the corresponding concrete state space),
- for SNB (with Crocodile) and SN (with both Crocodile and GreatSPN): the number of firings[6] performed to build the quotient state graph, consumed memory[7] and execution time in seconds.

Table 1 clearly shows (once again) the benefits brought by symbolic techniques compared to explicit ones, when models exhibit symmetries.

It also shows the very low number of firings Crocodile needs to build the SNB quotient state graph compared to GreatSPN for the SN one. Crocodile also explore more successors for SN than for SNB, which is related to the reduced number of transitions in the SNB. Both tools must canonize each new discovered state to check if it belongs to an already computed one. Since the canonization algorithm is time consuming, this has a dramatic impact on GreatSPN execution. This impact can be noticed in Figure 6(a) that shows execution time for $|People| = 6$ and $|Gift|$ varying from 2 to 15.

The benefits of decision diagram based representation is also highlighted when comparing GreatSNPN and Crocodile running on SN. For small values of P and G, shared parts of the quotient state graph are not sufficient to overload the initial cost of the decision diagram structure, this becomes false when G $\geq$ 8 in Table 1, thus leading to consequent gain in memory usage. This is also visible in Figure 6(b) that shows the evolution of memory consumption for $|People| = 6$ and $|Gift|$ varying from 2 to 15.

When G $>$ 9, the combinatorial explosion of firings forces GreatSPN to stop. Thus, we can process the example for large values such as 20 peoples with 40 gifts (last line in the table, asymptote point in the quotient state space for this system configuration).

[6] They are symbolic/symbolic firings for Crocodile and symbolic firing for GreatSPN.

[7] MOVF means memory overflow (around 2.03 Gbytes on our experiment machine).

| | | Number of Nodes | | SNB (Crocodile) | | | SN (Crocodile) | | | SN (GreatSPN) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P | G | Quotient Graph | Ordinary Space | # of Firings | Mem. in KB | Time in seconds | # of Firings | Mem. in KB | Time in seconds | # of Firings | Mem. in KB | Time in seconds |
| 5 | 6 | 116 | $6.70 \times 10^{05}$ | 285 | 502 | 0.4 | 361 | 703 | 0.7 | 10 430 | 407 | 17.0 |
| 5 | 7 | 120 | $2.75 \times 10^{06}$ | 296 | 602 | 0.4 | 377 | 745 | 0.7 | 60 850 | 433 | 29.7 |
| 5 | 8 | 124 | $1.10 \times 10^{07}$ | 303 | 684 | 0.4 | 388 | 781 | 0.8 | 99 920 | 71 958 | 2 041.1 |
| 5 | 9 | 125 | $4.18 \times 10^{07}$ | 305 | 653 | 0.4 | 392 | 784 | 0.9 | 3 497 661 | 77 628 | 3 128.2 |
| 5 | 10 | 126 | $1.51 \times 10^{08}$ | 306 | 572 | 0.4 | 394 | 783 | 0.9 | — | MOVF | — |
| 5 | 50 | 126 | $4.24 \times 10^{16}$ | 306 | 573 | 0.5 | 394 | 797 | 1.1 | — | MOVF | — |
| 6 | 6 | 180 | $4.12 \times 10^{06}$ | 496 | 794 | 0.7 | 652 | 875 | 1.2 | 28 612 | 422 | 39.8 |
| 6 | 7 | 190 | $1.97 \times 10^{07}$ | 527 | 976 | 0.8 | 695 | 934 | 1.3 | 146 775 | 448 | 77.7 |
| 6 | 8 | 200 | $9.24 \times 10^{07}$ | 550 | 1 206 | 0.9 | 730 | 1 087 | 1.6 | 289 301 | 73 855 | 5 857.8 |
| 6 | 9 | 204 | $4.22 \times 10^{08}$ | 561 | 1 277 | 0.9 | 745 | 1 132 | 1.7 | 10 589 107 | 79 526 | 10 564.5 |
| 6 | 10 | 208 | $1.86 \times 10^{09}$ | 568 | 1 357 | 1.0 | 756 | 1 183 | 1.8 | — | MOVF | — |
| 6 | 11 | 209 | $7.78 \times 10^{09}$ | 570 | 1 284 | 0.9 | 760 | 1 184 | 1.8 | — | MOVF | — |
| 6 | 12 | 210 | $3.07 \times 10^{10}$ | 571 | 1 182 | 0.9 | 762 | 1 227 | 1.9 | — | MOVF | — |
| 6 | 50 | 210 | $1.03 \times 10^{19}$ | 571 | 1 002 | 1.1 | 762 | 1 245 | 2.1 | — | MOVF | — |
| 7 | 6 | 260 | $2.29 \times 10^{07}$ | 744 | 1 275 | 1.2 | 967 | 1 242 | 2.0 | 64 531 | 453 | 83.5 |
| 7 | 7 | 280 | $1.24 \times 10^{08}$ | 811 | 1 578 | 1.5 | 1 052 | 1 408 | 2.4 | 304 504 | 880 | 196.6 |
| 7 | 8 | 300 | $6.65 \times 10^{08}$ | 864 | 1 909 | 1.8 | 1 127 | 1 615 | 2.9 | 680 594 | 75 753 | 12 024.3 |
| 7 | 9 | 310 | $3.19 \times 10^{09}$ | 896 | 2 135 | 2.0 | 1 168 | 1 734 | 3.2 | 22 553 532 | 81 424 | 23 548.3 |
| 7 | 10 | 320 | $1.81 \times 10^{10}$ | 919 | 2 151 | 2.1 | 1 200 | 1 985 | 3.5 | — | MOVF | — |
| 7 | 14 | 330 | $8.34 \times 10^{12}$ | 940 | 1 981 | 2.0 | 1 232 | 2 132 | 4.0 | — | MOVF | — |
| 7 | 100 | 330 | $4.18 \times 10^{27}$ | 940 | 1 811 | 2.7 | 1 232 | 2 172 | 4.9 | — | MOVF | — |
| 8 | 6 | 356 | $1.19 \times 10^{08}$ | 1 084 | 1 301 | 1.8 | 1448 | 1 611 | 3.04 | 123 639 | 486 | 151.86 |
| 8 | 7 | 390 | $7.11 \times 10^{08}$ | 1 208 | 1 967 | 2.33 | 1606 | 1 861 | 3.73 | 587 084 | 948 | 495.11 |
| 8 | 8 | 425 | $4.27 \times 10^{09}$ | 1 312 | 2 600 | 3.02 | 1754 | 2 240 | 4.63 | 1 496 928 | 77 654 | 21300.24 |
| 8 | 9 | 445 | $2.54 \times 10^{10}$ | 1 382 | 3 134 | 3.47 | 1845 | 2 379 | 5.2 | 40 063 379 | 83 326 | 43946.47 |
| 8 | 10 | 465 | $1.49 \times 10^{11}$ | 1 436 | 3 496 | 3.93 | 1924 | 2 531 | 5.84 | — | MOVF | — |
| 8 | 16 | 495 | $2.90 \times 10^{15}$ | 1 511 | 3 568 | 4.02 | 2032 | 2 653 | 6.97 | — | MOVF | — |
| 8 | 100 | 495 | $3.10 \times 10^{31}$ | 1 511 | 3 255 | 5.04 | 2032 | 2 742 | 8.34 | — | MOVF | — |
| 20 | 40 | 10 626 | $3.17 \times 10^{51}$ | 41 529 | 1 401 621 | 6016.79 | 57051 | 1 150 338 | 5422.91 | — | MOVF | — |

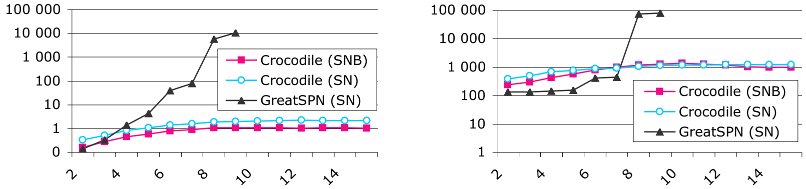
**Table 1.** Compared performances of Crocodile and GreatSPN on state space generation.

Let us notice two points for this model. First, the combinatorial explosion dramatically increases every two increments of G. This is due to the maximum bound of gifts to be bought (see guard in transition **shopping**, that bound this value to 2). This increases can be directly observed in the charts of Figure 6. Second, the number of symbolic markings stabilizes when $|Gift| \geq 2 \times |People|$. When $|Gift|$ is just below the stabilization value ($2 \times |People|$), the SDD structure is not fully dense; beyond this value, the sharing in the SDD structure is maximized. Reaching this point of maximal sharing results in a slight decrease of memory consumption for Crocodile.

**State Space Snalysis** So far, Crocodile provides analysis of reachability properties. Such properties are constraints that can be checked during state space generation. This does not bring extra complexity (just a constant due to the property evaluation). Evaluation of a reachability property is done using the following schema:

- translation of the property into constraints $c$ on the symbolic markings (expressed as a SDD),
- for each new symbolic state $s$, comparing the canonical representation of $s$ with $c$ (since both are SDD, this is a fast operation).

So far, once a state verifying the property is found, the tool must reexecute the state space generation algorithm to store the list of symbolic firings leading to the identified



(a) Computation time in seconds

(b) Memory consumption in Kbyte

**Fig. 6.** Memory and time measures for $|People| = 6$ and $|Gift|$ varying from 2 to 15

state. Thus, verification of a reachability property may lead to building twice the state space in the worst case. This complexity is compensated by the gain in the state space generation.

**Summary** As a conclusion to these experiments, we note the two so-called "symbolic" techniques (the one based on symmetries and the one based on decision diagram encoding) stack well. First, the traditional quotient state graph brings an exponential gain with respect to the ordinary graph. Then, the SDD based encoding brings another exponential gain on top of the previous one. As Figure 6 shows, most of the gains observed on SN are brought by the simultaneous use of these techniques.

We have another confirmation that coupling the two symbolic techniques is of interest. A prototype version of GreatSPN uses several variants of decision diagrams [1]: multi-way DD (MDD), multi-terminal MDD (MTMDD), and edge-valued MDD (EV+ MDD). None of these are hierarchical and they encode Stochastic P/T nets so far. Their results also show significant gain from the original version.

## 5 Conclusion

This paper presents the tool Crocodile that is original in several manners: *(i)* it is the first implementation of SNB [7] and *(ii)* it encodes the quotient state graph with decision diagrams (symbolic/symbolic approach).

From this work, we can draw three main results. First, SNB show good modeling compacity when manipulating sets or bags in Petri nets. This is illustrated by our small example: the SaleStore model and its unfolding to SN.

Second, as already foreseen in [1], the two so-called "symbolic" techniques (symmetry-based and decision diagram-based) stack very well. Each brings an exponential reduction factor in performances, as shown in section 4. In particular, the hierarchical encoding of markings in the quotient state graph even increases the sharing capacity, thus leading to significant gains in memory.

Third, the theoretical gain in the number of arcs for SNB is experimentally demonstrated. It increases performances already brought by the encoding technique since it simplifies the computation of the quotient state graph (less successors to examine). Moreover, since Crocodile relies on decision-diagrams, we do not explicitly represent arcs, thus increasing memory gain.

Crocodile is available at `http://move.lip6.fr/software/SNB/`. It shows good performances in both memory consumption and execution time. It is able to perform analysis of reachability properties. However, computation of a counter example might be optimized in the future. So far, it is based on a second computation of the state space.

It would also be of interest to formally define the unfolding from SNB to SN and its reverse operation to be able to enable on-the-fly use of the techniques embedded in Crocodile. We could then cumulate benefits of the SNB model with more classical modeling schemes.

## References

1. J. Babar, M. Beccuti, S. Donatelli, and A. Miner. GreatSPN Enhanced with Decision Diagram Data Structures. In *31st International Conference on Petri Nets and Other Models of Concurrency (ICATPN 2010)*, volume 6128 of *Lecture Notes in Computer Science*, pages 308–317, Braga, Portugal, June 2010. Springer.
2. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On well-formed coloured nets and their symbolic reachability graph. In K. Jensen and G. Rozenberg, editors, *Procedings of the 11th International Conference on Application and Theory of Petri Nets (ICATPN'90). Reprinted in High-Level Petri Nets, Theory and Application.* Springer-Verlag, 1991.
3. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. A symbolic reachability graph for coloured Petri nets. *Theoretical Computer Science*, 176(1–2):39–65, 1997.
4. J.-M. Couvreur, E. Encrenaz, E. Paviot-Adet, D. Poitrenaud, and P.-A. Wacrenier. Data Decision Diagrams for Petri Net Analysis. In *ICATPN'02*, volume 2360 of *LNCS*, pages 1–101. Springer-Verlag, 2002.
5. J.-M. Couvreur and Y. Thierry-Mieg. Hierarchical decision diagrams to exploit model structure. In *25th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems*, volume 3731 of *LNCS*, pages 443–457. Springer, 2005.
6. GreatSPN. Petri nets suite: `http://www.di.unito.it/~greatspn`.
7. S. Haddad, F. Kordon, L. Petrucci, J.-F. Pradat-Peyre, and N. Trèves. Efficient State-Based Analysis by Introducing Bags in Petri Net Color Domains. In *28th American Control Conference (ACC'09)*, pages 5018–5025, St-Louis, USA, June 2009. Omnipress IEEE.
8. T. Junttila. *On the symmetry reduction method for Petri Nets and similar formalisms*. PhD thesis, Helsinki University of Technology, Espoo, Finland, 2003.
9. MoVe team. The coloane home page : `http://move.lip6.fr/software/COLOANE`.
10. MoVe team. The `libddd` home page : `http://move.lip6.fr/software/DDD`.
11. Murphi. Murphi description language and verifier: `http://verify.stanford.edu/dill/murphi.html`.
12. SUMO'2011. Sumo model checking contest: `http://sumo.lip6.fr/Model_Checking_Contest.html`.
13. Y. Thierry-Mieg, J.-M. Ilié, and D. Poitrenaud. A symbolic symbolic state space representation. In D. de Frutos-Escrig and M. Núñez, editors, *FORTE*, volume 3235 of *LNCS*, pages 276–291. Springer Verlag, 2004.