An Evaluation of Metro Express

Shahram Esmaeilsabzali Leonard Wong Nancy A. Day

David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
{sesmaeil,lkhwong,nday}@cs.uwaterloo.ca

Technical Report: CS-2005-20
December 2005

Abstract. We study *Express*, a system that uses template semantics to map specifications to SMV models. We investigate the efficiency of the generated SMV models. We consider two case studies and compare manually created SMV models with models generated by Express. The generated models are more complex and have larger state spaces, and consequently longer verification times. We also analyze the effect of using different optimization schemes in Cadence SMV. Interestingly, some of the optimizations are more applicable to the generated models and made verification of the generated models outperform the manually created models for some queries. However, considering the best case for each model, the manually created model always outperforms the generated model. We propose some optimizations that could be used to improve dramatically the efficiency of the models generated by Express. We estimate the magnitude of improvements that such optimizations can provide.

1 Introduction

The purpose of this report is to investigate the efficiency of SMV models generated automatically by *Express* [Lu04]. Express is a tool that maps a specification written for the *Metro* tool suite [NAD03] to its equivalent SMV model. Metro is based on *template semantics*, a method for describing the semantics of a rich set of model-based specification notations. As such, Express is capable of understanding different notations and semantics, and mapping them into SMV models. SMV [McM93] is a symbolic model checker.

We analyzed the verification performance of the models generated by Express by comparing them against manually created SMV models. We considered the same two case studies as in [Lu04]: a heating system and a single-lane bridge. By inspecting the generated model, we observed that the model is not as readable as a manually created model. This can be considered inherent to any automatically generated artifact. Other complexity criteria, such as the size of model description, also confirmed that the generated model is significantly more complicated

than the manually created one. However, after getting used to the underlying logic of the generated model, and also template semantics, the generated model became more understandable.

We used the same set of properties to verify both the manually created and generated models. The verification time for the generated models were longer, as expected. Also, the size of the state space was substantially greater in the generated model, particularly for the heating system example.

We tried different SMV optimization schemes and observed that the default optimizations of Cadence SMV provide the best verification time for both the manually created and the generated models. We also measured the number of BDD nodes under different optimization schemes. Surprisingly, the generated model, when using some optimization schemes, used fewer BDD nodes than the manually created model. These findings can be further studied and analyzed to make more detailed conclusions about the nature of models.

The consistency of our experimental results for the two examples is encouraging. Consistency is important because the two examples are quite different and use different sets of composition operators.

While analyzing Express-generated models, we identified four optimization opportunities for making the generated model more efficient. All of our proposed optimizations aim at decreasing the number of variables in the system; and thus the size of the state space and, hopefully, the verification time. Decreasing the number of variables could potentially provide *exponential* savings in the size of the state space. We show that how those optimizations can be applied to one of the studied examples.

The rest of the report is organized as follows: In Section 2, we introduce our case studies and the properties that we verified. In Section 3, we discuss different experiments that we performed on models and present the results. In Section 4, we present our proposed optimization methods for Express. Finally, Section 5 presents conclusions of this report. We assume the reader has some familiarity with template semantics.

2 Case Studies

We considered two systems, the *Heating System* and the *Single-Lane Bridge*, for our analysis of Express. These examples were taken from Yun Lu's thesis [Lu04], where the SMV models are automatically generated from the specifications. We found these two systems interesting since they use notations with different semantics and, furthermore, they are specified using different sets of composition operators.

2.1 Heating System

The heating system controls the temperature of a house. It is specified in STATEMATE [HN96] semantics and consists of three major *hierarchical transition sys-*

tems (HTSs)[1]: `room`, `controller`, and `furnace`. These HTSs are composed using *parallel* compositions, meaning that all of them are active at any given time. The `room` HTS further consists of two other HTSs, namely, `noHeatReq` and `heatReq`, which are composed using *interrupt* composition. Interrupt composition means that exactly one of the components is active at any given time, and there are transitions that transfer control from one component to the other.

Manually Created vs. Generated SMV Model: To evaluate Express-generated SMV models, we manually created an SMV model that follows the structure of the specification as closely as possible. For instance, the interrupt composition for two HTSs, namely, `noHeatReq` and `heatReq`, is specified by introducing variables that model the occurrence of an interrupt between two HTSs. We could have instead flattened the two HTSs into one, and avoided explicitly modelling interrupts. In the heating system case study, we deliberately tried to do our translation from the specification to the SMV model as mechanically as possible. Such a mechanical translation, in our opinion, provides a fair ground for comparison between the automatically and manually created models.

While developing the model manually, an unintentional non-deterministic behavior was discovered in the specification: When the controller's active state is `controllerOn`, and the environment generates events `heatSwitchOff` and `furnaceFault` simultaneously, the next state of the controller can be either `off` or `error`. Both the generated and the manually created models determine the next state non-deterministically. However, it is not obvious whether this behavior is desired in the original specification.

Table 1 presents different comparison criteria for the generated and manually created models. The criteria are meant to provide a rough estimate of the complexity of each model. In particular, non-blank lines of the model, number of variables, and number of macro definitions, i.e., `DEFINE` statements, indicate how difficult it is for users to understand or trace through each model. This is a crucial factor when users are debugging the model or properties. As showed in Table 1, the manually created model had significantly more desirable values in all criteria. This implies that it is likely that the manually created model is more easily understood by users. This is not unexpected since the generated model is created based on some general model generation policies while the manually created model simply models a particular specification. Also, it may be possible to create a tool flow in which the generated model is never studied by the user, but rather counterexamples are illustrated on the source specification.

Lesson Learned (SMV Problem): During the creation of the heating system SMV model, a problem was discovered that caused SMV to return true for all properties, including explicit falsehood, in CTL terms `(EF 0)`. Basically, when

[1] "An HTS is a hierarchical, non-concurrent, extended state machine. In statecharts terminology, an HTS supports OR-state hierarchy but not AND-state hierarchy." [NAD03]

Measurement	Generated Model	Manually Created Model
SMV File Size	26.7 KB	10.3 KB
Non-blank Lines of the Model	804	292
Number of Variables	32	24
Number of Macro Definitions	237	77

Table 1. Comparing Generated vs. Manually Created SMV Models for the Heating System

the search space, constrained by the fairness statements, is empty, Cadence SMV passes all properties including explicitly false statements. The snippet of a model in Figure 1 shows a situation where if $bug = 1$ then all properties pass. The problem is that both of the fairness constraints can never be true in this model.

```
MODULE main()

VAR temp : {HOT, COLD};

DEFINE bug:= 1; -- Bug presence flag
DEFINE isStable := bug ? 0 : {0,1}; -- temp can change when isStable = 1

ASSIGN next(temp) := isStable ? {HOT, COLD} : temp;

SPEC (EF 0)
SPEC ((AG ~(temp=COLD)) & (EF (temp=COLD)))

FAIRNESS (temp = COLD)
FAIRNESS (temp = HOT)
```

Fig. 1: SMV Model where "False" Properties Pass

Properties: Four properties, as specified in Yun Lu's thesis [Lu04], were considered for verification. They are non-trivial properties that use various `CTL` operators such as `AG`, `->`, `AX`, `AF`, and `AU`. The properties were first encoded in the manually created model, and then were adapted to the generated model by simply renaming the variables and macro names appropriately. The following are the list of properties.

1. If the actual temperature is too low and stays too low after a timeout, then the furnace will be turned on, unless an error occurred.

```
AG ((furn.in_furnaceNormal & TOOCOLD
     & (room.waitedForWarm = WARMUPTIMER)) ->
  AX (A[~furn.in_furnaceOn U
     (furn.in_furnaceOn | furn.in_furnaceErr)] ));
```

2. If the actual temperature is too high and stays too high after a timeout, then the furnace will be turned off, unless an error occurred.

```
AG ((furn.in_furnaceNormal & TOOHOT &
     (room.waitedForCool = COOLDOWNTIMER)) ->
  AX ( A[~furn.in_furnaceOff U
     (furn.in_furnaceOff | furn.in_furnaceErr)]));
```

3. Whenever the furnace fails, it will not start before the user resets it.

```
AG ((envr.furnaceFault & ~envr.userReset) ->
  (AX A[(~furn.in_furnaceAct & ~furn.in_furnaceRun)
       U envr.userReset]));
```

4. If the room does not request heat, then eventually the furnace will be turned off. Also, if the room requests heat, then the furnace will be turned on unless there is an error.

```
AG ( (room.in_noHeatReq -> AF ~furn.in_furnaceOn)
   & (room.in_heatReq ->
        (AX A[~furn.in_furnaceOn U
           (furn.in_furnaceOn | furn.in_furnaceErr)])));
```

2.2 Single-Lane Bridge

The second system we considered for our study was a single-lane bridge, which is a system for controlling the traffic over a single-lane bridge. The specification mainly consists of some interleaved components. There are two red and two blue cars on different sides of the bridge. The bridge is controlled such that red cars and blue cars do not collide. The specification is in CCS [Mil95] with shared variables. Each car is specified as a separate component and cars are interleaved by using three interleaving composition operators. Red and blue cars have separate `enter` and `exit` coordinator entities, i.e., four coordinators in total. The coordinators are also interleaved by three interleaving composition operators. The car and coordinator components communicate through *environmental synchronization composition* events.[2] There is also a `bridgeStatus` component that represents the current status of the bridge, i.e., the number of red or blue cars

[2] Environmental synchronization requires that "both components execute in the same micro-step if the executing transitions all have the same trigger event that is a designated synchronization event; otherwise, one or the other component takes a step in isolation, executing transitions not triggered by synchronization events." [Lu04]

on the bridge. It communicates with car components through rendezvous composition.

In our second case study, we deviated from the mechanical manual translation and allowed the SMV modeller more freedom in his translation. The hand-generated SMV model for the single-lane bridge did not follow exactly the mechanical translation of states and transitions of the specification. This approach means that the decomposition and transitions of the two models are not exactly the same but yet they specify the same functionalities and both satisfy the same set of properties.

The fact that the specification heavily relied on interleaving processes made it a suitable system to be modelled in the SPIN model checker [Hol97]. Therefore, we also manually modelled the system in SPIN which provided us with some insights about verification efficiency in different tools.

Manually Created vs. Generated SMV Models: In our manually created SMV model, we simulated interleaving composition by using synchronous modules. Each module in an interleaving composition was non-deterministically given the turn to run. When one module ran, the others were idle. We avoided using the built-in SMV interleaving operator, since it did not allow us to model rendezvous and environmental synchronization compositions.[3] Table 2 shows different properties of the manually created and Express-generated SMV models. Similar to the heating system, the manually created model was significantly smaller and less complex.

Measurement	Generated Model	Manually Created Model
`SMV File Size`	64.8 KB	11.3 KB
`Non-blank Lines of the Model`	1593	324
`Number of Variables`	41	33
`Number of Macro Definitions`	757	0

Table 2. Generated SMV Model vs. Manually Created SMV Model For Single-Lane Bridge

In our manually created model, we flattened the six binary interleaving compositions into two interleaving compositions, one composition for the cars and one for the coordinator components. Similarly, we modified the generated SMV model to use only two interleaving compositions. We were hoping to improve the performance of the generated model since we removed some `INVAR` statements in the model. However, it turned out that this made the performance of the generated model worse. The verification time increased from 0.08 seconds to 0.24 seconds. While the size of state space remained the same for both models, the

[3] In SMV, modules can be made to execute asynchronously by using the `process` keyword.

number of BDD nodes was larger in the modified generated model, 32128 compared to 13456. We suspect that if we had used binary interleaving operations in our manually created SMV model, we might have had better performance.

Modelling In SPIN: SPIN is inherently an interleaving verification system. It also has built-in support for rendezvous composition. However, modelling environmental synchronization turned out to be a difficult task. The problem arose from the fact that environmental synchronization requires the two components to take one simultaneous step when an environmental event happens. SPIN, on the other hand, is a purely asynchronous system. We simulated this operation by using *rendezvous channels* along with *atomic* operations. One component always received the environmental event and then notified the second component, if it was ready, and otherwise, it just consumed that event and got ready for the next environmental event. This approach was not equivalent to environmental synchronization in template semantics, since first, only one of the component always received the event and notified the other component, and second, no matter how we simulated this composition we were not able to make the two components to take a *real* simultaneous step. This is as close as we managed to get to the semantics of environmental synchronization in SPIN. As an alternative we could have tried to broadcast the environmental event to all interested components. This approach would then have required multiple copies of the same event; also, all interested components should have been ready to receive that event at all times.

Table 3 compares the SPIN model and the Express-generated SMV model of the single-lane bridge. The SPIN model is much smaller than the generated SMV model. However, the presence of channels in SPIN introduced a level of complexity that is not present in SMV model.

Measurement	Generated SMV	SPIN
`SMV Reachable States`	3.0E+08	N/A
`SPIN Matched States`	N/A	2.0E+06
`File Size`	64.8 KB	5.8 KB
`Non-blank Lines of the Model`	1593	273
`Number of Variables`	41	N/A
`Number of Channels`	N/A	20

Table 3. Comparing SPIN and Express-generated SMV Model for the Single-Lane Bridge. N/A represents non-applicable items.

Properties Four properties were verified. They are all safety properties that basically state that a red and blue car cannot be on the bridge at the same time. The list of properties follows.

1. It is never the case that the number of red cars and the number of blue cars on the bridge are greater than zero at the same time.

```
AG !(numRed>0 & numBlue>0)
```

2. A red car can enter the bridge only if there is no blue car on the bridge.

```
AG ((onRedA | onRedB) -> !(onBlueA|onBlueB))
```

3. It is never the case that eventually the number of blue cars and the number red cars on the bridge are greater than zero at the same time.

```
!AF (numRed>0 & numBlue>0)
```

4. It is never the case that red car A and blue car B are eventually on the bridge at the same time.

```
!AF (onRedA & onBlueB)
```

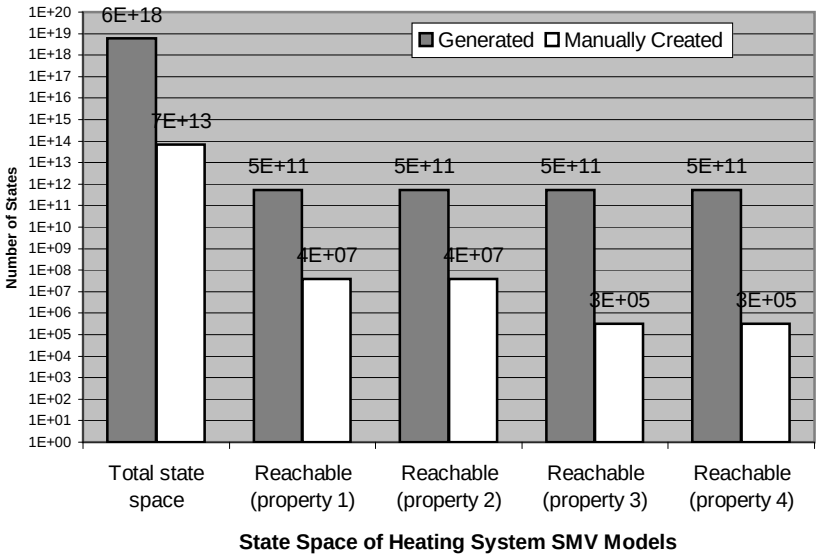
3 Evaluation

We carried out some experiments to measure the efficiency of Express-generated models against their manually created counterparts. We performed all our experiments on a Windows XP desktop, Pentium(R)IV, CPU 2.20 GHZ, with 512 MB of RAM. We used Cadence SMV (Release 10-11-02p46) for running our SMV models and used SPIN (Ver 4.2.1) for our SPIN model. Also, in our experiments, we considered the average of three experiments. Furthermore, the range of measurements were always in a stable range. In the following, we present our results.

3.1 State Space Size

Figure 2 presents the state space size of the generated and manually created models for both the heating and single-lane bridge systems. "Total state space" is the product of the number of possible values for all variables in a model. We provide the number of reachable states when verifying each property. The manually created heating system outperforms its generated counterpart on the order of 10^4. As for the single-lane bridge, the difference is not as dramatic as the heating system. We expect a more drastic out-performance by the manually created model if macros are used in our manually created SMV model.

Tables 4 and 5 provide more information about state spaces of the models. Tables 4 and 5, respectively, provide the state space information for the heating system and single-lane bridge systems, when one property is being considered in each experiment.



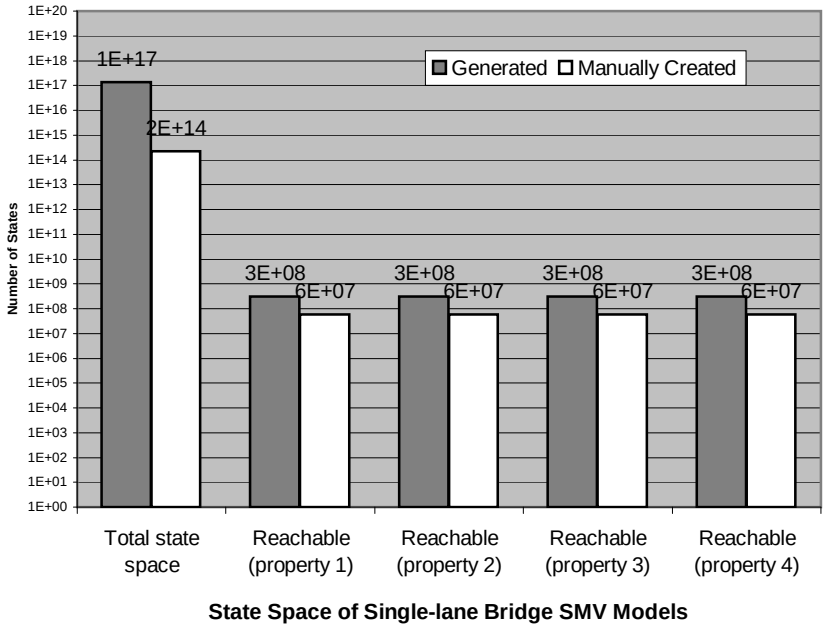


Fig. 2: State space size for the Manually Created and Express-Generated SMV Models.

Model Type	Generated Model		Manually Created Model	
Measure	BDD Variables	State Space	BDD Variables	State Space
Property 1	64	5.47E+11	51	3.77E+07
Property 2	64	5.47E+11	51	3.77E+07
Property 3	64	5.47E+11	43	311904
Property 4	64	5.47E+11	43	311904

Table 4. Number of BDD variables and reachable state space size when verifying each property of the heating system. The numbers are reported by SMV when using the default optimization scheme.

Model Type	Generated Model		Manually Created Model	
Measure	BDD Variables	State Space	BDD Variables	State Space
Property 1	51	3.02E+08	44	5.084E+07
Property 2	51	3.02E+08	44	5.084E+07
Property 3	51	3.02E+08	44	5.084E+07
Property 4	51	3.02E+08	44	5.084E+07

Table 5. Number of BDD variables and reachable state space size when verifying each property of the single-lane bridge system. The numbers are reported by SMV when using the default optimization scheme.

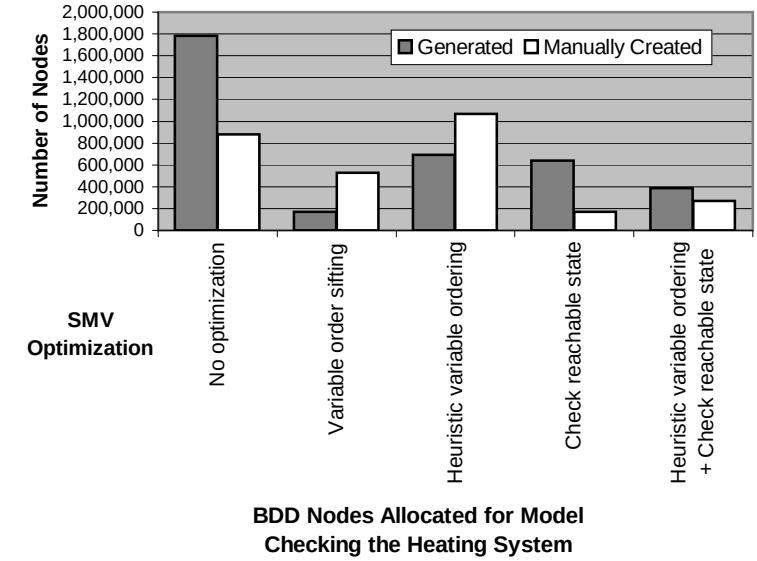
3.2 Number of BDD Nodes

The number of BDD nodes is an important factor for the efficiency of model checking. Figure 3 shows the number of BDD nodes of the manually created models with the automatically generated SMV models under different optimization schemes. The measurements are for the cases where all properties of the systems are verified together. "Check Reachable States" is an optimization in SMV, which we believe makes SMV compute the number of reachable states before actually doing the model checking. Interestingly, the generated model had a smaller number of BDD nodes when using the *variable order sifting* and *heuristic variable ordering* schemes. For all other optimization schemes, including the default, and no-optimization, the manually created model always had a smaller number of BDD nodes.

These observations are consistent for both the heating and single-lane bridge systems. Notice in the heating system chart, that the least number of BDD nodes belongs to the generated model when "variable order sifting" optimization is used.

3.3 Model Checking Time

Figure 4 shows the verification time measured when verifying all properties. As we expected, the manually created models outperformed the generated model



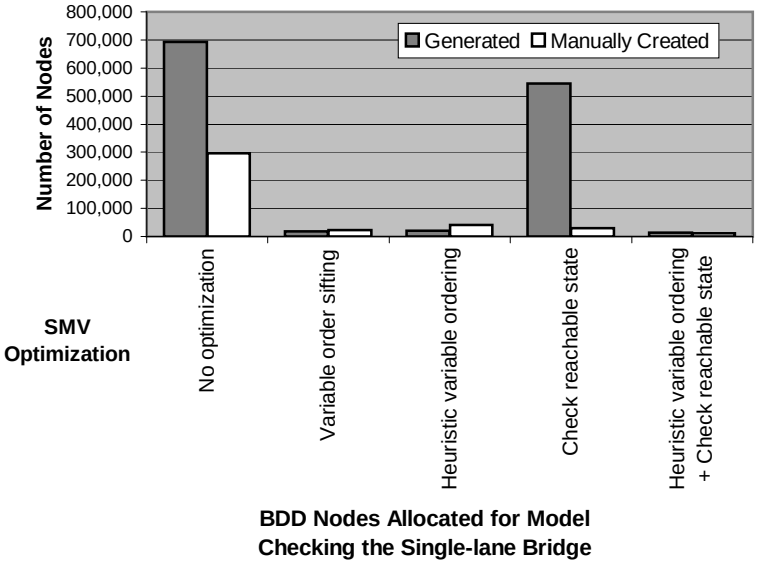


Fig. 3: Number of BDD nodes under different optimization schemes. The rightmost column represents SMV's default optimization scheme.

when both were considered at their best time. Again, we considered different Cadence SMV optimizations. The results are consistent with the BDD nodes experiments. The optimization schemes with fewer BDD nodes for the generated model had shorter verification times than their manually created counterparts. This observation is consistent in both the heating and single-lane bridge systems. The best verification times happen when the default optimization scheme, "Heuristic Variable Ordering + Check Reachable States", is used.

Based on our experiments, it seems that different optimization schemes are not more efficient than Cadence SMV's default optimization.

Another observation, worth noting, is that the manually created heating system model's best verification time is 38.8% of the Express-generated heating system model, while the ratio for the single-lane bridge is 80%. As mentioned earlier, we suspect that by using macros in the manually created model and reducing the number of variables, we can improve the manually created single-lane bridge model's performance.

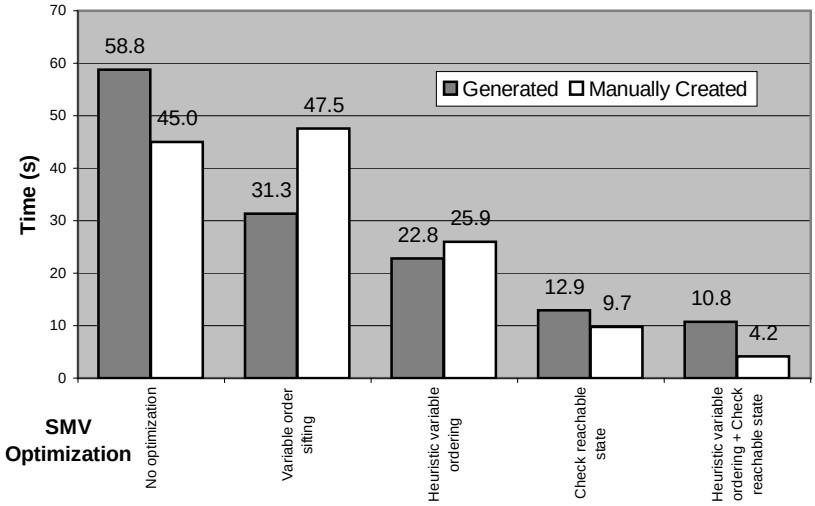
In Tables 6 and 7, verification times and number of BDD nodes for different optimization schemes are compared. (More optimization schemes are shown here than in the charts. Schemes that are worse than "no optimization" are not shown in the charts.) All experiments suggested a direct relation between the number of BDD nodes and the verification time.

Measure	BDD Nodes		Verification Time	
`Model`	`Generated`	`Manual`	`Generated`	`Manual`
No Optimization	1784256	881202	58.76	45.01
Variable Order Sifting	169416	527220	31.30	47.55
Heuristic variable ordering	691108	1065459	22.80	25.95
Sift before final order output	1784256	881202	58.78	45.24
Modified search order	1784256	881202	58.91	45.98
Check reachable state	639358	172074	12.91	9.75
Enumerate unknown values	1703993	845597	59.03	45.10
Heuristic ordering + reachable state	386719	267107	10.76	4.18

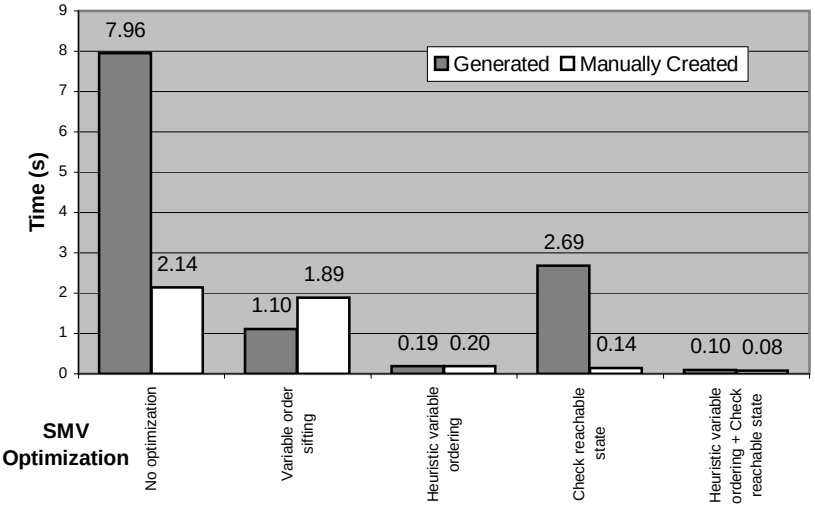
Table 6. Number of BDD nodes and the verification time for the heating system categorized by Cadence SMV optimization scheme. These numbers are measured when verifying all four properties.

SPIN does not report any model checking time for the verified model. We measured that time manually. The verification time is 24.5 seconds; which is not comparable with either SMV model. We also measured what seems to be the pure verification time, 14 seconds.[4] This is still not competitive with its SMV counterparts.

[4] SPIN creates a C file based on the SPIN specification, and after compiling that file carries out the verification. 14 seconds refers to the duration starting after the compilation finishes.



Time for Model Checking the Heating System with SMV



Time for Model Checking the Single-lane Bridge with SMV

Fig. 4: Verification times for generated and manually created SMV models. The results are consistent with the BDD nodes results in Figure 3.

Measure	BDD Nodes		Verification Time	
Model	Generated	Manual	Generated	Manual
No Optimization	692022	296830	7.96	2.14
Variable Order Sifting	19355	23734	1.10	1.89
Heuristic variable ordering	21091	40129	0.19	0.20
Sift before final order output	692022	296830	8.06	2.12
Modified search order	692022	296830	7.69	2.10
Check reachable state	544544	28664	2.69	0.14
Enumerate unknown values	692022	296830	7.66	2.10
Heuristic ordering + reachable state	13456	11816	0.10	0.08

Table 7. Number of BDD nodes and the verification time for the single-lane bridge system categorized by Cadence SMV optimization scheme. These numbers are measured when verifying all four properties.

4 Optimization Suggestions for Express

While analyzing the Express-generated SMV models, some potential optimizations were identified that can lead to more efficient generated models. Mainly, our optimizations suggest using fewer variables, by either more aggressive application of SMV macros or by removing duplicate variables, whenever applicable. Reducing the number of variables is crucial since variables increase the size of the state space. Note that macros do not introduce any extra state space while variables do.

In the following, we discuss our optimization suggestions, and the situations and variations of SMV, i.e. Cadence SMV and/or NuSMV [CCGR00], in which certain optimizations are applicable. As mentioned in [Lu04], NuSMV does not support non-deterministic assignments in macros and as such cannot enjoy some of the optimizations that we suggest.

4.1 Removing Unnecessary Environmental Events and Variables

In the generated models, every environmental event and variable is represented by two variables. In the generated heating system model, for example, the environmental event `userReset` can be found as both `pss.Ia.userReset` and `I.ev.userReset`. In the manually created heating system model, on the other hand, we avoided having two sets of duplicate variables for environmental events and variables.

One approach to remove the set of duplicate variables in the generated model, is to remove `envVars` and `envEvents`, and consequently remove the `Inputs` module. We can then merge `resetIa` and `nextIa` into the module `nextIa` and remove `resetIa` altogether. In the following snippet of a model, we show, as an example, how the new `nextIa` module would look for environmental variable `userReset`.

```
if stable
  userReset' := {0,1} -- Nondeterministic Assignment
else
  userReset' := userReset
```

This optimization is applicable to all models specified in template semantics, and can be implemented in both NuSMV and Cadence SMV. The following is an analysis of the reduction for the total state space if all the unnecessary duplicates of environmental events and variables are removed from the generated model. Consider n to be the number of variables that can be represented once, and s_i to be number of states for the ith variable, then the saving factor will be:

$$s_1 \times s_2 \times s_3 \times \ldots \times s_n$$

4.2 Using Macros to Represent Enabled Transitions

In the generated models, every HTS module has a variable to represent its enabled transitions. For instance, in module `execute_furnaceSysHts` variable `tran` had 8 possible values `t1_exe,...,t7_exe, noTran_exe`, one for each of its 7 possible transitions and one to indicate no enabled transition. By using macros to represent transition variables, the total state space size can be reduced drastically. The transition macro would be non-deterministically assigned one of the possible values and would be constrained by an invariant.

This optimization is applicable to all semantics specified in template semantics, but can be implemented only in Cadence SMV. The following is an analysis of the reduction for the total state space. Consider n to be the number of HTS modules and let t_n be the number of possible transitions for the nth module plus one, then the saving factor will be:

$$t_1 \times t_2 \times t_3 \times \ldots \times t_n$$

Since $t_n \geq 2$, the minimum saving factor will be 2^n. For the generated heating system, there are two modules with 7 transitions and one module with 10 transitions, which provides potential saving factor of $(7+1) \times (7+1) \times (10+1) = 704$.

4.3 Removing Output Variables

The outputs of an HTS can be communicated with the environment and other parallel components of the system. Express, however, keeps two copies of output variables. This optimization aims to remove redundant copies of the output variables.

Our proposed optimization, however, cannot be applied to all semantics.[5] It can only be applied to the semantics that: (1) do not distinguish between the generated events that have internal scope, called *internal-internal* events, and

[5] RSML [LHHR94] semantics, for example, cannot be considered for this optimization, but STATEMATE can be.

those generated events that are for the environment, called *external-internal* events; and, (2) do not care about the set of internal events generated in each micro-step.[6]

For all notations with semantics suitable for this optimization, we can remove the output variables of module `outputs`, and consequently, the `outputs`, `reset0`, and `next0` modules. We will then use the variables of `intEvents` instead as output variables.

This optimization can be implemented in both NuSMV and Cadence SMV. The saving factor of this optimization depends on the number of variables in the `outputs` module. For n variables in `outputs`, we will save 2^n states if all `outputs` variables are Boolean variables. In the heating system, the savings factor will be $2^4 = 16$ states.

4.4 Removing Error Variables

Each generated model has an error variable in its `variables` module, called `error_variables`, to deal with overflow of values. It is never used in the model, but could be used in properties to ensure that properties are only checked in executions of the model that do not contain overflow. This variable could be removed and would save 2 states. This optimization can be applied to all notations and can be implemented in both NuSMV and Cadence SMV.

4.5 Notes on Optimizations

Since each aforementioned optimization targets different variables, they can be applied together to the generated models. The effect of applying the proposed optimizations can be dramatic for systems, because all of the optimizations have potential saving factors that are *exponential* to the number of possible states for optimized variables.

5 Conclusions

We believe that Express is a useful tool for automatically creating a SMV model from the specification of a system. The fact that Express, by mapping Metro notations into SMV, is providing SMV mappings for different model-based notations is crucial. However, the generated model is not competitive with a manually created SMV model of the same specification.

In this report, we analyzed different properties of the generated model and, furthermore, showed how the generated model can be significantly improved by reducing some of the unnecessary variables. In particular, more aggressive use of SMV macros and eliminating some redundant variables in the model can provide exponentially smaller state spaces.

[6] In template semantics there are two types of steps: a micro-step is the execution of a single transition, and a macro-step is a sequence of zero or more micro-steps.

We also experimented with SPIN for modelling one of our case studies. It seems to us that SMV is both faster and more expressive than SPIN in modelling a larger range of models. This supports the choice of SMV as a target language for the translation of Metro models.

References

[CCGR00] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NuSMV: A new symbolic model checker. *Int. Journal on Soft. Tools for Technology Transfer*, 2(4):410–425, 2000.

[HN96] David Harel and Amnon Naamad. The statemate semantics of statecharts. *ACM Transaction on Software Engineering Methodology*, 5(4):293–333, 1996.

[Hol97] Gerard J. Holzmann. The Model Checker Spin. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997.

[LHHR94] Nancy G. Leveson, Mats Per Erik Heimdahl, Holly Hildreth, and Jon Damon Reese. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering*, 20(9):684–707, September 1994.

[Lu04] Yun Lu. Mapping Template Semantics to SMV. Master of Mathematics, School of Computer Science, University of Waterloo, August 2004.

[McM93] K. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic, 1993.

[Mil95] Robin Milner. *Communication and concurrency*. Prentice Hall International (UK) Ltd., 1995.

[NAD03] Jianwei Niu, Joanne M. Atlee, and Nancy A. Day. Template semantics for model-based notations. *IEEE Trans. Software Eng.*, 29(10):866–882, 2003.