Clause Forms Generated by Bounded Model Checking

Daniel Sheridan, Toby Walsh
Artificial Intelligence Group,
Department of Computer Science,
University of York,
York, England. YO10 5DD
{djs,tw}@cs.york.ac.uk

Introduction

Model checking is a method of verifying the behaviour of a dynamic system, comparing an implementation against a modal logic (typically, CTL) specification. A typical application is in the field of hardware verification, where a model checking tool can find bugs in hardware designs long before a product enters the testing phase.

Bounded model checking (BMC) (Biere et al., 1999) was proposed as a solution to some of the problems of conventional BDD-based symbolic model checking such as space explosion by introducing a temporal bound. The problem can then be encoded as a Boolean formula; the output of a BMC tool is a conjunction of state transition functions and state verification functions. Certain modal operators can be handled by including a check for loops in the state transitions.

The formulas produced by BMC tools may be used as the input to Boolean satisfiability (SAT) checkers, and this has influenced renewed interest in the performance of SAT procedures on highly structured problems (for example, Shtrichman (2000)).

As the specification is encoded as part of the Boolean formula, the size of the formula depends on the size of the specification, while it influences only the time complexity of BDD methods. The size of the encoding is also related to the bound size, possibly exponentially, making the correct choice of bound vital.

By analysing how CTL operations influence the number of clauses, we hope to find an indication of the difficulty of the SAT problems produced by BMC. We use the methods from Nonnengart et al. (1998) in predicting the size of the clause form conversion of a Boolean formula to determine the size of the output from a BMC tool. We present the first steps in doing this, initially for the case of non-nested operators, and show how subformula renamings can reduce the expected size of the clause form.

Encoding Model Checking

The first stage in the translation of a CTL formula is to negate it, and attempt to convert it to existential form; Biere et al. (1999) deals only with LTL, finding a counterexample at the point where the LTL formula fails to hold.

f	$[\![f]\!]_k^0$	${}_l[\![f]\!]_k^0$
$\mathbf{G}\, f_1$	false	$\bigwedge_{j=0}^k f_1(s_j)$
$\mathbf{F}\, f_1$	$\bigvee_{j=0}^k f_1(s_j)$	$\bigvee_{j=0}^k f_1(s_j)$
$\mathbf{X}\, f_1$	$f_1(s_1)$	$f_1(s_1)$
$f_1\,\mathbf{U}\, f_2$	$\bigvee_{j=0}^k (f_2(s_j) \wedge \bigwedge_{n=i}^{j-1} (f_1(s_n)))$	$\bigvee_{j=0}^k (f_2(s_j) \wedge \bigwedge_{n=i}^{j-1} (f_1(s_n)))$
$f_1\,\mathbf{R}\, f_2$	$\bigvee_{j=0}^k (f_1(s_j) \wedge \bigwedge_{n=i}^{j} (f_2(s_n)))$	$\bigwedge_{j=0}^k f_2(s_j) \vee \bigvee_{j=0}^k (f_1(s_j) \wedge \bigwedge_{n=i}^{j} (f_2(s_n)))$

Table 1: Optimised encodings for non-nested operations

We use $[\![f]\!]_i^k$ to denote the translation of a model checking problem with bound k, starting from state number i; where the path under consideration is a k-loop, starting from position l, we write ${}_l[\![f]\!]_i^k$.

The general translation of a model checking problem[1] is given by the following formula (where $\mathrm{L}_k = \bigvee_{l=0}^k {}_l\mathrm{L}_k$):

$$[\![M,f]\!]_k := [\![M]\!]_k \wedge \left(\left(\neg \mathrm{L}_k \wedge [\![f]\!]_k^0\right) \vee \bigvee_{l=0}^{k} \left({}_l\mathrm{L}_k \wedge {}_l[\![f]\!]_k^0\right) \right)$$

For the specifications consisting of one CTL operator with propositional arguments, the translations are given in Table 1.

Several features are clear: no translation depends on the value of l, so ${}_l[\![f]\!]_k^0$ may be moved outside of the disjunction, which in turn may be replaced by $\mathrm{L_k}$. Where the loop and non-loop translations are the same, the L terms are cancelled. Specifically, this occurs for LTL operators $\mathbf{F}, \mathbf{X}$ and $\mathbf{U}$ (corresponding to $\mathbf{AG}$, $\mathbf{AX}$ and $\mathbf{A}[\mathbf{R}]$ in the specification), implying that, given a choice, these operators are preferred. For $\mathbf{G}$ (corresponding to $\mathbf{AF}$), the non-loop translation is empty, and can be eliminated. Finally, exploiting the similarity between the loop and non-loop cases for $\mathbf{A}[\mathbf{U}]$ improves this case. We now see three cases for the general translation:

[1]This comes from definition 15 in Biere et al. (1999)

ϕ	$p(\phi)$	$\bar{p}(\phi)$
$\neg\phi_1$	$\bar{p}(\phi_1)$	$p(\phi_1)$
$\phi_1 \wedge \phi_2$	$p(\phi_1) + p(\phi_2)$	$\bar{p}(\phi_1)\bar{p}(\phi_2)$
$\phi_1 \vee \phi_2$	$p(\phi_1)p(\phi_2)$	$\bar{p}(\phi_1) + \bar{p}(\phi_2)$
$\phi_1 \Rightarrow \phi_2$	$\bar{p}(\phi_1)p(\phi_2)$	$p(\phi_1) + \bar{p}(\phi_2)$

Table 2: $p(\phi)$: the number of clauses produced

No non-loop part (AF) : $[\![M]\!]_k \wedge \bigvee_{l=0}^{k} {}_l\mathrm{L}_k \wedge {}_l[\![f]\!]_k^0$

Identical loop and non-loop parts ($\mathbf{AG}$, $\mathbf{AX}$ and $\mathbf{A}[\mathbf{R}]$): $[\![M]\!]_k \wedge {}_l[\![f]\!]_k^0$

Differing loop and non-loop parts ($\mathbf{A}[\mathbf{U}]$): Observing ${}_l[\![f\,\mathbf{R}\,g]\!]_k^0 = \bigwedge_{j=0}^{k} g(s_j) \vee [\![f\,\mathbf{R}\,g]\!]_k^0$, the general translation becomes $[\![M]\!]_k \wedge \left([\![f]\!]_k^0 \vee \mathrm{L}_k\right) \wedge {}_l[\![f]\!]_k^0$

Encoding Sizes

The number of clauses produced by a formula ϕ is given by $p(\phi)$, and for $\neg\phi$ by $\bar{p}(\phi)$ (Table 2).

The size of a translation does not depend on variable names, so the state parameters can be disregarded, leading to the simplified sizes for each LTL operator in Table 3.

The number of clauses produced by the general formula is

$$\left(p(\mathrm{L}_k) + p\left([\![f]\!]_k^0\right)\right) \left(\prod_{l=0}^{k} \left(p({}_l\mathrm{L}_k) + p\left({}_l[\![f]\!]_k^0\right)\right) \right)$$

This may be rewritten for each of the three cases above, taking the cancelling into consideration.

No non-loop part (AF) : $p\left([\![M]\!]_k\right) + p(\mathrm{L}_k) + p({}_l[\![f]\!]_k^0)$

Identical loop and non-loop parts ($\mathbf{AG}$, $\mathbf{AX}$ and $\mathbf{A}[\mathbf{R}]$): $p\left([\![M]\!]_k\right) + p({}_l[\![f]\!]_k^0)$

Differing loop and non-loop parts ($\mathbf{A}[\mathbf{U}]$): $p\left([\![M]\!]_k\right) + p(\mathrm{L}_k)\, p([\![f]\!]_k^0) + p({}_l[\![f]\!]_k^0)$

Renaming for $\mathbf{A}[\mathbf{U}]$

The translation for $\mathbf{A}[\mathbf{U}]$ is the most problematic, as it includes a product of two product terms ($[\![f]\!]_k^0$ and L_k), thus giving it a translation size far in excess of other operators. However, by renaming L_k and $[\![f]\!]_k^0$ using variables λ and ρ we hope to reduce the problem.

Since these two subformulæ are used only in positive forms (that is, with polarity of 1 as discussed in Nonnengart et al. (1998)) the definitions of the variables may be given by $\lambda \Rightarrow \mathrm{L}_k$ and $\rho \Rightarrow [\![f]\!]_k^0$, and so the translation is reduced to

$$[\![M]\!]_k \wedge (\lambda \vee \rho) \wedge \left(\bigwedge_{j=0}^{k} g(s_j) \vee \rho \right) \wedge (\lambda \Rightarrow \mathrm{L}_k) \wedge (\rho \Rightarrow [\![f]\!]_k^0) \quad (1)$$

Though this formula appears longer, much of the complexity in converting to clause form has been eliminated. The size of this translation is

$$p\left([\![M]\!]_k\right) + 1 + k\,p(g) + p(\mathrm{L}_k) + p([\![f]\!]_k^0),$$

which shows the elimination of the product term, and also how the size of ${}_l[\![f\,\mathrm{R}\,g]\!]_k^0$, originally a product, has been converted into a sum.

f	$p({}_l[\![f]\!]_k^i)$
$\mathbf{G}\,f_1$	$kp(f_1)$
$\mathbf{F}\,f_1$	$(p(f_1))^k$
$\mathbf{X}\,f_1$	$p(f_1)$
$f_1\,\mathbf{U}\,f_2$	$p(g)\prod_{j=1}^{k}(p(f_2) + j\,p(f_1))$
$f_1\,\mathbf{R}\,f_2$ [1]	$\prod_{j=0}^{k}(p(f_1) + j\,(p(f_2)))$
$f_1\,\mathbf{R}\,f_2$ [2]	$k\,p(g)\prod_{j=0}^{k}(p(f_1) + j\,(p(f_2)))$

[1] non-loop translation [2] loop translation

Table 3: Encoding sizes for non-nested operations

Conclusion

We have illustrated a useful technique in determining the quality of an encoding, and shown that subformula renaming in addition to careful simplification can successfully reduce the size of a clause form encoding of a model checking problem. Hopefully, by applying similar techniques to nested CTL operations, we will be able to deduce a more space efficient encoding in general. Further analysis remains to be done on the improvements to the time spent in the SAT checker (if any) to be made by performing renamings of this type.

Acknowledgements

My gratitude goes to Alan Frisch, for proof reading and checking this paper.

References

Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In W.R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems. 5th International Conference, TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., July 1999.

Andreas Nonnengart, Georg Rock, and Christoph Weidenbach. On generating small clause normal forms. In Claude Kirchner and Hélène Kirchner, editors, *Automated Deduction — CADE-15 International Conference*, LNAI 1421, pages 397–411. Springer, 1998.

Ofer Shtrichman. Tuning SAT checkers for bounded model checking. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., 2000.