Symbolic Model Checking for Propositional Dynamic Logic

Author: Hugo Eskes

Supervisor: Dr. Malvin Gattinger

May 2025

1 Introduction/literature review

In an increasingly digital world, the necessity for testing software properly continues to rise. Model checking is a potent way of verifying software (Beyer & Lemberger, 2017). This is illustrated by the verification of the flight software from NASA's Deep Space One mission. Gluck and Holzmann (2002) utilized the SPIN model checker to discover five different bugs, four of which were deemed critical by the software development team, also stating that these bugs would not have been found using traditional testing processes (Nelson & Pecheur, 2003).

1.1 Propositional Dynamic Logic

One of the logics used for software verification is Propositional Dynamic Logic. Propositional Dynamic Logic (PDL) was invented by Fischer and Ladner (1979) as a way to formally reason about programs. The introduction of modal μ-calculus and other temporal logics such as LTL and CTL, all three of which are more expressive than PDL, resulted in the quick obsolescence of PDL for its original purpose (Lange, 2006). However, PDL is still used frequently in different fields, such as computer science, mathematics, philosophy, and linguistics, for various other purposes (Lange, 2006).

PDL uses two modal operators and four program operators. The two modal operators are $[\alpha]\phi$ and $\langle\alpha\rangle\phi$. The operator $[\alpha]\phi$ checks if ϕ holds in all worlds accessible by executing the program α. The operator $\langle\alpha\rangle\phi$ checks if ϕ holds in at least one world accessible by executing program α. The operator $\alpha;\beta$ is the sequence operator, which executes program α first and then program β. The operator $\alpha\cup\beta$ is the disjunction operator, which executes either program α or program β. The operator α^* is the repeat operator, which executes program α for any finite amount of times, including 0. The operator $\phi?$ is the test operator, that continues if ϕ holds in the current world or fails if ϕ does not hold.

Multiple extensions like intersection, converse, *loop* and *repeat* have been proposed for PDL (Harel & Sherman, 1982; Lange, 2006)

1.2 Symbolic model checking

A crucial challenge for applying model checking to large and complex real-world problems is the *state explosion problem* (Rozier, 2011). The state explosion problem describes the phenomenon where the number of model-states grows exponentially with the number of variables in the model, thus limiting the practical application of model checking. Different techniques utilizing various data structures and algorithms have been used in order to avoid the state explosion problem, but in the worst case, it is still inevitable (Rozier, 2011).

One of the techniques used to mitigate the state explosion problem is *symbolic model checking*. Symbolic model checking was invented by McMillan in 1987 (Clarke, 2008), after which he presented it in his thesis in 1993 (McMillan, 1993). It is based on the idea of representing model states implicitly, not explicitly (Burch et al., 1992). With this technique, CTL and LTL can be model checked on models with many orders of magnitude more reachable states than explicit-state algorithms (Burch et al., 1991; Clarke et al., 1997). This technique is considered a major breakthrough in the world of model checking for its impact on the state explosion problem (Rozier, 2011).

1.3 Reduced Ordered Binary Decision Diagrams

Reduced Ordered Binary Decision Diagrams (BDD) can be used to efficiently store and manipulate the symbolically represented models. BDDs were conceptualized by Bryant (1986) as a data structure for storing and manipulating boolean functions. They are based on the *Binary Decision Diagrams* popularized by Akers (1978), but are *ordered*, so they are constructed based on a specific order of the variables, and they are *reduced*, so they do not have any redundant vertices or duplicate subgraphs. These two properties make them *canonical*, meaning that all boolean functions have a unique BDD. This canonicity makes testing for equivalence, satisfiability or a tautology straightforward (Bryant, 1992). Besides that Bryant (1986) showed that basic boolean operations can be performed efficiently, with a polynomial complexity dependent on the size of the BDD or the product of the BDDs used (Bryant, 1992). When talking about Reduced Ordered Binary Decision Diagrams we use the acronym BDD, not ROBDD, since most literature does the same.

To perform symbolic model checking, two main ingredients are needed: a model, often represented as a Kripke structure, and a property, which can be expressed in different logic systems like CTL, LTL, or PDL. Both the model and the properties need to be translated to a boolean formula stored in a BDD, so the efficient BDD operators can evaluate the property. The main challenge is translating the properties to a boolean formula, which has been done often for CTL and LTL, but is yet to be done for PDL.

Lange (2006) proposed a general model checking algorithm for both pure PDL, PDL without test or other extensions, and PDL with extensions. This algorithm is PTIME-complete. It represents the Kripke structures underlying the model as adjacency matrices. It uses manipulations from linear algebra on these adjacency matrices to check the model. Lacunes (2022) based an open source PDL model checking program on this algorithm, and found that representing the adjacency matrices as sparse matrices helped reduce run time, especially for larger models.

The goal of this thesis is to expand this research done by Lacunes (2022) by evaluating the potential of symbolic model checking to further accelerate model checking PDL and answering the question: how scalable and efficient is Symbolic Model Checking using Binary Decision Diagrams for verifying properties expressed in Propositional Dynamic Logic?

2 Method

To answer this question, an implementation of a symbolic model checker for PDL based on BDDs will be made. This will be done in Python, mainly for its interpretability. This project can function as a proof of concept. If it turns out the symbolic model checker has potential, a lower level programming language could be used for further optimizations.

This section will explain how the PDL semantics will be symbolically represented and how it is implemented in Python code. It starts with an explanation of the general PDL semantics. Then the symbolic representation of the semantics will be explained. Finally, the Python implementation, along with some design considerations, is presented.

2.1 PDL semantics

This section is largely based on the definitions provided by Fischer and Ladner (1979). PDL is based on two sets of symbols, a set Φ_0 of atomic formulas as the propositions and a set Σ_0 of atomic programs. Both these sets are defined by the following rules.

Formulas in Φ:

1. Atomic programs are programs.
2. with p and q as formulas and a as a program, $(p \vee q)$, $\neg p$ and $\langle a \rangle p$ are also formulas.

Programs in Σ:

1. Atomic formulas, `true` and `false`, are formulas.
2. with a and b as programs and p as a formula, $(a \cup q)$, $(a;b)$, $a*$ and $p?$ are also programs.

From these definitions, the following Boolean connectives can be derived.

1. $p \wedge q \equiv \neg(\neg p \vee \neg q)$
2. $p \Rightarrow q \equiv \neg p \vee q$
3. $[a]p \equiv \neg\langle a \rangle \neg p$

2.2 Symbolic representation

2.3 Python implementation

3 Evaluation

For the evaluation of the model checker, the same tests and processes as used in the thesis from Lacunes (2022) will be performed.

References

Akers. (1978). Binary Decision Diagrams. *IEEE Transactions on Computers*, *C-27*(6), 509–516. https://doi.org/10.1109/TC.1978.1675141

Beyer, D., & Lemberger, T. (2017). Software verification: Testing vs. model checking: A comparative evaluation of the state of the art. *Lecture Notes in Computer Science*, *10629 LNCS*. https://doi.org/10.1007/978-3-319-70389-3{_}7

Bryant, R. E. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, *C-35*(8). https://doi.org/10.1109/TC.1986.1676819

Bryant, R. E. (1992). Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys (CSUR)*, *24*(3). https://doi.org/10.1145/136035.136043

Burch, J. R., Clarke, E. M., & Long, D. E. (1991). Representing circuits more efficiently in symbolic model checking. *Proceedings - Design Automation Conference*. https://doi.org/10.1145/127601.127702

Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., & Hwang, L. J. (1992). Symbolic model checking: 10
20 States and beyond. *Information and Computation*, *98*(2). https://doi.org/10.1016/0890-5401(92)90017-A

Clarke, E. M. (2008). The birth of model checking. *Lecture Notes in Computer Science*, *5000 LNCS*. https://doi.org/10.1007/978-3-540-69850-0{_}1

Clarke, E. M., Grumberg, O., & Hamaguchi, K. (1997). Another Look at LTL Model Checking. *Formal Methods in System Design*, *10*(1). https://doi.org/10.1023/A:1008615614281

Fischer, M. J., & Ladner, R. E. (1979). Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, *18*(2). https://doi.org/10.1016/0022-0000(79)90046-1

Gluck, P. R., & Holzmann, G. J. (2002). Using SPIN model checking for flight software verification. *IEEE Aerospace Conference Proceedings*, *1*. https://doi.org/10.1109/AERO.2002.1036832

Harel, D., & Sherman, R. (1982). Looping vs. repeating in dynamic logic. *Information and Control*, *55*(1-3). https://doi.org/10.1016/S0019-9958(82)90553-8

Lacunes, S. (2022). *Model Checking for Propositional Dynamic Logic with Sparse Matrices* (Doctoral dissertation). University of Amsterdam. https://scripties.uba.uva.nl/search?id=record_27607

Lange, M. (2006). Model checking Propositional Dynamic Logic with all extras. *Journal of Applied Logic*, *4*(1). https://doi.org/10.1016/j.jal.2005.08.002

McMillan, K. L. (1993). *Symbolic Model Checking*. Springer. https://doi.org/10.1007/978-1-4615-3190-6

Nelson, S. D., & Pecheur, C. (2003). Formal verification for a next-generation space shuttle. *Lecture Notes in Artificial Intelligence*, *2699*. https://doi.org/10.1007/978-3-540-45133-4{_}5

Rozier, K. Y. (2011). Linear Temporal Logic Symbolic Model Checking. *Computer Science Review*, *5*(2). https://doi.org/10.1016/j.cosrev.2010.06.002