

Faculty of Computer System and Software Engineering

University Malaysia Pahang

**LECTURER : PROF.DR. VITALY MEZHUYEV**

**Title : LTSmin**

**COURSE CODE : BCS 2213**

**(ASSIGNMENT 1)**

**Student Name Matric ID Section**

Syed Nabil Naim Bin Syed Mohd Amudin CB14010 01

**ABSTRACT**

The abbreviation of LTSmin was derived from the word Minimization and Instantiation of Labelled Transition System. This model checking tool offers the ability to solve different types of verification problems as well as dealing with certain algorithm complexity. This research was made to give a clear explanation of LTSmin discovery and the correlation between the many types of Formal Method given structure. Custom made C-code made from the LTSmin was later extended into several modern new modeling languages such as probabilistic (MAPA) and timed systems (UPPAAL). Mathematical multi-core algorithms are also being analyzed to demonstrate the details of the LTSmin general overview as well as the implementation in real-time case situations.

**INTRODUCTION**

LTSmin modeling checking tool utilized several analysis algorithms as well as sharing the same interfaces. Besides that, it also employs both symbolic and a clear state of analysis algorithms in which allow different approaches to different verification problems to be resolved. In order to provide a comprehensive way to deal with different verification problems, Partitioned Next-State-Interface (PINS) was later introduced as a connecting interface in which later produce another labeling function called partitioned successor function (NEXTSTATE). It ease the user to create their own language module as well as to provide faster next-state functions such as SPIN’s (Open Source Software Verification Tool) that was introduced in 1980.

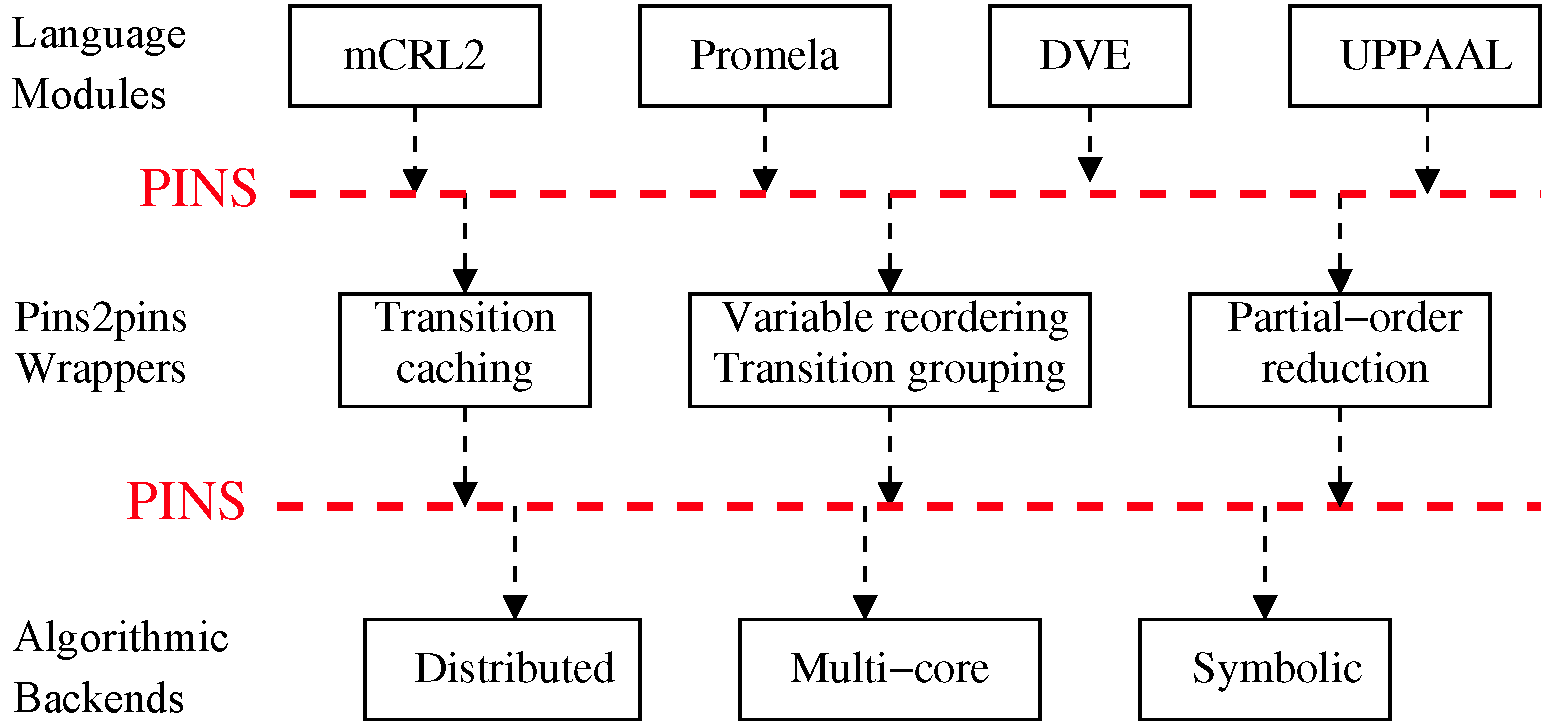


Figure 1.0 : Partitioned Next-State-Interface (PINS) architecture

**LITERATURE REVIEW AND BACKGROUND**

LTSmin maintains as a language-independent model checker tool which connect with new state existing tool such as muCRL, mCRL2, DiVinE, SPIN (SpinS\_ and UPPAAL (opal) and CADP. In order to reach the standard needed for attainability checking, it needs to integrate with symbolic state storage (vector set), fully symbolic (BDD state), attainability state distribution (MPI state) and hybrid of multi-core attainability (mix degree of multi-core compression and upsurge hashing). Moreover, the LTSmin also supplied and revolutionize the state calculation of the product formula using a PINS2PINS module which called Linear Temporal Logic (LTL). The LTL main purpose makes it possible to attain implicit state of detection such as deadlock detection, action detection, and assertion detection.

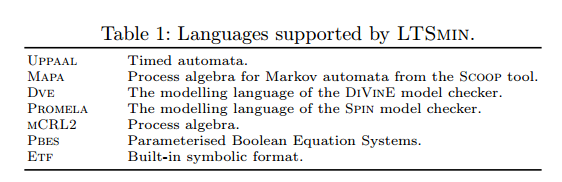


Figure 2.0 : Language supported from the usage of LTSmin

**HOW IT WORKS?**

The supported systems listed are:

* GNU/Linux
* Open SuSE 11.2
* Red Hat Enterprise Linux 6
* MacOS X Version 10.10 “Yosemite”
* MacOS X version 10.6 “Snow Leopard”

The LTSmin also have inbuilt existing verification tools which enables:

* muCRL’s process algebra
* mCRL2’s process algebra
* SPIN’s Promela via SpinS
* UPPAAL’s timed automated

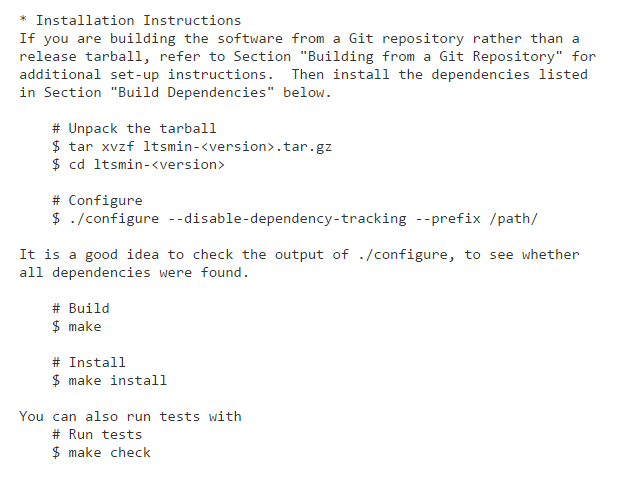


Figure 3.0 : Example of LTSmin installation in Linux Platform

**WHY IT IS NEEDED?**

In year 2012, 2013, and 2014 LTS involves in RERS challenges and won several first prizes. LTSmin can deal several RERS difficult and challenging problems. For example, backend tool check for temporal and assertion characteristics was demonstrated by the usage of LTSmin. Other than that, LTSmin enables the translation of RERS programs to PROMELA and to MCRL2. It is all in separate binary (.so file) which offers reliability and robustness during the whole running of the model coding checking. Other than that, it also user-friendly with logical theorem proving as well as process control system.

**SAMPLE/ IMPLEMENTATION**

This figure below shows LTSmin computes full state of error checking within 2 minutes divided over 480 BFS levels stored in 1 million MDD nodes. As compared to running a 48-core machine with 132 GB RAM.

D:\~Formal Method~\Assignment 1\5.PNG

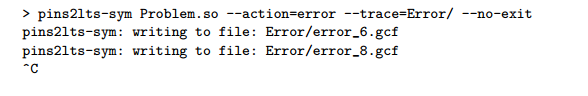


Figure 4.0: Show LTSmin Running for Error Checking

**CONCLUSION**

LTSmin offers great information for error checking and proving the model checker in user-friendly yet automatic and fast approaches. Other than that, the given formula makes it possible to perform in high performance mode and language-independent model. Besides that, the model checkers also supports on the go symbolic verification and full LTL model checking with POR depending to modular architecture. Finally, the toolsets also offers brief qualitative comparison to the available types of algorithms with supported logics.

**REFERENCES**

1. <https://github.com/utwente-fmt/ltsmin>
2. <http://fmt.cs.utwente.nl/tools/ltsmin/>
3. http://link.springer.com/chapter/10.1007%2F978-3-642-14295-6\_31#page-1
4. <http://www.researchgate.net/publication/221413038_Multi-Core_LTSmin_Marrying_Modularity_and_Scalability>
5. <https://books.google.com.sg/books?id=TSyqCAAAQBAJ&pg=PA354&lpg=PA354&dq=ltsmin+formal+method&source=bl&ots=W_Q5N5hR5M&sig=0WcVAE1qyjUjULm1vUlpQkJ54LE&hl=en&sa=X&redir_esc=y#v=onepage&q=ltsmin%20formal%20method&f=false>
6. https://books.google.com.sg/books?id=LN69BwAAQBAJ&pg=PA692&lpg=PA692&dq=ltsmin+formal+method&source=bl&ots=aj3DgxCxZr&sig=59ACmA3hC7bDhtJmoLSDaH5isPU&hl=en&sa=X&redir\_esc=y#v=onepage&q=ltsmin%20formal%20method&f=false