Concurrent Model Repair: Eshmun

Models, Repairs, and Formal Verification

Ali Cherri[1], Kinan Dak Al Bab[1]

[1] American University of Beirut
Department of Computer Science

19 May 2015

Getting Started

Why Formal Verification ?

Computing Catastrophes [1]

- Ariane 5 : Blew up due to code re-use without verification.
- Therac-25 : Radiation Therapy Machine. Killed 6 due to a software error.
- Mars Climate Crasher : Crashed on Mars after 286-days trip. Due to unit conversions.
- AT&T : AT&T assumed it was being hacked. In fact, the company's long distance switches kept rebooting in sequence.

Dijkstra's Thoughts [2]

The formal provability of a program is a major criterion for correctness : the central challenges of computing hadn't been met, due to an insufficient emphasis on program correctness.

Modeling Sequential Programs

Kripke Structure :

- State-transition Diagram : A Directed Graph where nodes are states of the program, and edges are transitions.
- Used to model sequential programs.
- The full diagram represents all the possible executions of the program.
- Each state satisfies certain properties (Boolean Formulae).

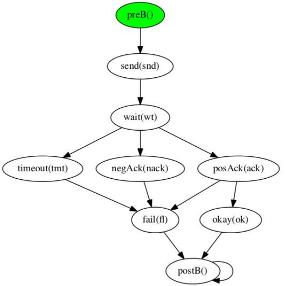


FIGURE: Kripke Structure for a Phone call

Modeling Asynchronous Concurrent Programs

Multi-Kripke Structure :

- Several Kripke Structure.
- Each Kripke Structure relates to two processes.
- Doing things in **pairwise representation** is more efficient !

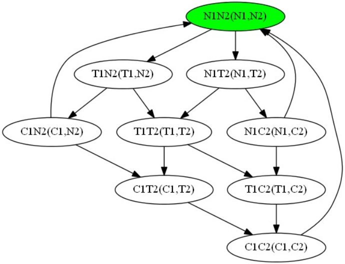


FIGURE: Pair Kripke Structure

Temporal Logic

Definition

Temporal Logic is a system of rules and boolean propositions that are quantified in terms of time. e.g. I am always Hungry.

Computation Tree Logic (CTL)[4]

- Subset of Temporal Logic.
- Useful in tree-like structures where the future is not determined.
- Path Operators : quantify over a branch, path, or combination of both.
- CTL is useful for expressing properties like liveness and safety.

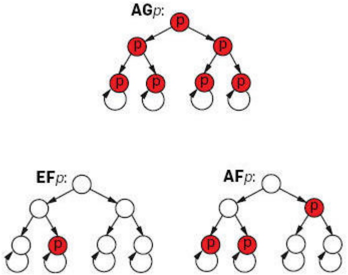


FIGURE: Notable Temporal Operators

Model Checking[4]

- Used to verify Model Correctness.
- Given a Model and its Specification (In CTL), determines if the Model satisfies the specs.
- Great impact on industry :
  - Used extensivly to verify correctness of Hardware.
  - Increasing use in verifying correctness of Software (**Windows Device Drivers**).

Emerson, Clarke, and Sifakis were given the **Turing Award** in 2007 for inventing model checking.

Model Repair

- We are given a Model and its specifications, the model doesn't satisfy the specifications.
- We want to apply a **Subtractive Model Repair Algorithm** in order to get a new model that satisfies the specifications.
- This requires us to find the un-wanted transitions/states that are causing the model to be incorrect.
- These transitions/states should be **deleted** in a way that doesn't affect the program's flow and totality.
- The decision version of the problem is proved to be NP-Complete[5].

Model Repair : The Solution

- We encode the problem as a boolean formula.
- The formula contains different types of variables.
- The value of each variable indicates whether the corresponding state (or transition) is kept or deleted in the new model.
- The formula forces the new model to be total and have some start state, it also relates States to transitions.
- The formula is then passed to a SAT solver, each satisfying assignment would represent a repaired model.
- The Overall length of the repair formula is $O(|States|^2 \times |Specifications| \times Out\_degree + |States| \times |Transitions| \times |Propositions|)$ [5].

Model Repair : Getting the New Model

Boolean Satisfiability Problem (SAT)

Determining If a boolean formula is Satisfiable or not. Is there an assignment that makes the formula true ? Boolean SAT is NP-Complete

Repairing with SAT Solver

- By construction, if the repair formula is unsatisfiable then the model is un-repairable.
- Furthermore, the satisfying assignment of the formula is the needed repair.
- The satisfying assignment tells us which transitions and states are to be deleted.
- If all the start states are deleted then the model is un-repairable.

What was already done

- A tool that repairs Single Kripke was done by M. Sakr (master's thesis).
- The tool used a Model Checker written by Emile Chartouni.
- The tool was a proof of concept.
- The tool had techniques that allow you to reduce the model into a smaller one to make repair more efficient.
- Repairing Mutual Execlusion for 4 Processes required more than 12GB of RAM.
- Data structures for models and repair algorithm were impossible to extend.

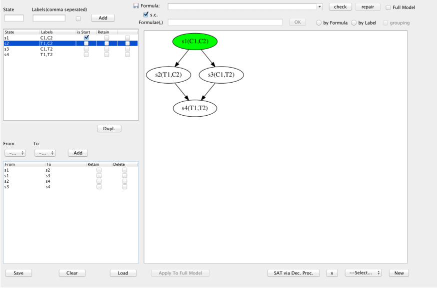


FIGURE: The old tool

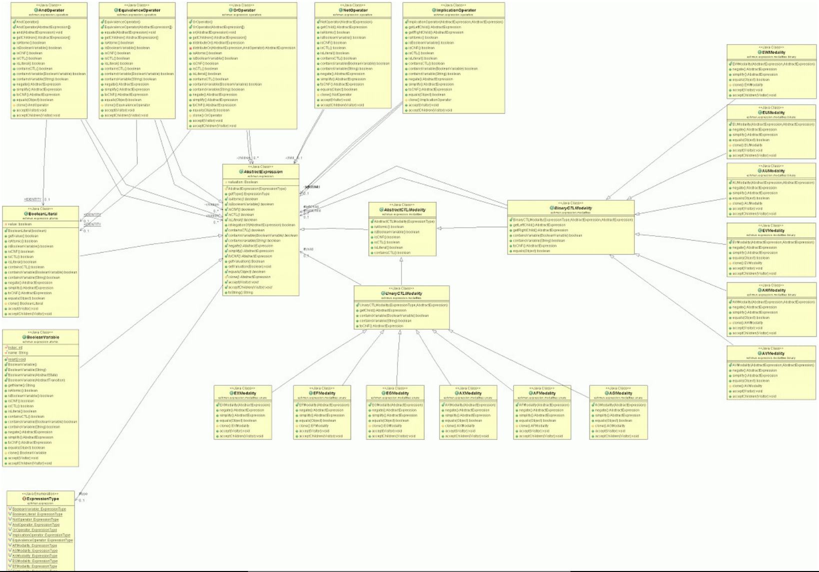
Our Contribution : Optimizing the tool

- Portability : The previous tool only ran on windows.
- An easy to use, interactive, and **User Focused** GUI.
- Feeding the SAT Solver from memory using a custom InputStream.
- The ability to iterate through different repairs.
- An IDE like enviroment for inputing Boolean Formulae.
- Memory Optimization, Implemented a different CTL-Logic API.
- New Data structures for models, Respecting OOP Guide lines.
- Complete seperation of modules, each functionality has its own classes in its own packages.
- Porting the Model Checker and Model Repair to use the new Logic API and datastructures.
- We re-implemented Everything ! !

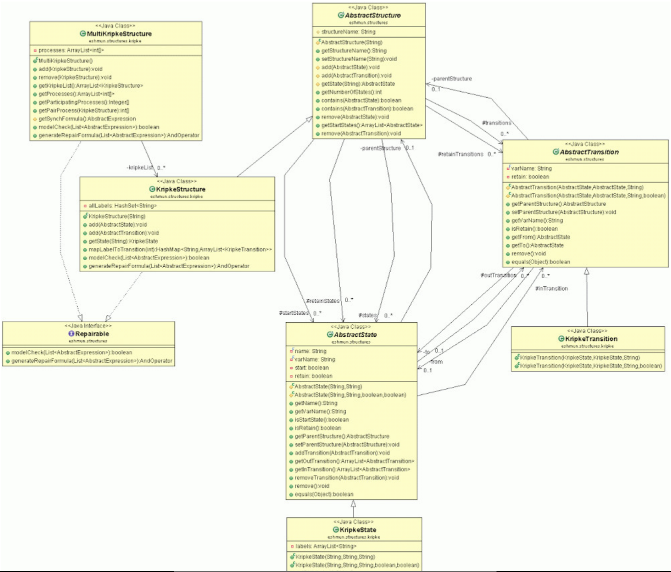
Our Contribution : Multi Kripke Repair

- The ability to repair concurrent programs, modeled as Multi-Kripke Structures **Breakthrough**.
- Each of the sub-kripke structure has its own repair formula.
- All the sub-repair formulae are conjucted.
- A formula for synchronizing the processes across different sub-structures is automatically inferred and added to the complete repair formula.

Logic API


AndOperator
EquivalenceOperator
OrOperator
NotOperator
ImplicationOperator
AbstractExpression
BooleanLiteral
BooleanVariable
ExpressionType
AbstractCTLModality
BinaryCTLModality
UnaryCTLModality
EWModality
EUModality
AUModality
EVModality
AWModality
AVModality
EXModality
EFModality
EGModality
AXModality
AFModality
AGModality

Data Structures



Some Bragging

Meterics

- Number of classes and interfaces : 88
- Number of methods : 1,027
- Lines of code : 19,103

Technologies Used

- Java : The tool is written in java, it uses extensively concepts from OOP.
- Java Swing and Graphics2D : used to develop the GUI and the graphics.
- ANTLR : for building a parser for CTL logic expressions.
- SAT4J : a sat solver for java.
- PYTHON : For writting testing scripts, and generating different models.
- JavaDoc : A complete and extensive documentation of the code is provided.
- HTML : Used in the documentation and in the user manuel.

Benchmarks

Mutual Exclusion

- 2 Process : 107MS — 93MS
- 3 Process : 104MS — 1,210MS
- 4 Process : 248MS — Out of Memory

Barrier Synchronization

- 2 Process : 63MS —- 67MS
- 3 Process : 107MS — 460MS
- 4 Process : 164MS — 1.1Second

These numbers result from repairing the given problems using a single Kripke Structure.

Future Work

- Get it published.
- Support more Concurrent Models : I/O Automata, BIP.
- Extend to Infinite State Models.
- Addative Repair Algorithm : add new transitions and states.

Credits

Credits

- We would like to thank our advisor Dr. Paul Attie, he offered a lot of help and guiding.
- We would like to thank Mohammad Ali Baydoun and George E. Zakhour for providing us with an algorithm and a working routine to auto space and format a graph, this was done by simulating the graph as a physical system with attraction and repulsion forces.

Summary

- We can Model behavior of programs using formal tools (Kripke, Multi-Kripke, ...).
- Temporal Logic is an extension of First order logic that is useful to quantify over time (execution paths in case of CTL).
- By using model checking we can know if a program satisfies a given specification (CTL Formula).
- We can also Generate a repair formula for the model based on the definition of temporal operators.
- Using a SAT Solver we can get a satisfying assignment (if it exists) which is by construction the repair formula.
- We have implemented a tool that does all that.

References

[1] Computer World **"Infamous 11 software Bugs"**. http ://www.computerworld.com/article/2515483/enterprise-applications/epic-failures–11-infamous-software-bugs.html ?page=4

[2] Dijkstra, Edsger W. **"On the Cruelty of Really Teaching Computing Science"**. E.W. Dijkstra Archive. Center for American History, University of Texas at Austin.

[3] Lynch, Nancy A. ; Tuttle, Mark R. (August 1987). **"Hierarchical correctness proofs for distributed algorithms"**. Proceedings of the sixth annual ACM Symposium on Principles of distributed computing. PODC '87

[4] Pelanek, Radek. **"Formal Verication, Model Checking"**, Talk in "investice do rozvoje vzdělávání". http ://www.fi.muni.cz/ xpelanek/IA158/slides/verification.pdf

[5] Attie, Paul. Sakr, Mouhammad Issam. (2014). **"Model Repair Via SAT Solving"**. Thesis. M.S. American University of Beirut. Department of Computer Science, 2014. T :6108

Conclusion

Thank You