Insert Title Here

by

Mathew Kuthur James

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2026

© Mathew Kuthur James 2026

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Meta Meta Professor, Cheriton School of Computer Science, University of Waterloo
Supervisor:	Nancy Day Professor, Cheriton School of Computer Science, University of Waterloo
Internal Member:	Meta Meta Professor, Cheriton School of Computer Science, University of Waterloo
Internal-External Member:	Meta Meta Professor, Cheriton School of Computer Science, University of Waterloo
Other Member(s):	Meta Meta Professor, Cheriton School of Computer Science, University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This is the abstract.

Lorem ipsum dolor sit amet

Acknowledgements

I would like to thank all the little people who made this thesis possible.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

background - discuss formal verification, TLA+, Alloy, Dash, model-checking etc.

goal of the project:

1. build a translator for Dash+ and Alloy
2. evaluate performance

Chapter 2

Background

2.1 Alloy

2.2 Dash

2.3 TLA+

Objects whose value do not change with the traces of the model are called CONSTANTS in TLA+. Often, these CONSTANTS are given values in the .cfg file accompanying the .tla file fed into the TLC model checker. These objects correspond to instances generated by the Alloy Analyzer when a Dash+ model is run after translation to Alloy.

2.4 Configuration generation

In a paper [4] by Cunha et. Somson, the authors introduce a tool called Blast, which enables the use of TLA+ to find these objects during the model-checking phase, rather that having the user set specific values manually in the .cfg file.

The .tla file and .cfg file are augmented with annotations read by blast. These annotations determine the type and scope of the configurations to be tested. Blast produces a modified .tla file and a modified .cfg file.

Blast performs the following procedure on the input files:

1. For every CONSTANT C in the original .tla file, Blast adds a VARIABLE V_C in the modified .tla file. (CONSTANTS with no type annotations are ignored by Blast)

2. Blast makes these variables keep the same values during the trace, using the UNCHANGED modifier in TLA+ (equivalent to appending / $V_C' == V_C$ to every transition).

3. Blast adds the type of the variable to the TypeOK relation in TLA+. Blast ignores CONSTANTS whose type is not specified in the annotation, which remain as CONSTANTS in the output files produced by Blast.

4. Blast constructs compound types (such as sets, relations, tuples, et. cetera.) from the annotation, which are constructed using basic types (such as Nat). Blast identifies basic types by the use of all caps identifiers.

5. Blast modifies the Init relation by adding $V_C \in C$, where C is now a CONSTANT.

6. The value of C in the .cfg file is a set whose size is determined by the scope annotations read by Blast.

Chapter 3

Translating Alloy elements to TLA+

3.1 Signatures

Signatures are modelled as sets, explain in greater detail with example code for a simple signature

3.1.1 Fields

3.1.2 Multiplicities

Explain the four multiplicities with two basic types, the use of quantifiers over cardinality as optimizations and set comprehension.

3.2 Expressions/Formulae

3.3 Facts

3.4 Predicates/Functions

Chapter 4

Translating Dash+ elements to TLA+

A Dash+ model is written with Alloy expressions and code embedded inside Dash structures. The Dash structures are used to model a transition system. Since TLA+ supports transitions natively, the dash elements are translated directly to TLA+, without an intermediate Alloy step.

4.1 States

Each state is associated with a unique string in TLA+. The uniqueness of this string is guaranteed by using the fully qualified name of the state. No two states in a Dash model can have the same fully qualified name, so turning the fully qualified name into a string preserves uniqueness. These strings are associated with formulae in TLA+, and all further references to the state is made via the formula, rather than the string.

The set of all states and the set of all default states are assigned their own formulae, which are used later in the translation. The current state is modelled as a TLA+ VARIABLE, whose type depends on the Dash model. In the general case, the type is a set. For Dash models without concurrency, where the system cannot be in multiple states simultaneously, the type of the state VARIABLE is a string.

TODO: find out embedding code snippets and give examples

TODO: explain state hierarchies here.

4.2 Events

This doesn't have an implementation yet, all examples are hand-translated as of now.

TODO: explain big-step small-step semantics

4.3 Transitions

Transitions are modelled as boolean formulae in TLA+. In Dash, the transitions are defined inside the state from which the transitions is taken. In the translation, all transitions are defined at the top-level. Each transition has the following:

- (implicit) - the 'from' state, translated as state = {from state}
- goto - the 'to' state, translated as state' = {to state}
- when - the guard, translated as an Alloy expression
- do - the action, translated as an Alloy expression
- on - the triggering event
- (emitting event - todo: fill this up post discussion)

4.4 Actions

Alloy

4.5 Guards

Alloy

Chapter 5

Translating LTL properties to TLA+ invariants

5.1 Commands

Chapter 6

Evaluation

The evaluation of the translator is broadly divided into three sections: Correctness, performance of the model, and performance of the translator.

The translator produces a tla+ file from a .dsh file, which can be analyzed statically for complexity metrics, such as the number of lines of code. This can be compared to the number of lines required for a manual translation for specific models, to observe how efficient the translator is at capturing the semantics of the model.

Through the analysis of simple models, certain optimization techniques are revealed and implemented, in an iterative process. However, a weakness of such a system is overfitting to certain kinds of models, whereby optimizing for small, hand-written models may have undesirable effects on the efficiency of other types of models.

The correctness of the translation is a binary correct/not-correct. Since the space of all possible models cannot be reasoned about, a determination of correctness is made via:

- custom unit tests of known models
- fuzz-testing using randomly generated models

To check for specific instances, a pair (model, instance) is preprocessed into model', which is a model that tests specifically for that instance. Model' is translated and tested for the only property present.

TODO: find an analogue

The following metrics are used to judge the performance of the model:

- time taken to execute, as compared to the time on the Analyzer
- clock cycles to execute (usually tightly correlated with time)
- trace lengths generated during execution
- state space of the translated model, in comparison to the original model in the Analyzer

TODO: formalize the notion of specific model types in terms, and discuss how the performance is affected

Chapter 7

Related Work

A list of papers with related work and their summaries, of the form:

A paper by A Cunha (todo: cite this from the journal it was published in, not the arXiv preprint) compares the performance of Alloy and TLA+ implementations of the spanning tree algorithm (used in cite Blast paper). The authors constructed a model of the algorithm by hand for both languages, and compared the performance of the TLA+ implementation with the Alloy implementation. They concluded that in general, TLA+ outperforms Alloy when dealing with simple configurations with large scopes, while Alloy is more suitable for models with complex configurations whose validity takes a non-trivial amount of work by the model-checker to prove. The findings of this paper were limited to a single model, written manually in both languages. The thesis involves comparing Alloy models to their machine-translated TLA+ models.

Chapter 8

Conclusion

Summary of the thesis and possible future work.

References

[1] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, revised edition, 2016.

[2] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, Boston, MA, United States, second edition, 2002.

[3] Jose Serna, Nancy Day, and Shahram Esmaeilsabzali. Dash: declarative behavioural modelling in alloy with control state hierarchy. *Software and Systems Modeling*, 22, 08 2022.

[4] Paul Somson and Alcino Cunha. Verifying Multiple TLA+ Configurations with Blast. In *2025 IEEE/ACM 13th International Conference on Formal Methods in Software Engineering (FormaliSE)*, pages 135–145, 2025.

APPENDICES

Appendix A

Matlab Code for Making a PDF Plot

A.1 Using the Graphical User Interface

Properties of Matab plots can be adjusted from the plot window via a graphical interface. Under the Desktop menu in the Figure window, select the Property Editor. You may also want to check the Plot Browser and Figure Palette for more tools. To adjust properties of the axes, look under the Edit menu and select Axes Properties.

To set the figure size and to save as PDF or other file formats, click the Export Setup button in the figure Property Editor.

A.2 From the Command Line

All figure properties can also be manipulated from the command line. Here's an example:

```
x=[0:0.1:pi];
hold on % Plot multiple traces on one figure
plot(x,sin(x))
plot(x,cos(x),'--r')
plot(x,tan(x),'.-g')
title('Some Trig Functions Over 0 to \pi') % Note LaTeX markup!
legend('{\it sin}(x)','{\it cos}(x)','{\it tan}(x)')
hold off
```

```
set(gca,'Ylim',[-3 3]) % Adjust Y limits of "current axes"
set(gcf,'Units','inches') % Set figure size units of "current figure"
set(gcf,'Position',[0,0,6,4]) % Set figure width (6 in.) and height (4 in.)
cd n:\thesis\plots % Select where to save
print -dpdf plot.pdf % Save as PDF
```