# A Verified and Compositional Translation of LTL to Deterministic Rabin Automata

Julian Brunner, Benedikt Seidl, Salomon Sickert

ITP 2019

Technical University of Munich

## Introduction

## Terminology

What is LTL?

- LTL = Linear Temporal Logic
- Modal logic interpreted on infinite sequences of valuations
- Example: $\mathbf{G}(\mathit{request} \longrightarrow \mathbf{F}\,\mathit{response})$

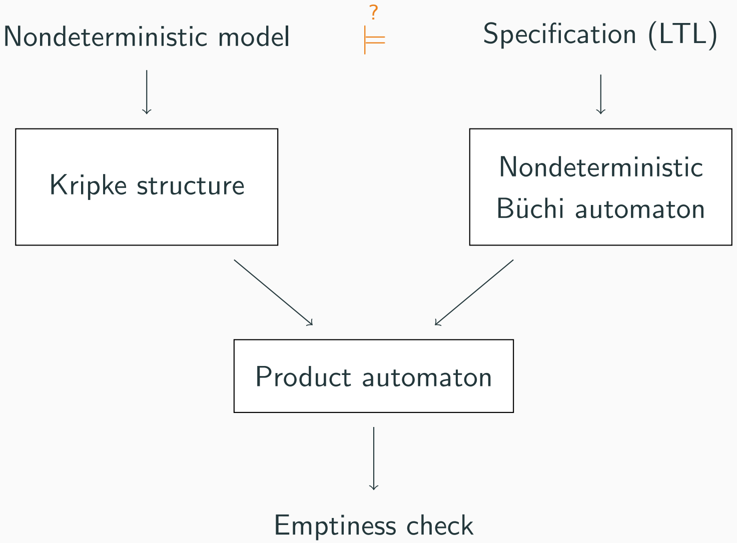
Why do we want to translate LTL to automata?

- Model Checking

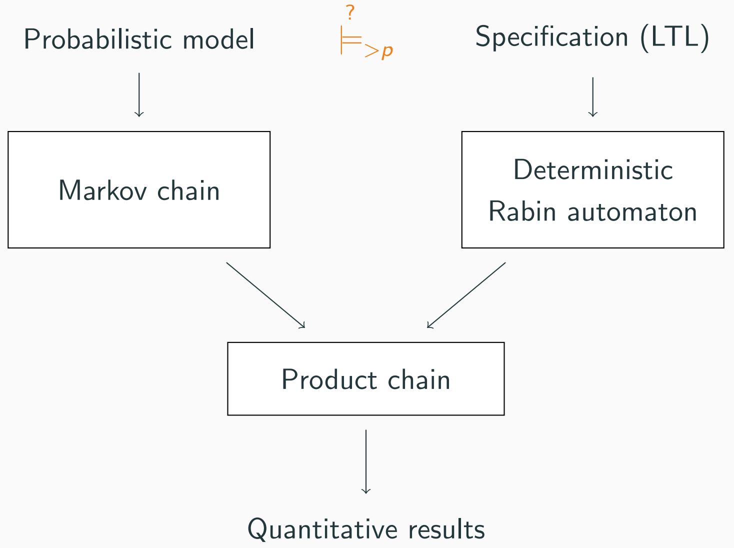
Why do we need deterministic automata?

- Probabilistic Model Checking

## Model Checking



## Probabilistic Model Checking



## Motivation

Model checking is a widespread technique for software verification.

However, model checkers are themselves very complex software.

Their bugs can lead to false confidence in the checked software.

## Motivation

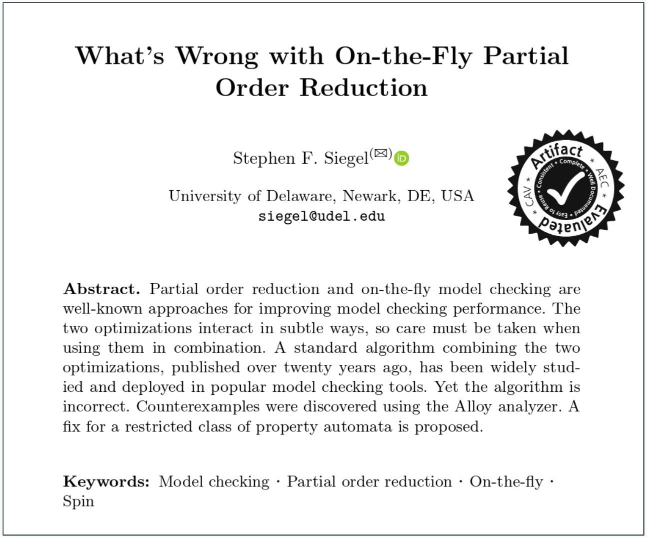
**Spin** is a popular model checker.

- Initially released in 1989
- 44,000 lines of C code

Used in real world applications:

- Infrastructure projects (algorithms for flood control barrier)
- Actual rocket science (Mars Science Laboratory, Deep Space, Cassini, Mars Exploration Rovers, Deep Impact, etc.)

## Motivation



# What's Wrong with On-the-Fly Partial Order Reduction

Stephen F. Siegel[✉]

University of Delaware, Newark, DE, USA
siegel@udel.edu

**Abstract.** Partial order reduction and on-the-fly model checking are well-known approaches for improving model checking performance. The two optimizations interact in subtle ways, so care must be taken when using them in combination. A standard algorithm combining the two optimizations, published over twenty years ago, has been widely studied and deployed in popular model checking tools. Yet the algorithm is incorrect. Counterexamples were discovered using the Alloy analyzer. A fix for a restricted class of property automata is proposed.

**Keywords:** Model checking · Partial order reduction · On-the-fly · Spin

CAV 2019

## Motivation

"A standard algorithm combining the two optimizations, published over twenty years ago, has been widely studied and deployed in popular model checking tools. Yet the algorithm is incorrect."

— Stephen F. Siegel, CAV 2019

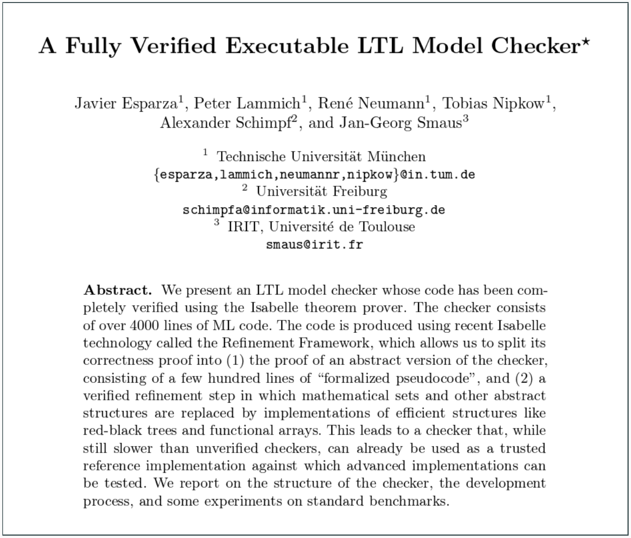
## Motivation

This sounds really bad. What can we do about it?

We verify the model checker using **interactive theorem provers**.

Verifying a large piece of software like Spin is infeasible. Instead, we formalize a **reference implementation** that can be tested against.

## Preliminary Work



# A Fully Verified Executable LTL Model Checker*

Javier Esparza[1], Peter Lammich[1], René Neumann[1], Tobias Nipkow[1], Alexander Schimpf[2], and Jan-Georg Smaus[3]

[1] Technische Universität München
{esparza,lammich,neumannr,nipkow}@in.tum.de
[2] Universität Freiburg
schimpfa@informatik.uni-freiburg.de
[3] IRIT, Université de Toulouse
smaus@irit.fr

**Abstract.** We present an LTL model checker whose code has been completely verified using the Isabelle theorem prover. The checker consists of over 4000 lines of ML code. The code is produced using recent Isabelle technology called the Refinement Framework, which allows us to split its correctness proof into (1) the proof of an abstract version of the checker, consisting of a few hundred lines of "formalized pseudocode", and (2) a verified refinement step in which mathematical sets and other abstract structures are replaced by implementations of efficient structures like red-black trees and functional arrays. This leads to a checker that, while still slower than unverified checkers, can already be used as a trusted reference implementation against which advanced implementations can be tested. We report on the structure of the checker, the development process, and some experiments on standard benchmarks.

CAV 2013

## Preliminary Work

**CAVA** is a verified and executable model checker à la Spin.

- 16,000 lines of Isabelle/HOL code
- During formalization of On-the-Fly Partial Order Reduction, Julian stumbled over the mentioned soundness issue.
- Translation of LTL to Nondeterministic Büchi automata
- Limited to model checking on Kripke structures

We want to extend the idea of CAVA to **probabilistic model checking**. Hence we need translations to (partially) deterministic automata.

## A Compositional Approach

## Current Situation

Translations from LTL to automata are quite complicated.

Currently, the two formalizations in Isabelle do not share any code and even define their own automata models.

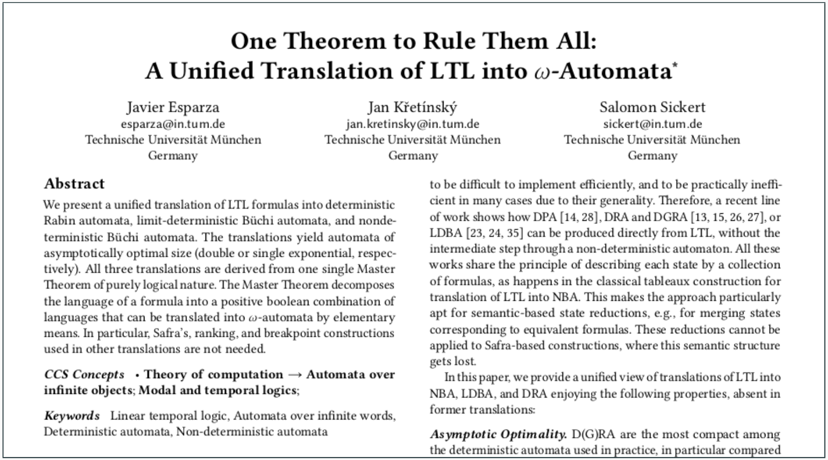
For deterministic automata, they are even harder since the minimal automaton can have **double exponentially** many states.

Getting deterministic has been an open problem for a long time as the constructions are way more complicated.

## Current Situation

$$\begin{tikzcd}
 & \text{NBA} \\
\text{LTL} \arrow[ru, "\sim 4{,}300 \text{ LOC}"] \arrow[r, dashed] \arrow[rd, "\sim 11{,}100 \text{ LOC}"'] & \text{LDBA} \\
 & \text{DRA}
\end{tikzcd}$$

## The Master Theorem



# One Theorem to Rule Them All: A Unified Translation of LTL into $\omega$-Automata*

Javier Esparza
esparza@in.tum.de
Technische Universität München
Germany

Jan Křetínský
jan.kretinsky@in.tum.de
Technische Universität München
Germany

Salomon Sickert
sickert@in.tum.de
Technische Universität München
Germany

**Abstract**

We present a unified translation of LTL formulas into deterministic Rabin automata, limit-deterministic Büchi automata, and nondeterministic Büchi automata. The translations yield automata of asymptotically optimal size (double or single exponential, respectively). All three translations are derived from one single Master Theorem of purely logical nature. The Master Theorem decomposes the language of a formula into a positive boolean combination of languages that can be translated into $\omega$-automata by elementary means. In particular, Safra's, ranking, and breakpoint constructions used in other translations are not needed.

***CCS Concepts*** • **Theory of computation** → **Automata over infinite objects**; **Modal and temporal logics**;

***Keywords*** Linear temporal logic, Automata over infinite words, Deterministic automata, Non-deterministic automata

to be difficult to implement efficiently, and to be practically inefficient in many cases due to their generality. Therefore, a recent line of work shows how DPA [14, 28], DRA and DGRA [13, 15, 26, 27], or LDBA [23, 24, 35] can be produced directly from LTL, without the intermediate step through a non-deterministic automaton. All these works share the principle of describing each state by a collection of formulas, as happens in the classical tableaux construction for translation of LTL into NBA. This makes the approach particularly apt for semantic-based state reductions, e.g., for merging states corresponding to equivalent formulas. These reductions cannot be applied to Safra-based constructions, where this semantic structure gets lost.

In this paper, we provide a unified view of translations of LTL into NBA, LDBA, and DRA enjoying the following properties, absent in former translations:

***Asymptotic Optimality.*** D(G)RA are the most compact among the deterministic automata used in practice, in particular compared

LICS 2018

## The Master Theorem

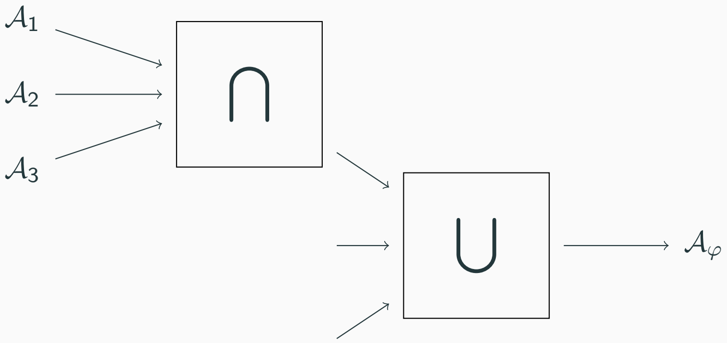
We decompose the language of our LTL formula into a Boolean combination of “simpler” languages using the **Master Theorem for LTL**.

Boolean operations

$$\mathcal{L}(\varphi) = \bigcup (L_1 \cap L_2 \cap L_3)$$

“simple” languages

## Construction Blueprint



## Simple Languages

Why are these languages simple?

- We can define very simple deterministic (co-)Büchi automata accepting them.

For $L_2$ and $L_3$ we need another intersection of automata:

$$L_2 = \bigcap \mathcal{L}(\mathbf{GF}(\psi)) \qquad \text{where } \psi \text{ is a } \textit{safety} \text{ formula}$$

$$L_3 = \bigcap \mathcal{L}(\mathbf{FG}(\psi)) \qquad \text{where } \psi \text{ is a } \textit{co-safety} \text{ formula}$$

## A Compositional Approach

$$\begin{tikzcd}
 & \text{NBA} \\
\text{LTL} \arrow[ur, sloped, "\sim \text{4,300 LOC}"] \arrow[r, dashed] \arrow[dr, sloped, "\sim \text{11,100 LOC}"'] & \text{LDBA} \\
 & \text{DRA}
\end{tikzcd}$$

## A Compositional Approach

$$\begin{tikzcd}
 & & \text{NBA} \\
\text{LTL} \arrow[r, "\sim 2{,}700 \text{ LOC}"] & \text{Master Theorem} \arrow[ur, dashed] \arrow[r, dashed] \arrow[dr, "\sim 1{,}600 \text{ LOC}"'] & \text{LDBA} \\
 & & \text{DRA}
\end{tikzcd}$$

## Transition Systems and Automata

## Motivation

Why do we need a new library for transition systems?

- In our construction we use many different automata models.
- Generic formalizations are superior to ad-hoc definitions.
- We have learned from existing libraries which are either difficult to use or not flexible enough.

## Library Goals

Our library should have the following features:

- Support for a wide range of automata
- Definition of concepts on the most abstract level
- Formalization of Boolean operations
- Good usability and proof automation

## Library Goals

Although transition systems seem to be a very simple concept, they can have many different nuances in the representation.

**Examples**

**Incomplete Deterministic Transition System**

$$\text{succ} :: \rho \Rightarrow \alpha \Rightarrow \rho\,\text{option}$$

**Implicit Nondeterministic Transition System**

$$\text{succ} :: \rho \Rightarrow \alpha \Rightarrow \rho\,\text{set}$$

**Explicit Nondeterministic Transition System**

$$\text{succ} :: (\rho \times \alpha \times \rho)\,\text{set}$$

## Abstract Transition System

We use locale instantiations instead of specializations of one general datatype to support these different representations.

**Definition**

**locale** transition-system =
  **fixes** execute :: *transition* ⇒ *state* ⇒ *state*
  **fixes** enabled :: *transition* ⇒ *state* ⇒ bool

- "execute" calculates the target state for a transition.
- "enabled" returns whether a transition can actually be taken.

## Concrete Automata

### Examples

**Incomplete Deterministic Transition System**

$$\text{execute} = \lambda a\ p.\ \text{the}\ (\text{succ}\ a\ p) \qquad \mathit{transition} = \alpha$$

$$\text{enabled} = \lambda a\ p.\ \text{succ}\ a\ p \neq \text{None} \qquad \mathit{state} = \rho$$

**Implicit Nondeterministic Transition System**

$$\text{execute} = \lambda (a, q)\ p.\ q \qquad \mathit{transition} = \alpha \times \rho$$

$$\text{enabled} = \lambda (a, q)\ p.\ q \in \text{succ}\ a\ p \qquad \mathit{state} = \rho$$

**Explicit Nondeterministic Transition System**

$$\text{execute} = \lambda (a, q)\ p.\ q \qquad \mathit{transition} = \alpha \times \rho$$

$$\text{enabled} = \lambda (a, q)\ p.\ (p, a, q) \in \text{succ} \qquad \mathit{state} = \rho$$

## Constant Definitions

We define the basic concepts of transition systems on the abstract level. They are available in every instantiation of our locale.

**Definitions**

$$\begin{aligned}
\mathrm{target} &:: \mathit{transition}\ \mathrm{list} \Rightarrow \mathit{state} \Rightarrow \mathit{state} \\
\mathrm{trace} &:: \mathit{transition}\ \mathrm{list} \Rightarrow \mathit{state} \Rightarrow \mathit{state}\ \mathrm{list} \\
\mathrm{strace} &:: \mathit{transition}\ \mathrm{stream} \Rightarrow \mathit{state} \Rightarrow \mathit{state}\ \mathrm{stream}
\end{aligned}$$

$$\begin{aligned}
\mathrm{path} &:: \mathit{transition}\ \mathrm{list} \Rightarrow \mathit{state} \Rightarrow \mathrm{bool} \\
\mathrm{spath} &:: \mathit{transition}\ \mathrm{stream} \Rightarrow \mathit{state} \Rightarrow \mathrm{bool}
\end{aligned}$$

## Library Structure

$$\begin{tikzcd}
\text{Abstract Transition System} \arrow[d, dashed] & & \\
\text{Concrete Transition System} \arrow[dd] & & \text{Automaton datatype} \\
& & \boxed{\Sigma\ Q_0\ \Delta}\ A \arrow[ull] \arrow[dll, bend left] \\
\mathcal{L} & & \text{Concrete Automaton}
\end{tikzcd}$$

## Concrete Automata

For the following translation formalisation, we contribute deterministic Büchi, co-Büchi and Rabin automata.

Furthermore, we implement the corresponding union and intersection operations on these automata classes.

## Deriving the DRA construction

## Assembling the Pieces

We construct automata for the simple languages $L_1$, $L_2$, and $L_3$, and combine them using the Boolean operations provided by the automata library.

The following lemma follows directly from the Master Theorem:

$$\text{language } (\text{ltl-to-dra } \varphi) = \text{language } \varphi$$

## Verified Size Bound

The verified size bound guarantees that our construction produces at most double exponentially large automata.

$$\mathsf{card}\ (\mathsf{nodes}\ (\mathsf{ltl\text{-}to\text{-}dra}\ \varphi)) \leq 2^{2^{2 \cdot (\mathsf{size}\ \varphi) + \log_2 (\mathsf{size}\ \varphi) + 4}}$$

For the existing LTL to DRA construction, only a triply exponential bound was shown.

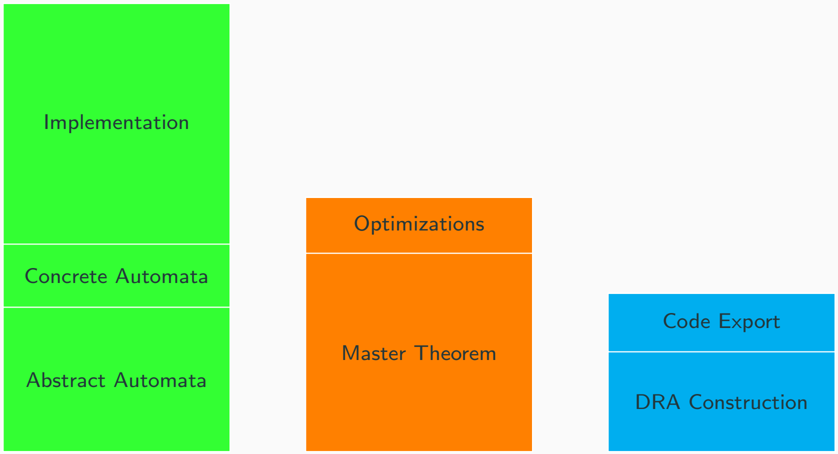
## Code Export

In order to output the resulting automaton, we use Isabelle's Refinement Framework and DFS Framework.

The constants are exported to Standard ML and compiled to an executable tool.

The only unverified parts are the LTL parser and the output to HOA, a standardized automata serialization format.

## Statistics



Automata Library

4,800 LOC

Logical Characterization

2,700 LOC

LTL to DRA

1,600 LOC

## Current and Future Work

Optimizations on several levels

- Improved Master Theorem
- Better intersection constructions
- Improving performance using data refinement

Additional automata models

- NBA and LDBA
- Transition acceptance

## Thank you!

Our theories can be found in the AFP entries `Transition_Systems_and_Automata` and `LTL_Master_Theorem`.