Elixir code generation from TLA+ specifications

Gabriela Mafra

September 26, 2021

/ 1 /

Introduction

1.1 | My background

- Computer Science Bachelor
  - I got interested in Functional Programming really early and did some research in Type Systems
- Working full-time with web development (and lucky enough to end up building some interesting distributed systems)
- Learned about TLA+ in 2018

1.2 | Some context on this research

This is academic research

- I came up with the idea and prototyped it for my bachelor's thesis
- It's on hold

/ 2 /

When does generating code make sense?

TLA+ is a very good language to model transition systems

There are a number of ways to use and to reason about transition systems

Just as PlusCal translates better to imperative code, I want to show how well TLA+ can translate to functional code

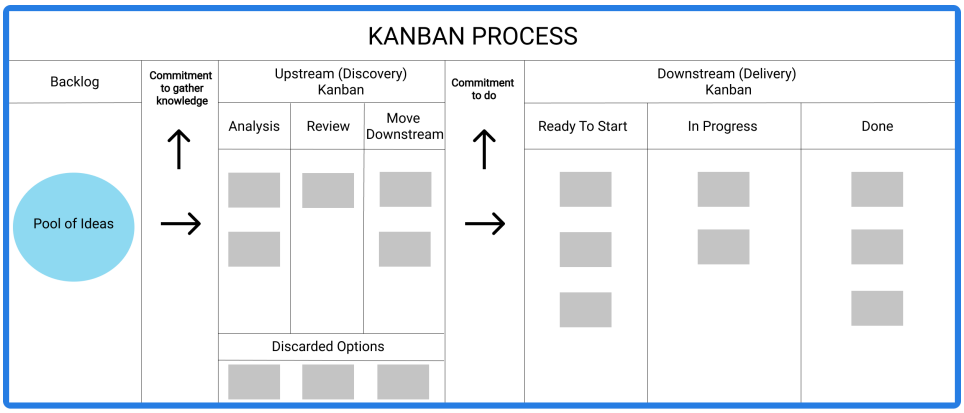
Hypotheses: Seeing the generated code makes it easier for people new to TLA+ to understand what the definitions mean.

Example: Pump station case study

/ 3 /

Where does it fit?

The standard development workflow is divided between upstream and downstream, with some sort of technical discovery that needs to be finished before actual implementation.



We use to make proofs of concept at this stage that are discarded after validation.

/ 4 /

Why Elixir?

1. Reliable concurrency and inter-process communication from Erlang's virtual machine (BEAM)
2. Functional Programming
3. Focused on legibility and maintainability, borrowing a lot from Ruby's syntax

/ 5 /

Transition system, non-determinism

Non-determinism in transition systems happen when more than one transition is defined from the same state.

Non-determinism in computer programs can have many different sources.

How to translate a specification's non-determinism to something executable? My first ideas where spawn processes, choose randomly.

5.1 | Oracle

The oracle is an abstraction that allows the programmer to define how non-determinism scenarios are computed.

It is a separate process that is called by the main process when it reaches a state where more than one transition is possible. The message received by the oracle is a list of possible actions to be taken, and it should respond choosing each action to take.

The action choice can be done by reading some input, by generating some random value, or any other deterministic or non-deterministic computation.

/ 6 /

Translation

6.1 | Actions as functions

```
-------------------------- MODULE TrafficSemaphore --------------------------
VARIABLE color

TurnRed == color = "yellow" /\ color' = "red"
TurnYellow == color = "green" /\ color' = "yellow"
TurnGreen == color = "red" /\ color' = "green"

Init == color \in {"red", "yellow", "green"}

Next == TurnRed \/ TurnYellow \/ TurnGreen
===============================================================================
```

Each action definition is translated into two functions

```
def turn_red_condition(variables) do
  variables[:color] == "yellow"
end

def turn_red(variables) do
  %{ color: "red" }
end
```

6.2 | Detecting non-determinism

Operators such as \/ (or) and ∃(existential quantifier) can introduce non-determinism. However:

```
Next == TurnRed \/ TurnYellow \/ TurnGreen
```

is completely deterministic.

So, actions conected by this operators are always translated to an invocations of the decide() function, which:

- Halts if no action's condition is satisfied (deadlock)
- Chooses an action if it is the only action whose condition is satisfied
- Calls the oracle to choose if more than one condition is satisfied

/ 7 / How to test

7.1 | From TLC state exploration graph

White-box testing by calling the main function (translated from the next state formula) repeatedly and checking that all transitions are present on the graph.

7.2 | From TLC-generated counterexamples

Black-box testing by using the oracle to choose transitions that match the counterexample's trace and checking that it is possible.

/ 8 /

Next steps and ideas

- Parse and translate more of TLA+ syntax so there is a better set of examples
- Test generation helps the development as well as code generation helps further validating tests
- [Markus's suggestion] Use a random oracle to simulate transitions and compare the results with PlusPy/TLC Simulator
- Prove that generated code matches the specification

/ 9 /

Thanks!

This work is available at my GitHub

- Code Generation: `https://github.com/gabrielamafra/tla-transmutation`
- Pump-station case study: `https://github.com/gabrielamafra/pump-station`