Practical TLA$^+$

Jan Macháček

March 19, 2018

Abstract

The only way for me to find out what my lively, but chaotic in-laws have planned for the weekend is to phone the grandmother's house to find out who is there; and then call or text the people on the list to find the actual answers. (Grandmother's job is to have litres of tea, kilos of biscuits, and a stash of poitín; this ᴛᴇʟᴇ-p̀ᴏɴᴇ is young people's business!) Some of the in-laws answer immediately with concrete plans; some answer, but haven't made a decision yet; some don't answer calls, but will eventually text back; some have their phones switched-off. The challenge is to contact everyone on the list within 10 minutes; or else go back to the start of the process: they all come & go and change their minds!

An actor system using the Akka toolkit seems like a perfect fit for the implementation: one actor as an aggregate of what all in-laws are doing coordinating an actor-per-in-law to handle the person's plans, and an actor for each communication method: the mobile phone, messenger app, etc. Perfect! The happy-day scenario, where everyone responds in time, will work and will be efficient; but what about the case where there are failed calls, undelivered or unanswered text messages? What if the system receives "will ping you in 5 mins!" an hour after sending out the request SMS, but from an unknown number–all because aunt Hazel got a new phone number, checks the old one, but responds only using the new one?

This paper shows how to use TLA$^+$ to construct a model of the various states the system will be in, verify the invariants that should hold in each state; but also discover traces where this model fails. Once the model is verified, it can be turned to Akka and Scala code.

1 Lively, but chaotic family system

sasd