What's my type?
Examining how model checking and type systems may complement and help each other

Background

There has been an explosion in where computer systems are found. They are everywhere and they control more and more of our lives, directly or indirectly. A couple of examples of this are power plants, planes, sewage systems, medical devices, and spacecrafts. All of these systems use a computer at some point, usually to control a part of them. The systems given as examples are all critical in the sense that a failure would be catastrophical, potentially resulting in the loss of lives.

For certain critical systems, model checking is a way of guaranteeing that the software works as expected by exhaustively testing every possible state the modelled program or protocol could be in. However, model checking is also slow and often runs into the "State Explosion Problem" [8] where the model to be checked results in many more states than could physically be stored, even on modern hardware. This problem has been, and still is, studied in great detail [2, 3, 4, 5, 6, 7] but the most common solution remains to simplify the actual model into smaller submodels (e.g. [9]) which can result in problems going unnoticed due to it only emerging when the system as a whole is considered.

Type systems allow us to formulate the inputs and outputs of programs in a way that the compiler can then check that we are indeed using the right things. This allows us to be confident that the program we have written is correct with respect to the defined types. A simple example (from [1]) models a door that can be opened, closed, and where the doorbell can only be rung if the door is closed:

```
data DoorState = DoorClosed | DoorOpen

data DoorCmd : Type -> DoorState -> DoorState -> Type
  where
    Open : DoorCmd      () DoorClosed DoorOpen
    Close : DoorCmd     () DoorOpen DoorClosed
    RingBell : DoorCmd  () DoorClosed DoorClosed

    Pure : ty -> DoorCmd ty state state
    (>>=) : DoorCmd a state1 state2 ->
            (a -> DoorCmd b state2 state3) ->
            DoorCmd b state1 state3
```

Given the types and actions described above, a program which uses the types and type-checks must correctly follow the 'protocol' of operating the door. The advantage to using types and type-checking is that useful programs can be modelled and type-checked (proven to be valid) *without* running into the state explosion problem.

However, it is not known whether the types correctly model the problem we are trying to solve. The programs may type-check but if the types themselves do not capture the problem correctly, then that is never caught and the programs are still invalid (despite the type-checker not knowing so).

Proposal

The aim of this research project would be to investigate how formal methods and model checking could complement or assist dependent types and functional programming and vice versa.

As discussed in the background section above, there is no guarantee that the types correctly model the desired problem. Given that model checking can be used for checking that protocols are valid (e.g. that it is possible to reach a certain state under the current protocol), it is possible that model checking could also be used to answer the question "Do my types correctly model my problem?" The door example could be modelled in Spin with the following Promela code:

```
mtype:state = { DoorOpen, DoorClosed };
mtype:action = { Open, Close, RingBell };

chan c = [0] of { mtype:action };

mtype:state doorstate;

init
{
    do
    :: doorstate = DoorOpen; break;
    :: doorstate = DoorClosed; break;
    od
    run SENDER();
    run RECEIVER();
}

proctype SENDER()
{
    do
    :: c!Open
    :: c!Close
    :: c!RingBell
    od
}

proctype RECEIVER()
{
    do
    :: (c?Close && doorstate == DoorOpen) ->
           doorstate = DoorClose;
    :: (c?Open && doorstate == DoorClosed) ->
           doorstate = DoorOpen;
    :: (c?RingBell && doorstate == DoorClosed) ->
           doorstate = DoorClose;
    od
}
```

With this model in Promela, we could then use Linear Temporal Logic and SPIN to examine questions like "Is it always possible to open the door?" or "Can the door infinitely often be in the `DoorOpen` state?"

Initially, an investigation could be conducted into how type checking dependent types and model checking might overlap. For example: for verifying that the index of a fixed-length array does not go out of bounds, how does a dependent type system model and verify the indices, and how would this be done in a model checker like SPIN?

Based on the information obtained from this investigation, a framework for model checking types could be drafted, discussing potential compromises that may need to be made. This framework could then either be implemented as a formal model in an existing modelling language, e.g. PROMELA, or it may prove necessary to build a custom model checker (or customise an existing one).

Overall, the aim of the project would be to explore how model checking dependent types could strengthen (or even prove) that the types capture the scenario the user is trying to model correctly. Additionally, going the other way, the aim would also be to explore how dependent types can help guarantee that the programs written are correct, instead of having to rewrite (often in a significantly simplified form) and check them in a formal modelling language.

References

[1] Brady, E. *Type-driven development with Idris.* OCLC: ocn950958936. Shelter Island, NY: Manning Publications Co, 2017. 453 pp. ISBN: 978-1-61729-302-3.

[2] Clarke, E. M. “Model Checking – My 27-Year Quest to Overcome the State Explosion Problem”. In: (2008), p. 1. DOI: `10.1007/978-3-540-89439-1_13`.

[3] Clarke, E. M. et al. “Model Checking and the State Explosion Problem”. In: *Tools for Practical Software Verification.* Ed. by B. Meyer and M. Nordio. Vol. 7682. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–30. ISBN: 978-3-642-35745-9 978-3-642-35746-6. DOI: `10.1007/978-3-642-35746-6_1`. URL: `http://link.springer.com/10.1007/978-3-642-35746-6_1` (visited on 01/15/2020).

[4] Clarke, E. et al. “Progress on the State Explosion Problem in Model Checking”. In: *Informatics.* Ed. by R. Wilhelm. Red. by G. Goos et al. Vol. 2000. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 176–194. ISBN: 978-3-540-41635-7 978-3-540-44577-7. DOI: `10.1007/3-540-44577-3_12`. URL: `http://link.springer.com/10.1007/3-540-44577-3_12` (visited on 01/15/2020).

[5] Demri, S. et al. “A parametric analysis of the state-explosion problem in model checking”. In: *Journal of Computer and System Sciences* 72.4 (June 2006), pp. 547–575. ISSN: 00220000. DOI: `10.1016/j.jcss.2005.11.003`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S0022000005001297` (visited on 01/15/2020).

[6] Kress-Gazit, H. et al. “Correct, Reactive, High-Level Robot Control”. In: *IEEE Robotics & Automation Magazine* 18.3 (Sept. 2011), pp. 65–74. ISSN: 1070-9932. DOI: `10.1109/MRA.2011.942116`. URL: `http://ieeexplore.ieee.org/document/6016593/` (visited on 01/15/2020).

[7] Stuart, D. et al. “Simulation-verification: biting at the state explosion problem”. In: *IEEE Transactions on Software Engineering* 27.7 (July 2001), pp. 599–617. ISSN: 00985589. DOI: `10.1109/32.935853`. URL: `http://ieeexplore.ieee.org/document/935853/` (visited on 01/15/2020).

[8] Valmari, A. “The state explosion problem”. In: *Lectures on Petri Nets I: Basic Models.* Ed. by W. Reisig and G. Rozenberg. Red. by G. Goos et al. Vol. 1491. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 429–528. ISBN: 978-3-540-65306-6 978-3-540-49442-3. DOI: `10.1007/3-540-65306-6_21`. URL: `http://link.springer.com/10.1007/3-540-65306-6_21` (visited on 01/15/2020).

[9] Yan Sun et al. “Verifying Noninterference in a Cyber-Physical System The Advanced Electric Power Grid”. In: *Seventh International Conference on Quality Software (QSIC 2007).* 2007 7th International Conference on Quality Software. Portland, OR: IEEE, Oct. 2007, pp. 363–369. ISBN: 978-0-7695-3035-2. DOI: `10.1109/QSIC.2007.4385521`. URL: `http://ieeexplore.ieee.org/document/4385521/` (visited on 01/15/2020).