Symbolic Gossip

Me

Saturday 25th May, 2024

Abstract

Contents

1 Introduction

The Gossip Problem is the problem of sharing information in a network. An explicit model checker for Gossip called GoMoChe exists [Gat23]. Explicit model checkers are generally less efficient than symbolic ones, and GoMoChe too is computationally limited to small examples. On the other hand, SMCDEL is a symbolic model checker for dynamic epistemic logic and contains symbolic representations for various logics.

The goal of this project is to implement a symbolic model checker for gossip models. Ultimately, this model checker could replace GoMoChe's explicit model checking functionality and extend the SMCDEL library.

Many variants of the gossip problem exist, each with their own computational challenges. Most notably, there is a distinction between synchronous and asynchronous gossip models, where synchronicity means the presence of a global clock tick with each call made. Due to both the mathematical and computational complexity of the asynchronous models, this project focuses mainly on the standard example in a synchronous setting. In order to achieve this, we start by developing a symbolic representation capable of handling so-called transparent models, where each call is visible to all agents.

2 Method

We will first extend the SMCDEL project with a knowledge structure for (transparent) Gossip models and a call free language. Initially, this approach aims to reuse as many of the existing SMCDEL modules as possible.

After providing a minimal viable product in this way, we will aim to expand the knowledge structure of the symbolic model checker in order to enable synchronous models. Additionally, a formula rewriting procedure based on the call reduction axioms in [VDVDHK20] might enable evaluation of formulae in the full language.

Based on the performance of the symbolic model checker at each stage, it may be desirable to improve its performance by implementing more specific implementations that leverage characteristics of the gossip models.

3 Wrapping it up in an exectuable

We will now use the library in a program.

```
module Main where

import Basics

main :: IO ()
main = do
  putStrLn "Hello!"
  print somenumbers
```

```
  print (map funnyfunction somenumbers)
  myrandomnumbers <- randomnumbers
  print myrandomnumbers
  print (map funnyfunction myrandomnumbers)
  putStrLn "GoodBye"
```

We can run this program with the commands:

```
stack build
stack exec myprogram
```

The output of the program is something like this:

```
Hello!
[1,2,3,4,5,6,7,8,9,10]
[100,100,300,300,500,500,700,700,900,900]
[1,3,0,1,1,2,8,0,6,4]
[100,300,42,100,100,100,700,42,500,300]
GoodBye
```

4 Simple Tests

We will discuss the tests below.

```
module Main where

import Test.Hspec

import qualified SimpleTransformerSpec
import qualified ClassicTransformerSpec
```

We test the implementations using hspec. We verify that the

```
main :: IO ()
main = hspec $ do
    describe "SimpleTransformer" SimpleTransformerSpec.spec
    describe "ClassicTransformer" ClassicTransformerSpec.spec
```

To run the tests, use `stack test`.

To also find out which part of your program is actually used for these tests, run `stack clean && stack test`. Then look for "The coverage report for ... is available at ... .html" and open this file in your browser. See also: `https://wiki.haskell.org/Haskell_program_coverage`.

```
module SimpleTransformerSpec where

import SimpleTransformer

import Test.Hspec hiding ( after )
import SMCDEL.Examples.GossipS5
import SMCDEL.Language
import HaitianS5
```

We test the implementation of the Simple Transformer with the following tests.

```
spec :: Spec
spec = do
    describe "SimpleTransformer" $ do
        it "after same result as individual calls" $ do
            afterSimple 3 [(0,1),(1,2)] `shouldBe` doSimpleCall (doSimpleCall (
                simpleGossipInit 3) (0,1)) (1,2)
        it "secret was exchanged after 1 call" $ do
            eval (afterSimple 3 [(0,1),(1,2)]) (K "0" $ has 3 0 1) `shouldBe` True
        it "secret learnt in first was exchanged in second call" $ do
            eval (afterSimple 3 [(0,1),(1,2)]) (K "2" $ has 3 2 0) `shouldBe` True
        it "SmpTrf: higher-order knowledge fails" $ do
            eval (afterSimple 3 [(0,1),(1,2)]) (K "2" $ has 3 0 1) `shouldBe` False
```

Note in particular the test `SmpTrf: higher-order knowledge fails`, which returns false. However, the tested formula $K_2 S_0 1$ should be true after calls $01; 12$: after the second call, agent 2 should be able to infer that the prior call was between agents 0 and 1 and conclude that their secrets were exchanged.

```
module ClassicTransformerSpec where


import Test.Hspec hiding ( after )
import SMCDEL.Examples.GossipS5
import SMCDEL.Language
import SMCDEL.Symbolic.S5
```

We can verify that $K_2 S_0 1$ is indeed true after the calls $01; 12$ in the classic transformer.

```
spec :: Spec
spec = do
    describe "ClassicTransformer" $ do
        it "clsTrf: higher-order knowledge works" $ do
            eval (after 3 [(0,1),(1,2)]) (K "2" $ has 3 0 1) `shouldBe` True
```

Indeed this test passes, highlighting the limitations of our earlier simple transformer.

5 Conclusion

ADD CONCLUSION

References

[Gat23] Malvin Gattinger. Gomoche-gossip model checking. *Branch async from*, 1, 2023.

[VDVDHK20] Hans Van Ditmarsch, Wiebe Van Der Hoek, and Louwe B. Kuijer. The logic of gossiping. *Artificial Intelligence*, 286:103306, September 2020.