Symbolic Gossip

Joris Galema
Madeleine Gignoux
Djanira Gomes
Wouter Smit

Saturday 25th May, 2024

Abstract

Contents

1 Introduction

The *Gossip Problem* is the problem of sharing information in a network. Many variants of the gossip problem exist, each with their own computational challenges. Most notably, a distinction is made between the situation where agents know which agents share information and when, known as the *transparent* Gossip Problem and when agents only know that information is shared, known as the *synchronous* Gossip Problem.

For simulating the Gossip Problem, an explicit model checker for Gossip called GoMoChe exists [?]. Explicit model checkers are generally less efficient than symbolic ones, which aim to cut down on computation time. GoMoChe too is therefore computationally limited to small examples. On the other hand, a symbolic model checker for dynamic epistemic logic called SMCDEL exists, and contains symbolic representations for various logics, including the *synchronous* Gossip Problem (CITE). This tool is much more general, and can be used for many logic problems and puzzles, however in terms of the Gossip Problem, only a symbolic model checker for the synchonous gossip problem exists, and due to keeping track of the higher-order logic and distinctions between calling and secrets, this implementation can take a simple model and blow it up to check simple formulas.

This project expands on SMCDEL's functionality. To begin with, in Section X we start by implementing some functions for interpretting the current state which makes Gossip Problems in SMCDEL, as well as our future work, easier to interpret. Next, in Section Y we use the definition of knowledge transformers provided in SMCDEL to create knowledge transformers for the *transparent* Gossip Problem.

Finally, in Section Z we make a simpler knowledge transformer, which cuts down on the complexity of computing the synchronous gossip problem, with the tradeoff of losing higher-order knowledge.

– explain – transparent – simple

2 Background

3 Wrapping it up in an exectuable

We will now use the library in a program.

```haskell
module Main where

main :: IO ()
main = undefined
```

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