Protocol Synthesis in the Dynamic Gossip Problem

Leo Poulson

April 10, 2019

1 Introduction

1.1 Motivation

1.1.1 The Gossip Problem

The gossip problem is a problem regarding peer-to-peer information sharing. A set of agents start out with some secret information, and their goal is to transfer this information across the whole network, thus finding out the secret of all of the other agents. This information is transferred through a phone call from one agent to another. When two agents communicate they tell each other all the secrets that they know; hence the name *gossip*.

In the original problem formulation (Tijdeman 1971), each agent began with the phone number of every other agent, and hence the starting network is a complete graph. Recently, there has been a lot of work in studying a variation of this problem named the *dynamic* gossip problem. In this, the agents start out with an incomplete set of phone numbers, and as they call the other agents, they exchange all of the phone numbers they know and so the graph's edges grow in number.

The dynamic gossip problem is a popular research topic (van Ditmarsch et al. 2018, van Ditmarsch et al. 2016), with applications in work in epidemics and research discoverd (Haeupler et al. 2016) **Add more references.**. In such scenarios, our network may be some peer to peer network, with our agents as computers and the goal to be finding the IP addresses of all of the other computers; or our network might be a social network, where we have people as agents who want to connect with everyone around them. In both cases, our network is updated as the agents communicate and they learn the contact details of other agents.

Applications of DEL here . . .

There are several existing implementations of model checkers for epistemic logic and gossip problems; three of them are van Eijck 2014, Gattinger 2018a, Gattinger 2018c. The three all differ in implementation style, but they all aim to solve the same problem; given some system with an initial state and a way to move between states, what events can occur to take the system from its initial state to a successful one? Gattinger 2018a aims to give the user the events required to take the system to the state, whereas the other two aim to tell the user if a string of events given are successful.

1.1.2 Planning

Automated planning is the act of computing a finite sequence of actions to take some system from a given initial state to some successful state. Recently, a variant of this has been studied by the dynamic epistemic logic community, called *epistemic* planning. Given an initial set of knowledge states of a set of agents, a finite set of available events, and an epistemic objective, the epistemic planning problem consists of computing the finite sequence of events whose occurence takes us from the initial state to a state satisfying the epistemic objective.

This problem is undecidable for many cases (Aucher and Bolander 2013), but by placing certain restrictions on the models we use it becomes decidable.

1.1.3 Model Checking

1.2 Contributions

In this thesis, we put forward a process to solve the decidable epistemic planning problem. We also give an implementation of this process in Haskell. This process also will solve the gossip problem, by finding a sequence of calls to make every agent know the secret of every other agent; however it is also capable of finding sequences of calls that lead to a situation in which every agent knows that every agent is a secret. **TODO: Find applications for this**.

Given that our software solves just the epistemic planning problem and not strictly the gossip problem, it can plan for any appropriate models. There is no other software currently that solves this problem.

2 Background

2.1 Dynamic Epistemic Logic

2.1.1 Epistemic Logic

Epistemic Logic is the logical language that we use to reason about knowledge. It is a modal logic; a logic with some operator, that can be used to model the passage of time, knowledge, obligation, or any other modality. The essential reference on epistemic logic is Fagin et al. 1995, and it is from here that most of the information in this section comes.

If we have some set of agents A and some set of propositions Λ, then we define the language of epistemic logic over this set of propositions, $\mathcal{L}(\Lambda)$, with the following BNF;

$$\phi ::= \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid K_i\phi$$

where $p \in \Lambda$ and $i \in A$. We also can define false, conjunction and implication in the classic way. We read $K_i\phi$ as "agent i knows that ϕ is true". We can also define the dual to K, in the classic way as $\widehat{K}_i\phi ::= \neg K_i \neg\phi$. We read this as "agent i considers it possible that ϕ is true". We can give our epistemic logic a semantics through use of *Kripke models*.

A Kripke model $\mathcal{M}$ over a set of agents I and a set of propositions Λ is a triple (W, R, V), where W is a **(maybe)** finite set of worlds, R is a set of binary relations over W indexed by an agent, such that $R_i \subseteq W \times W$, and $V : W \to \mathcal{P}(\Lambda)$ is a valuation function that associates to every world in W some set of propositions that are true at it.

For epistemic logic, we think of R_i as being the set of pairs of worlds that an agent i cannot distinguish between, and thus considers possible. In our semantics, we use this relation to define knowledge in terms of possibility; agent i knows that something is true if it's true at all of the worlds that i cannot distinguish between.

When giving a semantics to a formula on a Kripke model, we need to use a *pointed Kriple model*. This is just a pair $(\mathcal{M}, w)$ where w is a world of $\mathcal{M}$. Then we read $(\mathcal{M}, w) \models \phi$ as "w satisfies ϕ". We define the evaluation as follows:

$(\mathcal{M}, w) \models \top$
$(\mathcal{M}, w) \models p$ iff $p \in V(w)$
$(\mathcal{M}, w) \models \neg\phi$ iff $(\mathcal{M}, w) \not\models \phi$
$(\mathcal{M}, w) \models \phi \wedge \psi$ iff $(\mathcal{M}, w) \models \phi$ and $(\mathcal{M}, w) \models \psi$
$(\mathcal{M}, w) \models K_i\phi$ iff for all v such that $(w, v) \in R_i, (\mathcal{M}, v) \models \phi$

We need to give some properties to our knowledge operator in order to better understand it. We make all of our relations R_i equivalence relations. This means three things;

- R_i is *reflexive*; for all $w \in W$, $(w, w) \in R_i$.
- R_i is *symmetric*; for all $w, v \in W, (w, v) \in R_i$ iff $(v, w) \in R_i$.
- R_i is *transitive*; for all $w, v, u \in W$ if $(w, v) \in R_i$ and $(v, u) \in R_i$, then $(w, u) \in R_i$.

This is done in order to convey that agent i considers world v possible from world w if in both w and v agent i has the same information; that is, they are indistinguishable to the agent.

It is identical to say that our relations R_i are equivalence relations, as it is to say that our model is an S5 model. This is defined as a model in which the modal operator K obeys the following axioms:

- K: $K(\phi \to \psi) \to (K\phi \to K\psi)$
- T: $K\phi \to \phi$
- 5: $\widehat{K}\phi \to K\widehat{K}\phi$

These axioms hold independent of which agent's knowledge we are reasoning about.

2.1.2 Event Models

Public announcement logic was the first development in epistemic logic to support information change in epistemic logic, and is described in Plaza 2007. However, in public annoucement logic, we may only model truthful, public communications; but we all know that there are many other types of communication and events after which some information has changed. An agent may communicate with another agent in a private phone call, or may be lying to them; furthermore, we may have some *physical* action, like a coin flip occuring, after which some knowledge has changed.

To model these more complicated events, we use *event models*. These treat events in a very similar way to how Kripke models treat worlds; we think of a set of possible events that can occur, and encode the events that an agent can tell apart. We give the modern definition of event models, as in Gattinger 2018b, and Aucher, Maubert, and Pinchinat 2014, however these were first defined in **Cite 1998 Baltag paper**

Formally, an event model $\mathcal{E}$ is a tuple $(E, Q, \mathsf{pre}, \mathsf{post})$. E is a finite set of events; Q is a set of relations Q_i for each $i \in Ag$, such that $Q_i \subseteq E \times E$. As before, we make all relations Q equivalence relations. $\mathsf{pre} : E \to \mathcal{L}(\Lambda)$ is the *precondition function*; given an event $e \in E$, it returns to us a formula that has to be true in order for e to occur. $\mathsf{post} : E \times \Lambda \to \mathcal{L}(\Lambda)$ is the *postcondition function*; given an event $e \in E$ and a proposition $p \in \Lambda$, it returns to us some formula that had to be true at the prior state s in order for p to be true after event e occurs at s. We will later give examples for these two functions.

Updating a Kripke model $\mathcal{M}$ with an event model $\mathcal{E}$ is gives us another Kripke model $\mathcal{M} \times \mathcal{E} ::= (W', R', V')$, where:

$$
\begin{aligned}
W' &::= \{(w, e) \in W \times E \mid (\mathcal{M}, w) \models \mathsf{pre}(e)\} \\
R'_i &::= \{((w, e), (v, f)) \mid (w, v) \in R_i, (e, f) \in Q_i\} \\
V'(w, e) &::= \{p \in \Lambda \mid (\mathcal{M}, w) \models \mathsf{post}(e, p)\}
\end{aligned}
$$

We see that it is the postcondition function that allows for factual change; that is, the update of what is true at a state given the occurence of some event.

2.2 The Gossip Problem

Gossip is a procedure for spreading secrets around a group of agents, where the agents are commonly displayed as nodes in a graph and the ability of one agent to contact another displayed as an edge between two nodes. Gossip was first put forward in Tijdeman 1971, where the network is a complete graph; all agents can contact one another. One question here was to find how many calls are needed for every agent to learn the secret of every other agent. We will henceforth describe an agent knowing the secret of every other agent as this agent being an *expert*. It was quickly proved (A. Hajnal and Szemere 1972, Baker and Shostak 1972) that this number, for a network where the number of agents n is greater than 4, is $2n - 4$.

2.2.1 Dynamic Gossip

Dynamic Gossip is a variant of the classical gossip problem, in which we start off with an incomplete graph, representing the fact that the agents have only the phone numbers of some of the other agents. In this case, when two agents talk on the phone, they also exchange all of the phone numbers that they know as well as the secrets that they know.

2.2.2 Protocols

The gossip problem, as we have mentioned so far, can rely on some central scheduler to tell the agents what call to make, and lets them know when to stop. However in distributed computing, methods that do not require this central authority are desirable. In such a situation, the agents need to have some form of rules to follow to decide how to behave and who to call next. This is the motivation for a *gossip protocol*, which are short conditions that must be fulfilled in order for an agent to make a specific call. These protocols were first proposed in R. Apt, Grossi, and Hoek 2016, Attamah et al. 2014, and some exemplar ones are:

- ANY - If x knows the phone number of y, x may call y.
- LNS - If x knows the phone number of y and x does not know the secret of y, then x may call y.

In this thesis, we do not study the distributed gossip problem; rather, a version with a central authority that surveys the network topology and then decides which agent is allowed to make a call, and also who they will call. However, these protocols do still have use for us. ANY allows for infinite call sequences - for example, one where agent a just repeatedly calls b, whereas all of the call sequences induced by LNS are of finite length. In the long run this does not really matter; both ANY and LNS have runtime expected execution length in $O(n \log n)$ (van Ditmarsch et al. 2018), yet in our implementation tests performed with LNS took considerably less time than ANY. For this pragmatic reason LNS is often used in this thesis.

It should also be noted that there are certain classes of graphs for which LNS cannot induce a successful call sequence, yet ANY can (van Ditmarsch et al. 2018). However, this thesis does not investigate this topic, and as such this will no longer be mentioned.

2.2.3 Formalisation

We now go on to formalise some of the ideas mentioned so far in this section. The definitions of gossip graphs are classic and can be found in all of the related literature (e.g. van Ditmarsch et al. 2018, Gattinger 2018b)

We formally denote a gossip graph $\mathcal{G}$ with a triple (A, N, S). A is a finite set of agents in the graph; $N \subseteq A \times A$ is a set of ordered pairs of agents such that $(u, v) \in N$ (or Nuv) iff u knows the phone number of v. $S \subseteq A \times A$ is a set of ordered pairs of agnets such that $(u, v) \in S$ (or Suv) iff u knows the *secret* of v.

We have that for all gossip graphs, for any agent a, $(a, a) \in N$ and $(a, a) \in S$, expressing that any agent always knows their own phone number and secret.

2.3 Planning

Automated planning is the process of computing which set of events must occur to take a system from some given initial state to some successful state. Such a system is defined as a triple $\Sigma = (S, A, \gamma)$, where:

- S is some set of states;
- A is some set of *actions*;
- $\gamma : S \times A \hookrightarrow S$ is a state-transition function. It is partial; for $s \in S, a \in A$, either $\gamma(s, a) \in S$ or it is undefined.

Then an instance of the planning problem is a triple (Σ, s_i, S_g), where;

- Σ is a planning system;
- $s_i \in S$ is some initial state;
- $S_g \subseteq S$ is some set of goal states.

A *solution* to the planning problem is some ordered set of actions $a_0, a_1, \ldots, a_n$ such that $\gamma(\gamma(\ldots\gamma(s_0, a_0)\ldots, a_{n-1}), a_n) \in S_g$.

Note that we call a planning problem *multi-agent* if the number of agents in the system is greater than 1, and *single-agent* if the number of agents is equal to 1.

2.3.1 Epistemic Planning

The dynamic epistemic logic community has recently been investigating a special case of planning, namely *epistemic* planning (Bolander and Birkegaard 2011, Aucher and Bolander 2013). This is planning where we may have some initial epistemic state (e.g. agent a knows ϕ), some actions that update these epistemic states, and some accepting epistemic state (e.g. agent b knows ψ). It is not hard to see how this can be applied to the gossip problem.

We give a formal definition of the epistemic planning problem that slightly differs from convention, however with this definition and those from the literature are equivalent. Then the epistemic planning problem is as follows; given a pointed epistemic model $(\mathcal{M}, w)$ and an event model $\mathcal{E}$, and some goal formula ϕ, find some set of events $e_1, e_2, \ldots, e_n$ such that $(\mathcal{M}, w) \otimes (\mathcal{E}, e_1) \otimes \ldots \otimes (\mathcal{E}, e_n) \models \phi$. The *propositional epistemic planning problem* is the restriction

of this to event models with pre- and post-condition functions whose codomains are in the propositional fragment of $\mathcal{L}$; that is, they do not contain the modal operator K. These event models are what we call propositional event models.

It is proven in Bolander and Birkegaard 2011 that the multi-agent epistemic planning problem is undecidable for an aribtrary system. However, placing certain restrictions on the system used yields decidability (Yu, Wen, and Liu 2013, Aucher, Maubert, and Pinchinat 2014). One particularly important result is that the propositional epistemic planning problem is decidable; indeed, it is this result that this thesis relies on.

In this thesis, we focus on solving the epistemic planning problem by means of the automata constructions described in Aucher, Maubert, and Pinchinat 2014. These methods, and the resulting piece of software associated with this thesis, are unique in that they will work for any epistemic model and (propositional) event model. This unfortunately means that we cannot plan for situations like muddy children or drinking logicians (van Eijck 2014) as they require epistemic pre- or post-conditions.

2.4 Existing Tools

We now give an overview of existing software in the realm of model checking or planning for epistemic systems.

2.4.1 DEMO-S5

DEMO-S5 (van Eijck 2014) is a model checker for epistemic models, limited to $S5$ models. It is also limited in that it only supports events that are either public announcements (e.g. an agent telling every other agent some statement) or publicly observable factual change (e.g. the flip of a coin, where the result is seen by every agent).

It performs the task of model checking by taking an epistemic model and a formula representing either the announcement or the fact that changed, and updating the model with the event represented by the formula. The definition of the update used is very similar to the update of an epistemic model with an event model as defined in Section 2.1.2. The given events are applied, and at this point we check if we're in a successful state or not.

Unfortunately we cannot use this to model check a gossip problem, as events in the gossip problem are neither of these; they are private one-to-one communications. Furthermore, the software is incapable of planning; simply checking. Despite this, the software was incredibly useful in our project in order to aid understanding of epistemic and event models. Indeed, the formalisation of epistemic and event models demonstrated here was used in the code associated with this thesis.

2.4.2 Gossip

Gattinger 2018a is another model checker, however now for strictly the gossip problem. This program has a significant advantage over the other two pieces of software; namely that it is capable of planning. It does this by generating all of the possible sequences of calls on the

graph and then checking which of these are successful (i.e., after execution of all of the calls the graph is in some successful state) and which are not. It also has further capabilities for studying protocols deeply, however these are not relevant to our thesis.

Although the methods used within this software are very different to the ones used in our implementation, this system proved to be very useful for testing purposes, due to its simplicity of use. The results returned by our software during testing were verified using this system. Further elaboration on this will be given in the evaluation section of this thesis.

One limitation of this system is that is may only handle gossip states, and not generic epistemic models. Also limiting is that it enumerates all possible call sequences from the initial state; we are only interested in successful ones. This means that we have the possibility of lots of wasted computation if we need to validate lots of unsuccessful paths before we get to a successful one.

2.4.3 SMCDEL

SMCDEL is the most sophisticated of the three pieces of software. A technical report is given in Gattinger 2018c, whilst a lot of the underlying theory is in Gattinger 2018b. The main difference is that it uses *symbolic model checking*, a method put forward in Burch et al. 1992. This gives a much more efficient implementation **why**?, as can be seen in the Chapter 4 of Gattinger 2018b.

This software is capable of efficiently model checking all classes of epistemic models, and also has capacity for planning; however this side of the program is not well-displayed, and it is unclear precisely how to use it. Furthermore, due to a reasonably unpleasant interface we choose not to use this software for benchmarking and instead use the simpler Gattinger 2018a.

2.5 Uniform Strategies

In this thesis, we use a lot of ideas from Aucher, Maubert, and Pinchinat 2014. This paper puts forward a method for epistemic planning, through use of automata representing states of the model and events in the event model as the states and transitions respectively. This then references a process in Maubert and Pinchinat 2014 for the creation of an automata which has a set of the other indistinguishable states baked into the states of the automata. This is a very useful thing for our context, as it means that we can evaluate formulas of the form $K_a\phi$ *positionally*; that is, we need only to look at the state to be able to evaluate the formula. This is in contrast to the alternative, which is to check our current state then compute the possible call sequences that are indistinguishable from our current one, and then go and check there too, and so on.

The process in Maubert and Pinchinat 2014 is very verbose and is beyond the scope of this thesis; however, in order to illustrate the fact that our algorithm is a special case of the process, we give a brief introduction to the process and its theory. We slightly change some of the definitions to lend itself better to our scenario. The setting is game theory; hence the frequent references to plays and strategies.

2.5.1 Game Arenas

A game with *imperfect information* is a game in which the players have some uncertainty concerning hte current configuration of the game (for example Poker; a player does not know which cards are in the other players' hands). A player in such a game cannot plan to play differently in a situation she cannot distinguish; hence we have to define her strategy *uniformly* across situations she cannot distinguish.

We consider two-player turn-based games played on a some graph, where the vertices are labelled with propositions. These propositions hold some useful information about the state; it could be what people know about it, or some fact about the environment the player is in and so on. This set of propositions will be referred to as AP.

Then a game arena is a structure $\mathcal{G} = (V, E, v_I, l)$ where $V = V_1 \uplus V_2$ [1] is a set of positions split between those of Player 1 (V_1) and those of Player 2 (V_2). $E \subseteq (V_1 \times V_2) \cup (V_2 \times V_1)$ is the set of edges, $v_I \in V$ is the initial position and $l : V \to \mathcal{P}(AP)$ is a valuation function, which maps each position to the finite set of propositions that are true at it.

2.5.2 Transducers

We give a quick recap of transducers, as they are used heavily in the next two sections. A transducer is like a nondeterministic finite automaton with two tapes; an *input* tape and an *output* tape. The transducer reads an input finite wood on its input and writes a finite word on its output tape. Hence a transducer defines a binary relation. Relations recognised by a transducer are called *regular* or *rational* relations.

A transducer is a 6-tuple $T = (Q, \Sigma, \Gamma, I, F, \Delta)$, where Q is a finite set of states, Σ is the input alphabet, Γ is the output alphabet, $I \subseteq Q$ is a finite set of initial states, $F \subseteq Q$ is a finite set of final states, and $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times Q$ is the transition function.

$(q, a, b, q') \in \Delta$ means that the transducer can move from state q to state q' by reading a from the input tape and writing b to the output tape. Then we can define the extended transition relation Δ^* as the smallest relation satisfying:

- for all $q \in Q$, $(q, \epsilon, \epsilon, q) \in \Delta^*$
- if $(q, w, w', q') \in \Delta^*$ and $(q', a, b, q'') \in \Delta$, then $(q, w.a, w'.b, q'') \in \Delta^*$

We can see Δ^* as the transitive closure of Δ. We abbreviate $(q, w, w', q') \in \Delta^*$ to $q \rightsquigarrow_{w|w'} q'$ in the rest of this section.

Finally, we say that the relation recognised by some transducer T is

$$[T] ::= \left\{(w, w') \mid w \in \Sigma^*, w' \in \Gamma^*, \exists q_F \in F, \exists q_i \in I, q_i \rightsquigarrow_{w|w'} q_F\right\}$$

2.5.3 The Powerset Arena

In games with imperfect information, the information set of a player after a certain play is the set of positions she may possibly be in, consistent with what she has observed[2]. We can define

[1] We use $\uplus$ here to indicate that this is a partition

[2] For instance in a game of poker with one deck of cards, a player may have no idea what the other players have in their hands except that they know they don't have the cards the player has in their own hands, nor the cards face-up on the table.

a similar notion in our setting, and we show that this is sufficient to build a powerset arena in which formulas of the form $K_a\phi$, where ϕ is propositional, can be evaluated positionally.

For an arena $\mathcal{G}$ and a transducer T over finite sequences of events $e_i \in E$, we construct this automata $\widehat{\mathcal{G}}$ in which we can evaluate formulas of the form $K_i\phi$...

Our new information set after a move occurs cannot be computed knowing only the previous information set and the new position; we need to simulate the nondeterministic execution of T, taking as input the sequence of positions played and writing as output the related plays. Precisely, we need two things; the set of states the transducer may be in after reading the sequence of positions played so far, and for each of these states the set of possible last positions written on the output tape. We need only remember the last letter and not the whole tape because the information set we aim to compute is just the set of the last positions of related plays.

Then our positions are of the form $(v, S, Last)$, where $v \in V$, $S \subseteq Q$, where Q is the set of states in the transducer and S is the set of possible current states of T, and $Last : S -> \mathcal{P}(V)$[3] associates to a state $q \in S$ the set of the possible last positions on the output tape fo T if the current state is q. The transitions in this arena follow those in $\mathcal{G}$, except we now maintain the additional information about the configuration of the transducer.

To construct the initial position $\widehat{v_I} = (v_I, S_I, Last_I) \in \widehat{\mathcal{G}}$, we need to simulate the execution of T starting from its initial state and reading v_I. To do this, we introduce an artificial position $\widehat{v_{-1}} = (v_{-1}, S_{-1}, Last_{-1})$, where $v_{-1} \notin V$ is some fresh position, $S_{-1} = \{q_i\}$ where q_i is the initial state of the transducer because before starting the transducer is in its initial state, and $Last_{-1}(q_i) = \emptyset$ because we have nothing written on the output tape.

We now come onto defining $\mathcal{G}$. Let $\mathcal{G} = (V, E, v_I, l)$ be an arena and $T = (Q, V, q_i, Q_F, \Delta)$ be an FST. Then we define the arena $\widehat{\mathcal{G}} = (\widehat{V}, \widehat{E}, \widehat{v_I}, \widehat{l})$ as:

- $\widehat{V} = V \times \mathcal{P}(Q) \times (Q \to \mathcal{P}(V))$
- $(u, S, Last)\widehat{\to}(v, S', Last')$ if
  - $u = v_{-1}$ and $v = v_I$, or $u \to v$,
  - $S' = \{q' \mid \exists q \in S, \exists \lambda' \in V^*, q \rightsquigarrow_{v|\lambda'} q'\}$ and
  - $Last'(q') = \{v' \mid \exists q \in S, \exists \lambda' \in V^*, q \rightsquigarrow_{v|\lambda' \circ v'} q',$ or $q \rightsquigarrow_{v|\epsilon} q'$ and $v' \in Last(q)\}$
- $\widehat{v_I}$ is the only $\widehat{v} \in \widehat{V}$ such that $\widehat{v_{-1}}\widehat{\to}\widehat{v}$.
- $\widehat{l}(\widehat{v}) = l(v)$ if $\widehat{v} = (v, S, Last)$.

The definition of transitons is quite complicated, and as such we give a high-level explanation of it. Regarding the definition of the transitions, the first point means that our transitions in $\widehat{E}$ follow those in E, except for the transition leaving $\widehat{v_{-1}}$, which we use to define $\widehat{v_I}$.

The second point expresses that when we move from u to v in $\mathcal{G}$, we give v as input to the transducer. Then the set of states that we can be in, that is, S', is the set of states that can be reached from some previous possible state in $q \in S$ by reading v on the input tape and outputting some sequence λ'.

Finally, the third point expresses that if some position v' is at the end of the output tape after the transducer reads v and reaches q, then it's either because while reading v the last

[3] It may be more helpful to think of this as something a bit more tangible than a function, e.g. a list of tuples $(S, [P])$

letter it wrote is v', or it wrote nothing and v' was already at the end of the output tape before reading v.

Finally, we say that $\widehat{v}_I$ is the only successor of $\widehat{v}_{-1}$, and that the valuation of a position in the powerset arena is the valuation of the underlying position in the original arena.

2.5.4 Lifting Transducers

We can keep on repeating this process, creating a power-powerset arena $\widehat{\widehat{G}}$. However we first need to lift our transducer T where $[T] \subseteq Plays_* \times Plays_*$ to a transducer $\widehat{T}$ where $[\widehat{T}] \subseteq \widehat{Plays_*} \times \widehat{Plays_*}$.

We write $T\downarrow$ for the transducer that computers the bijective function f that maps a play $\widehat{p} \in \widehat{Plays_*}$ to the underlying play $p \in Player_*$, and $T\uparrow$ for the deterministic transducer that computes f^{-1}. Both can be easily constructed from our powerset arena $\widehat{G}$.

Then the lift of a transducer T is $\widehat{T} = T\downarrow \circ T \circ T\uparrow$.

Once we have the transducer $\widehat{T}$, we can repeat the process in Section 2.5.3 with $\widehat{G}$ to get $\widehat{\widehat{G}}$, and so on ad infinitum. $\widehat{\widehat{G}}$ would allow us to positionally check formulae of the form $K_iK_j\phi$.

3 Algorithm

We now give a description of the algorithm we put forward in order to plan for epistemic models.

TODO: Add some more to this.

3.1 $\mathcal{ME}^*$

Bibliography

A. Hajnal, E. C. Milner and E. Szemere (1972). “A cure for the telephone disease”. In: *Canadian Math Bulletin*.

Attamah, M et al. (2014). “Knowledge and gossip”. In: *Frontiers in Artificial Intelligence and Applications* 263, pp. 21–26. DOI: 10.3233/978-1-61499-419-0-21.

Aucher, Guillaume and Thomas Bolander (2013). “Undecidability in Epistemic Planning”. In: *IJCAI - International Joint Conference in Artificial Intelligence*.

Aucher, Guillaume, Bastien Maubert, and Sophie Pinchinat (2014). “Automata Techniques for Epistemic Protocol Synthesis”. In: *Proceedings of the 2nd International Workshop on Strategic Reasoning*.

Baker, Brenda and Robert Shostak (1972). “Gossips and Telephones”. In: *Discrete Mathematics*.

Bolander, Thomas and Mikkel Birkegaard (2011). “Epistemic planning for single- and multi-agent systems”. In: *Journal of Applied Non-Classical Logics*.

Burch, J. R. et al. (1992). “Symbolic Model Checking: 10^20 states and beyond”. In: *Information and Computation*.

Fagin, Ronald et al. (1995). *Reasoning about Knowledge*. MIT Press.

Gattinger, Malvin (2018a). *Explicit Epistemic Model Checking for Dynamic Gossip*. Online; accessed 7-April-2019. URL: https://github.com/m4lvin/gossip.

— (2018b). “New Directions in Model Checking Dynamic Epistemic Logic”. PhD thesis. Institute for Logic, Language and Computation, Universiteit van Amsterdam.

— (2018c). *SMCDEL — An Implementation of Symbolic Model Checking for Dynamic Epistemic Logic with Binary Decision Diagrams*. Tech. rep. Institute for Logic, Language and Computation, Universiteit van Amsterdam.

Haeupler, Bernhard et al. (2016). “Discovery through Gossip”. In: *Random Structures and Algorithms*.

Maubert, Bastien and Sophie Pinchinat (2014). “A General Notion of Uniform Strategies”. In: *International Game Theory Review*.

Plaza, Jan (2007). “Logics of Public Communication”. In: *Synthese*.

R. Apt, Krzysztof, Davide Grossi, and Wiebe Hoek (2016). “Epistemic Protocols for Distributed Gossiping”. In: *Electronic Proceedings in Theoretical Computer Science* 215, pp. 51–66. DOI: 10.4204/EPTCS.215.5.

Tijdeman, R. (1971). “On a telephone problem”. In: *Nieuw Archief voor Wiskunde (3)* 19, pp. 188–192.

van Ditmarsch, Hans et al. (2016). “Epistemic Protocols for Dynamic Gossip”. In: *Journal of Applied Logic,*

— (2018). “Dynamic Gossip”. In: *Bulletin of the Iranian Mathematical Society*.

van Eijck, Jan (2014). *DEMO-S5*. Tech. rep. CWI Amsterdam.

Yu, Quan, Ximing Wen, and Yongmei Liu (2013). “Multi-Agent Epistemic Explanatory Diagnosis via Reasoning about Actions”. In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*.