# Probabilistic Agent Based Modelling: Mathematical foundations

Daniel Tang
*Leeds Institute for Data Analytics*

## Abstract

In order to do modelling well we need to do calibration, data assimilation and uncertainty quantification properly. All of these problems can be posed in terms of probabilistic inference. At present, tools to do probabilistic inference on agent based models are lacking. In this paper I introduce the idea of an agent based dynamical system and develop a mathematical foundation for thinking about and working with probabilistic inference in these systems.

## 1 Introduction: The problem

In this paper we will consider the following problem: Suppose we have a complex system that can be described in terms of interacting agents. Suppose also that we have some (possibly incomplete) knowledge of the behaviour of those agents. Suppose we also have a set of real-world, noisy observations of the aggreagate behaviour of the system and are able to define an observation operator for each observation type that defines the conditional probability of making an observaton given the state of the system. We would like to gain information about a the posterior distribution of a (hitherto unobserved) observable at some time in the past, present or future, given our knowledge of the behaviour of the agents, our observations and our observation operators. That information may be an expectation value, a modal value, or a distribution chosen from a family that minimises the KL divergence. (or the modal trajectory in time)

### 1.1 Defining agents

We will proceed by specifying our knowledge of the behaviour of an agent as a computer program written from the point of view of an agent. In order to express any incompleteness in our knowledge of the behaviour, we suppose there is a set of global paramaters, which we do not know but which we have a prior distribution for, that may infulence an agent's behaviour. In addition, we may make the agent's behaviour stochastic in order to express our uncertainty in the agent's behaviour. By doing this, we can track the effect of our uncertainty in the agent's behaviour on the uncertainty in the aggregate behaviour of the system.

### 1.2 Defining observation operators

Similarly, we will define observation operators as computer programs that take a state of the system as input and return an observation (i.e. a real number or vector of reals). The observation operator will itself be stochastic in order to represent the noise in the observation.

We can connect real-world observations to observations of the model by means of an `observe(x,v)` command which should be interpreted as "we observe the variable `x` to have value `v`".

In this way, our whole problem can be specified in a single computer program that looks very much like a normal simulation which, if executed, would generate an instance of our data and an instance of the observable we're interested in. The values generated would probably not match our actual data, but there is a positive, if small, probability that they would match.

Our problem can now be cast as one of doing inference on this computer program. That is, finding the expectation value of the observable of interest given that the generated data matches our real-world observations.

## 2 Agent based dynamical systems

We introduce the concept of an *agent based dynamical system* which consists of a set of interacting *agents*. Each agent has a state and can interact with other agents. At any given time, $t$, each agent has a propensity, $r$, to perform an action such that, in the infinitesimal slice of time, $[t, t+dt]$, there is a probability $rdt$ that the agent performs the action. The only actions an agent can perform are to create a new agent or iterate over agents that have a given property. Surprisingly, we will show that this will be enough to express all agent behaviours.

## 3 Object oriented implementation of an agent based dynamical system

In order to implement an agent based dynamical system in an object oriented programming language, we have a class, `ABM`, which is templated on the agent state type `T`. This class represents a probability distribution over ABM states and has three methods:

| | |
|---|---|
| `add(s)` | add an agent in state `s` to the ABM and return the modified ABM |
| `integrate(B,T)` | integrate the ABM over T units of time, using behaviour `B` for the agents |
| `observe(o,v)` | observe that the observable `o` has value `v` and return the resulting `ABM` |
| `evaluate(e)` | Calculate the expectation value of the function e |

A `Behaviour` class consists of a set of `actions` and `interactions`. An `action` is a triplet $(t,r,l)$, where $t$ is the trigger which defines which agents the action applies to, as a function of the agent's state, $r$ is the rate at which the action happens once it is triggered and $l$ is a lambda function that performs the action. $l$ takes the current state of the acting agent, $k$, and a distribution that represents the agent's environment (i.e the distribution of the other agents in the system, given that there is an agent in state $k$) and returns the ABM distribution after the action is performed. An `interaction` is very much like an action but the trigger is a function of two agent states and $l$ can refer to both of those states.

## 4 A probabilistic interpretation of an `ABM`

An `ABM` object represents a probability distribution over the states of an agent based model. To be mathematically precise, an `ABM` belongs to a symmetric Fock space on the state space of the agents, $T$, which we'll write as $\mathscr{F}(T)$. However, a Fock space has infinite dimensionality so it is not at all obvious how we could go about representing such a mathematical object as a data structure of manageable size for numerical computation.

We'll start by naming two special distributions: the distribution that is definitely the empty set (i.e. with probability 1 there are no agents in the system), which we'll write as $\emptyset$, and the state that has zero probability for all states, which we'll write as $\mathbf{0}$.

Surprisingly, we can express everything we need with these two states and just two Fock space operators; the *annihilation operator*, $a_k$, and the *creation operator*, $a_k^\dagger$.

The creation operator, $a_k^\dagger$, represents the addition of an agent in state $k$ to every state in the probability distribution. By definition, the probability of the ABM being empty after the action of a creation operator is zero.

Creation operators always commute with other creation operators

$$a_k^\dagger a_q^\dagger = a_q^\dagger a_k^\dagger \qquad (1)$$

We define the annihilation operator $a_k$ as the operator that has the following properties

$$a_k a_q = a_q a_k \qquad (2)$$

$$a_k a_q^\dagger - a_q^\dagger a_k = \delta_{kq} \qquad (3)$$

$$a_k \emptyset = \mathbf{0} \qquad (4)$$

where $\delta_{kq}$ is the Kronecker delta function[1].

For a given (non-probabilistic) ABM, we define the *occupation number* of agent state $s$ to be the number of agents that are in that state. The annihilation operator $a_s$ multiplies the probability of each ABM state by the occupation number of $s$ and removes one agent in state $s$.

This behaviour follows from the properties above. Using a recursive proof, suppose that $a_k a_k^\dagger S = nS$ for some integer $n$ then, from equation 3,

$$a_k a_k^\dagger a_k^\dagger S = (1 + a_k^\dagger a_k) a_k^\dagger S = (n+1) a_k^\dagger S$$

We can start the recursion using equations 3 and 4

$$a_k a_k^\dagger \emptyset = (1 + a_k^\dagger a_k)\emptyset = 1\emptyset$$

We can now interpret the `add(s)` operation as the creation operator $a_s^\dagger$. When the same operator is applied

[1]To be precise, the 1 and 0 of the Kronecker delta function should be replaced by the identity operator and the operator that always returns $\mathbf{0}$, but for notational simplicity we'll allow the free interpretation of numbers as operators that multiply all probabilities of an ABM distribution by that number.

multiple times, we will use the power notation, e.g. $a_s a_s = a_s^2$.

A probability distribution over an ABM can be represented as a sum of creation operators on the empty set. The result looks something like a polynomial:

$$P = \sum_{\mathbf{n}\in\mathbb{N}^{|k|}} p_{\mathbf{n}} \prod_k a_k^{\dagger n_k} \emptyset$$

where $\mathbf{n} = (n_1 ... n_{|k|})$ and $p_{\mathbf{n}}$ is the probability that there are $n_1$ agents in state 1, $n_2$ agents in state 2 etc.

For ease of notation, from now on we will omit the trailing $\emptyset$ on distributions when this is not ambiguous.

An `deselby.Action` object represents an operator which takes an ABM distribution and returns a rate of change of an ABM distribution, and can be defined as

$$A(t,r,l) = \int_{t(k)} (l_k - a_k^\dagger) a_k r\, dk$$

where the integral reverts to a sum in the case of discrete state variables.

An `Interaction` object is very similar to an `deselby.Action` but with a double integral

$$I(t,r,l) = \int \int_{t(j,k)} (l_{jk} - a_j^\dagger a_k^\dagger) a_j a_k r\, dj\, dk$$

A `Behaviour` object is just the sum of its `Actions` and `Interactions`:

$$H = \sum_n A_n + \sum_m I_m$$

So, a `Behaviour` is also an operator from an ABM distribution to a rate of change. In this way, we can write the equation of motion of a probabilistic ABM as

$$\frac{\partial P}{\partial t} = HP$$

and we see that the `Behaviour` takes the place of the Hamiltonian when seen as a dynamical system. Given this, we can now express the meaning of the `integrate` method as

$$P(T) = \int_0^t HP\, dt = e^{tH} P(0) \tag{5}$$

where we define the exponential of an operator to be

$$e^{tH} = \sum_{k=0}^{\infty} \frac{(tH)^k}{k!}$$

and $H^n$ is just $n$ repeated applications of $H$ (taking care to respect the order of non-commutative operators).

Let the dot product of two ABM probability distributions be defined as

$$\left(\sum_{\mathbf{n}} c_{\mathbf{n}} a^{\dagger\mathbf{n}}\right) \cdot \left(\sum_{\mathbf{m}} d_{\mathbf{m}} a^{\dagger\mathbf{m}}\right) = \sum_{\mathbf{k}} c_{\mathbf{k}} d_{\mathbf{k}}$$

where $\mathbf{n} = (n_1 ... n_N)$ and we use the convenience notation

$$a^{\dagger\mathbf{n}} = \prod_i a_i^{\dagger n_i}$$

so that, for example

$$a^{\dagger\mathbf{n}} \cdot P = p_{\mathbf{n}}$$

is just the probability that $P$ is in the state with occupation number $\mathbf{n}$.

If we let an observable, $P(v|s)$, be defined as the conditional probability of making a particular observation, $v$ given the state, $s$, of the ABM, then we can define the `observe` method as the operator $O_{P(v|s)}$ such that

$$O_{P(v|s)} P = \frac{1}{P(v)} \int_S s P(v|s)(s \cdot P)\, ds$$

where the integration ranges over the ABM states. The observation operator has the effect of applying a Bayesian update to a distribution in order to reflect how knowledge of the observation's value updates our belief about the distribution of the system. The prior probability $P(v)$ is often not known so we define an unnormalised version of the observation operator $\hat{O}_{P(v|s)}$ which omits it (i.e. takes its value to be 1).

WThe `evaluate` method is defined as

$$\mathbf{E}[f]P = \int_S f(s)(s \cdot P)\, ds$$

With the above interpretations, we can see how a distribution over an ABM, which belongs to an infinite dimensional Fock space, can be represented in a data structure whose size is of the order of the size of the computer program that defines the ABM's behaviour, and we can see how such a representation can be systematically derived from the program.

## 5 Some analytic examples

As a demonstration, let's look at some very simple ABMs which, with the help of the formalism we have developed, we can solve analytically.

### 5.1 Single agent

We start with the simplest possible agent based dynamical system: a system which is either empty or there is a single, stateless agent. Suppose that at time $t = 0$ the agent is definitely present and between time $t$ and $t + dt$ the agent leaves the system with probability $dt$. So, we would expect the probability that the agent exists to start at 1 and reduce exponentially according to $e^{-t}$ as time progresses.

Here's the program that represents this system

```
ABM abm
abm.add(void)

Behaviour b
b.add({true}, {1.0}, (state, abm) -> {
  return abm
})
return b.integrateTime(abm, T)
```

remember that the ABM passed to the action in the bahaviour is the ABM minus the agent whose behaviour we're considering. So in this case, it's the empty ABM and returning this is equivalent to the agent removing itself from the system.

Starting from the first line of the program, we create a new ABM which is definitely empty, so is in the state $\emptyset$. We then add an agent, after which the ABM will be in state $a^\dagger\emptyset$. Next we create a behaviour and add an action to it. The action is applicable to all agents (i.e. the only agent in the system) and occurs at a rate of 1 per unit time. The act itself is just the "do nothing" action, which is just the identity operator, 1. So, the behaviour can be expressed as the operator

$$H = (1 - a^\dagger)a$$

so, the equation of motion is

$$\frac{\partial P}{\partial t} = HP = (1 - a^\dagger)aP$$

Solving this for $P(0) = a^\dagger\emptyset$ gives

$$\begin{aligned} P(t) &= e^{t(1-a^\dagger)a}a^\dagger\emptyset \\ &= \sum_{k=0}^{\infty} \frac{\left(t(1-a^\dagger)a\right)^k}{k!} a^\dagger\emptyset \\ &= \sum_{k=0}^{\infty} \frac{(-t)^k \left((a^\dagger-1)a\right)^k}{k!} a^\dagger\emptyset \end{aligned}$$

but since $a^n a^\dagger\emptyset = \mathbf{0}$ for $n > 1$

$$\left((a^\dagger - 1)a\right)^k a^\dagger\emptyset = (a^\dagger - 1)aa^\dagger\emptyset$$

for $k > 0$, so

$$P(t) = \left(1 + \sum_{k=0}^{\infty} \frac{(-t)^k}{k!}(a^\dagger - 1)\right)\emptyset$$

where the initial 1 is a correction to the sum in the case that $k = 0$. We can now turn the sum back into an exponential to give

$$P(t) = (1 + e^{-t}(a^\dagger - 1))\emptyset = (1 - e^{-t})\emptyset + e^{-t}a^\dagger\emptyset$$

So, the probability that the agent exists (i.e. state $a^\dagger\emptyset$) reduces exponentially as time progresses, exactly as we expected, and the probability of the empty state increases exponentially so that the sum over both states remains at 1.

## 5.2 Many agents with state

We can now show how our operators extend very elegantly to systems with more than one agent and add state to the agents. We can see immediately that our creation operator works as we would expect:

$$a_k^\dagger a_k^{\dagger n} = a_k^{\dagger n+1}$$

This also applies to distributions. Suppose there's a 0.5 probability that the system is empty and a 0.5 probability that there's one agent, so $P = 0.5 + 0.5a^\dagger$. So

$$a^\dagger P = a^\dagger(0.5 + 0.5a^\dagger) = 0.5a^\dagger + 0.5a^{\dagger 2}$$

So, after applying the creation operator, there's an equal probability of there being either one or two agents, i.e. the operator simultaneously adds an agent to all possible states.

## 5.3 Poisson model

Suppose we start with a system with one agent in state 0 (i.e. $P(0) = a_0^\dagger$) and that that single agent adds new agents in state 1 at a constant rate, $r$. This is just a Poisson process, so we would expect the distribution over the number of agents in state 1 to be distributed according to the Poisson distribution.

The program for this model would be

```
ABM abm
abm.add(0)

Behaviour b
b.add(state -> state==0, {r},
      (state, abm) -> {
        abm.add(0)
        abm.add(1)
        return abm
      })
return b.integrateTime(abm, T)
```

Using our rules for constructing the equation for behaviour, for this program we have

$$\frac{\partial P}{\partial t} = (a_0^\dagger a_1^\dagger - a_0^\dagger)a_0 r$$

This can again be integrated analytically:

$$P(t) = e^{tH}P(0) = e^{-rt(1-a^\dagger)a_0^\dagger a_0}a_0^\dagger$$

but since there's always exactly one agent in state 0, $a_0^\dagger a_0$ is just the identity operator so

$$P(t) = e^{-rt}e^{rta_1^\dagger}a_0^\dagger = e^{-rt}\sum_k \frac{(rt)^k}{k!}a_1^{\dagger k}a_0^\dagger$$

so, the probability of there being $k$ agents in state 1 at time $t$ (i.e. the coefficient of the $k^{th}$ power of $a_1^\dagger$) is

$$c_k = \frac{(rt)^k e^{-rt}}{k!}$$

which is just the Poisson distribution, as we expected.

## 5.4 Binomial model

Suppose we start with $n$ agents and each agent has a propensity, $r$, to annihilate itself. We would expect the distribution over the number of remaining agents to follow a binomial distribution, with an exponentially decaying probability of survival, since we are effectively taking $n$ independent flips of a weighted coin to decide whether each agent has annihilated itself or not.

The program for this, and the resulting behaviour, would be the same as in the example we gave for the single agent case, but this time we add more agents to the initial state.

```
ABM abm
for(i in 1...n) {
        abm.add(void)
}

Behaviour b
b.add({true}, {r}, (state, abm) -> {
  return abm
})
return b.integrateTime(abm, T)
```

The behaviour is expressed as

$$H = (1-a^\dagger)ar$$

The boundary condition this time is $P(0) = a^{\dagger n}$. The solution to this is

$$P(t) = \sum_m \binom{n}{m} e^{-rtm}(1-e^{-rt})^{n-m}a^{\dagger m}$$

The coefficients of $a^\dagger$ (i.e. the probabilities that there are $m$ agents) make up a Binomial distribution, which, again, is what we expected.

The above examples, although very simple, demonstrate how we can start with a computer program which specifies the behaviour of a single agent, convert this into an operator, expressed in terms of creation and annihilation operators, which can then be used to specify the rate of change of a probability distribution over the whole system, then integrate that operator to find the probability distribution of the system at some time in the future.

## 5.5 Interacting agents: SIR model

Now let's get agents to interact. Suppose we have a conpartmental model of infectious disease spread. Suppose there's a population of $n$ people. Each person can be one of suseptible, infected or recovered. Infected people have a propensity, $\beta$, to meet and infect a suseptible person, in which case the suseptible person becomes infected. Also, infected people have a propensity, $\gamma$ to recover.

The program for this would look something like

```
ABM abm
for(i in 1...ns) {
        abm.add(S)
}
for(i in 1...ni) {
        abm.add(I)
}
for(i in 1...nr) {
        abm.add(R)
}

Behaviour b
b.add((s1,s2) -> {s1==S && s2==I},
      {beta},
      (state1, state2, abm) -> {
      abm.add(I) // replace self
      abm.add(I) // replace other
                 // as an infected
  return abm
})
b.add(s -> {s == I}, {gamma},
      (state, abm) -> {
  abm.add(R) // agent recovers
})
return b.integrateTime(abm, T)
```

If we let $a_S^\dagger$, $a_I^\dagger$ and $a_R^\dagger$ represent suseptible, infected and recovered people respectively, then the Hamiltonian of the above behaviour is

$$H = (a_I^{\dagger 2} - a_I^\dagger a_S^\dagger)a_I a_S \beta + (a_R^\dagger - a_I^\dagger)a_I\gamma$$

Although this is a differential equation, it's worth pointing out explicitly that it is representing discrete interactions between agents. So, it is still valid for small numbers of agents and correctly captures things like disease eradication, unlike the standard differential version of the SIR model, which is only accurate for large numbers of agents.

Although this cannot be solved analytically, we can see that it has the expected behaviour. For example, if there are no infected agents, the $a_I$ operators will return 0 and the rate of change will be zero.

## 5.6 Arithmetic

If an agent has numerical variables in its state, we can represent arithmetic on those variables with a change of state operation. For example, if an agent has two integers, $a$ and $b$ in its state, we can represent the operation $a = a + b$ as

$$H = (a^{\dagger}_{[a+b,b]} - a^{\dagger}_{[a,b]})a_{[a,b]}$$

The same method can be used to apply any operator on any number of variables.

## 6 Transformations

It will often be useful to transform the ABM distriution between different representations so we'll build up a way of doing this now.

First of all, we'll construct the expectation operator which returns the sum of all probabilities in a distribution. We can formally define this as

$$\mathbf{E}[] = \emptyset \cdot e^{\int_S a_s ds}$$

which, if we take the expansion of the exponential, can be seen to have the effect of integrating over the whole Fock space.

We'll now transform to a moment representation, where the moment of a distribution $P$ is defined as

$$m(s_1^{j_1}...s_n^{j_n})[P] = \frac{\mathbf{E}[a_{s_1}^{j_1}...a_{s_n}^{j_n}P]}{(j_1!)...(j_n!)}$$

for convenience, we'll use the shorthand notation

$$m(s_1^{j_1}...s_n^{j_n})[P] = m(S^J)[P] = \frac{\mathbf{E}[a_S^J P]}{J!}$$

To help our intuitive understanding of the moments, we note that the moment, $m(s)$, is just the average $s$-occupation number. We now extend the idea of occupation number to pairs of states $(s_1, s_2)$ so that the $(s_1, s_2)$-occupation number is the number of distinct pairs of agents whose states are $s_1$ and $s_2$ respectively. So the $(s_1, s_2)$-occupation number of state $...s_1^n s_2^m...$ is $mn$. If we're after pairs of agents in the same state, $m(s^2)$ from a state $...s^n...$, then there are $\frac{n(n-1)}{2}$ *distinct* pairs. In the same way, we can extend occupation number to any number of agent states. The moments can now be seen to be the expectations of the occupation numbers.

So, if there is an interaction that requires a set of agents in given states, the moment will tell you the expectation value of the rate at which that interaction will occur in the whole system if the agent-based interaction rate is 1.

We can now build an moment heirarchy transform which transforms a distribution that is expressed in terms of the creation operators into a distribution that is expressed in terms of a *moment*-space creation operator, $A_s^{\dagger}$

$$\mathbf{M}[P] = \sum_{n=0}^{\infty} \int_{s_1...s_n, m_1...m_n} m(a_{s_1}^{j_1}...a_{s_n}^{j_n})[P] A_{s_1}^{\dagger j_1}...A_{s_n}^{\dagger j_n}\emptyset$$

We can now derive the relationship between the moment-space and Fock-space creation and annihilation operators.

$$\mathbf{M}[a_x^{\dagger}P] = \sum_{n=0}^{\infty} \int m(a_S^J a_x^{\dagger})[P] A_S^{\dagger J}\emptyset$$

but by the commutation relation on Fock-space creation operators,

$$a_x^j a_x^{\dagger} = a_x^{(j-1)} a_x^{\dagger} a_x + a_x^{(j-1)}$$

so

$$a_x^j a_x^{\dagger} = a_x^{\dagger} a_x^j + j a_x^{(j-1)}$$

but for $j > 0$

$$\frac{\mathbf{E}[j a_x^{(j-1)}P]}{j!} = \frac{\mathbf{E}[a^{(j-1)}P]}{(j-1)!} = m(a^{j-1})[P]$$

so

$$\mathbf{M}[a_x^{\dagger}P] = \sum_{n=0}^{\infty} \int m(a_x^{\dagger} a_S^J)[P] A_S^{\dagger J}\emptyset + \sum_{n=0}^{\infty} \int_{j_x>0} m(a_x^{-1} a_S^J)[P] A_S^{\dagger J}\emptyset$$

But since $\mathbf{E}[a_x^{\dagger} a_{s_1}...a_{s_n}P] = \mathbf{E}[a_{s_1}...a_{s_n}P]$

$$\mathbf{M}[a_x^{\dagger}P] = \mathbf{M}[P] + A_x^{\dagger} \sum_{n=0}^{\infty} \int_{j_x>0} m(a_x^{-1} a_S^J)[P] A_x^{\dagger -1} A_S^{\dagger J}\emptyset$$

$$= (1 + A_x^{\dagger})\mathbf{M}[P]$$

Similarly with the annihilation operator

$$\mathbf{M}[a_x P] = \sum_{n=0}^{\infty} \int \frac{\mathbf{E}[a_S^J a_x P]}{J!} A_S^{\dagger J}\emptyset$$

$$= \sum_{n=0}^{\infty} \int \frac{\mathbf{E}[a_S^J a_x P]}{J!} A_S^{\dagger J} A_x A_x^{\dagger}\emptyset$$

which, if we define the moment-annihilation operator to have the commutation relation

$$A_k A_q^\dagger - A_q^\dagger A_k = \delta_{kq}$$

in the same way as the Fock-space annihilation operator, then

$$= \sum_{n=0}^{\infty} \int \frac{\mathbf{E}[a_S^J a_x P]}{J!} \frac{1}{j_x + 1} A_x A_S^{\dagger J} A_x^\dagger \emptyset$$

$$= A_x \sum_{n=0}^{\infty} \int m(a_S^J a_x)[P] A_S^{\dagger J} A_x^\dagger \emptyset$$

so

$$\mathbf{M}[a_x P] = A_x \mathbf{M}[P]$$

This provides a very convenient, easily automated way of going from a computer program of an ABM to the dynamics of the moments of the model. At its simplest, this provides a way of automatically generating the moment closure for an ABM.

## 7 The path integral and conditional probability

The time integrated probability

$$P(t) = e^{tH} P(0) \tag{6}$$

can be given an alternative interpretation as a path integral. To do this, we start by expressing $P(0)$ as a sum

$$P(0) = \int (a_{s_1}^\dagger ... a_{s_n}^\dagger \emptyset ds_1 ... ds_n \cdot P(0)) a_{s_1}^\dagger ... a_{s_n}^\dagger$$

$$= \int (a_S^\dagger \cdot P(0)) a_S^\dagger dS$$

so

$$P(t) = e^{tH} \int (a_S^\dagger \cdot P(0)) a_S^\dagger dS$$

if we expand into another sum

$$P(t) = \int a_C^\dagger \cdot \left( e^{tH} \int a_S^\dagger \cdot P(0) a_S^\dagger dS \right) a_C^\dagger dC$$

rearranging

$$P(t) = \int \int \left( a_C^\dagger \cdot e^{tH} a_S^\dagger \right) \left( a_S^\dagger \cdot P(0) \right) a_C^\dagger dS dC$$

The first term above can be interpreted as the probability that the system ends in state $C$ at time $t$ given that it started in state $S$ at $t = 0$.

$$P(C|S) = \left( a_C^\dagger \cdot e^{tH} a_S^\dagger \right)$$

we can now split the exponential

$$a_C^\dagger \cdot e^{tH} a_S^\dagger = a_C^\dagger \cdot e^{0.5tH} e^{0.5tH} a_S^\dagger$$

$$= \int_I (a_C^\dagger \cdot e^{0.5tH} a_I^\dagger)(a_I^\dagger \cdot e^{0.5tH} a_S^\dagger) dI$$

$$= \int_I P(C|I) P(I|S) dI$$

[this comes out of the linearity of the operators and the definition of dot product]

Using the same principle, we can split it into as many pieces as we please

$$P(C_N|C_1) = \int_{C_{1...N}} \prod_{n=1}^{N-1} (a_{C_{n+1}}^\dagger \cdot e^{\frac{tH}{N}} a_{C_n}^\dagger) dC_{1...N}$$

## 8 Making observations and doing inference

If we have a prior $P(S)$ over system states and an observation operator $O$ that goes from a probability distribution over a system state to a probability distribution over the reals and we observe that the obervation operator has value $o$ then we can define the posterior

$$P(S|OS = o) \propto P(OS = o|S) P(S)$$

i,e, we multiply the prior probability distribution over $S$ by the likelihood of the observation. We can represent likelihoods as polynomials in a similar way to probability distributions

$$P(o|.) = \sum_n P(o|a^{\dagger n}) a^{\dagger n}$$

However, multiplication of two probability distributions is not the same as polynomial multiplication. Instead, it is multiplication of the coefficients:

$$\left( \sum_n c_n a^{\dagger n} \right) \times \left( \sum_m d_m a^{\dagger m} \right) = \sum_n c_n d_n a^{\dagger n}$$

Going back to the example of a single agent that has a unit propensity to annihilate itself. If we start with a prior that there's a 0.5 probability that the agent exists, then observe that it doesn't exist at time $t$, what's the posterior probability that it existed at time $t = 0$? The equation of motion is

$$\frac{\partial P}{\partial t} = HP = \left( \frac{\partial}{\partial a^\dagger} - a^\dagger \frac{\partial}{\partial a^\dagger} \right) P$$

Integrating this for the case $P(0) = (1 - p) + pa^\dagger$

$$P(t) = e^{\frac{\partial}{\partial a^\dagger} - a^\dagger \frac{\partial}{\partial a^\dagger}} S$$

$$= 1 - pe^{-rt} + e^{-rt} pa^\dagger$$

So,

$$P(o|.) = 1 + (1 - e^{-rt})a^\dagger$$

So

$$P(.|o) \propto (1-p) + (1 - e^{-rt})pa^\dagger$$

normalising

$$\int P(.|o)dp = (1 - 0.5e^{-rt})$$

$$P(.|o) = \frac{(1-p) + (1-e^{-rt})pa^\dagger}{(1-0.5e^{-rt})}$$

So, if the prior $P(0) = 0.5 + 0.5a^\dagger$ then the posterior is given by

$$P(.|o) = \frac{0.5 + 0.5(1-e^{-rt})a^\dagger}{(1-0.5e^{-rt})}$$

## 9 Numerical methods

So far, we've considered simple systems that have analytic solutions. However, almost all of the systems of interest will not have an analytic solution. We'll now introduce some techniques to help us to numerically approximate solutions.

Because the number of coefficients in the probabilistic state of a system expands exponentially with the number of variables in an agent's state and with the number of agents, for most real world models it isn't feasible to explicitly represent the probabilistic state as a list of coefficients. There are three ways around this. We can represent the state symbolically, we can approximate the state or we can abstract over the state.

### 9.1 Symbolic representation of probability distributions

In almost all cases of interest a probability distribution, although very high dimensional and complex, will be the result of a (potentially large but manageable) number of operations. What's more, we can construct these states from the empty set of agents $P = 1$. So, instead of representing a system state as a list of monomial coefficients we can represent it as a list of operations on 1:

$$P = \prod_n O_n 1$$

all of which are constructed from the creation and annihilation operators.

When thought of like this, we never have to explicitly represent a probability distribution as a polynomial or as any other kind of representation. Instead, all we ever need to deal with is creation and annihilation operators on probability distributions.

We can reduce the necessary behaviour of these operators to an abstract algebra that obeys the following axiomatic rules:

$$a^\dagger 0 = 0 \tag{7}$$

where $d_{kq}$ is the Kronecker delta function.

Using these rules, we can show that everything acts as expected. Starting with the number operator, $a^\dagger a$, we have, from equations 4 and 7

$$a^\dagger a 1 = 0$$

and more generally, if $a^\dagger aS = nS$ then, from equation 3

$$a^\dagger aa^\dagger S = a^\dagger(a^\dagger a + 1)S = (n+1)a^\dagger S$$

so the number operator acts as expected.

Given this we can show that the annihilation operator also acts as expected. Suppose again that $a^\dagger aS = nS$ then

$$aa^\dagger S = (a^\dagger a + 1)S = (n+1)S$$

Equations 2 and 1 ensure that these results extend to multivariate cases.