Anoma Research Topics | TECHNICAL REPORT

Message Logic: a logical foundation for service commitments

Murdoch J. Gabbay [a] **and Naqib Zarin** [a]

[a] Heliax AG

* *E-Mail: jamie@heliax.dev, naqib@heliax.dev*

Abstract

We introduce *Message Logic*, a dynamic modal logic for specifying and reasoning about message-passing in distributed systems. We study in particular how to express that participants make commitments to each other about their future behaviour. This leads us to develop a kind of 'partial implication' which we call *pragmatic implication*, and to an axiomatisation of the action of offering a commitment. We then discuss how this logic can be understood in the context of industrial practice.

Keywords: Distributed Systems ; Dynamic Modal Logic ; Service Commitments

(Received: November 30, 2024; Version: December 4, 2024)

Contents

1. Introduction

1.1. What is a distributed system?

A **distributed system** consists of a set of **participants** (or **processes**) working in collaboration by passing **messages** to one another. Furthermore, we assume that messages are the *only* way for participants to communicate, and the *only* observable of the system.[1]

In such a situation, the behaviour of a system *is* the messages that get sent, and expectations on its behaviour *are* expectations about who sends messages, the contents of the messages, and when the messages are sent. Accordingly, we define:

Definition 1.1.1. A **service commitment** is an assertion that purports to restrict messaging behaviour.

[1] If participants have internal state, the only way this has any relevance to the system is if . . . they put it in a message.

We write 'purports to' because, as we shall see, there are (at least) three concepts in play here:

1. *making* a service commitment,
2. *upholding* a service commitment, and
3. *being evaluated* by another participant to have upheld a service commitment.

Matters are further complicated because service commitments may be conditional ('p will uphold X, barring network outages'), probabilistic ('p will uphold X at least half of the time'), and evaluation may be based on incomplete or gossiped information ('p has upheld X so far, but we haven't been doing business for very long' or 'q told me that p upheld X, and I trust q').

In this document, we will build a logical and operational framework within which the intuitions of

- making and upholding (or not upholding) service commitments, and also
- evaluating whether other participants have done so,

can be formulated and made operationally useful. This logical framework should be of interest to the following readers:

1. Programmers of distributed systems, who might benefit from a logical framework which to design and implement correct and safe algorithms that make, communicate, evaluate, and act on service commitments.
2. Computer scientists, especially those interested in applying formal verification to assert and prove correctness properties from a high-level logical description.
3. Users, who may also want a high-level description of how a distributed system handles its service commitments — one that is higher-level than raw code, but more rigorous than a verbal description.
4. Mathematicians (especially logicians active in modal and epistemic logics) who may be interested in our framework as a modal logic structure with strong practical applications.

1.2. An informal example: back-scratching

We will start by introducing service commitments, using a simple informal example. Even from this example, we will see that the simplicity is deceptive, matters are not completely straightforward, and the ideas conceal quite a lot of structure.

Example 1.2.1. Consider this (apparently) simple service commitment taken from informal life:

'If you scratch my back, I'll scratch yours'.

This seems a straightforward enough commitment, but it is not. For, suppose that you scratch my back — and then you wait, and you wait, but I do not scratch yours. Am I in breach of my service commitment?

Maybe, but maybe not. It depends:

1. I could say 'I said I would scratch your back, but I didn't say when; be patient'. When do you stop waiting and conclude that I broke my promise? A microsecond? A day? A hundred years?
2. I could say 'For my requirements, your backscratch was not of sufficient quality; I'm not reciprocating on the basis of that'.
3. I could say 'Yes, I did say I would scratch your back ... but every time I came to scratch your back, you were out' or '... but you kept asking me at clearly inappropriate times, like at 2am', and so on.
4. I could say 'Actually, I didn't say that at all! Someone must have impersonated me to get a free backscratch. Sorry for your trouble, but I can't help'.
5. You could have been lying:
   (a) You could have claimed to have scratched my back (without having actually done so) and then you demand a 'free' backscratch from me.
   (b) Regardless of whether I actually scratched your back and/or you actually scratched mine, once we have been seen to communicate, you can attack my reputation by reporting negative reviews: that I breached our agreement, or that my backscratching was of poor quality. This may cause third parties to do business with you rather than with me.

1.3. An example from distributed systems: packet forwarding

One of the key challenges of distributed systems is how information gets disseminated to the network of participants. Coordination in an all-to-all communication pattern is too resource-intensive, especially if the network grows large. Instead, participants coordinate with a smaller portion of the network (their *peers*) and rely on these participants to receive or forward new information packets.

Example 1.3.1. Here is a *packet relay* service commitment that participants might make to their peers:

'if you send me a data packet destined for a participant p, then I will forward it to p as soon as possible'.

Again, this is not as straightforward as it seems:

1. What does "soon as possible mean"? Perhaps the packet contains a vote for a block proposal, which becomes irrelevant after a certain deadline.
2. What does "forward it to p mean"? Am I in breach of my service commitment if I forward the packet to one my peers q and request that q deliver the packet to p, because I know that q has p's IP address (and I do not)?
3. What if I have successfully relayed 99 data packets, but then I fail to forward the 100th. Am I in breach of my service commitment — or can I rely on other participants excusing this, given by previous 99 successes?
4. How will you be able to tell if I actually sent the packet to p, or just claim to have done so?
5. Am I responsible only for sending the packet, or am I also responsible for node p actually receiving it?
6. What if p maliciously claims that it has not received the packet, even though I have sent it?
7. Am I in breach of my service commitment if my failure to deliver the 100th packet was due to an event outside of my reasonable control, such as some major incident that takes down the entire network? How about if that incident was initiated by p? What if it was provoked by you?
8. Am I in breach of my service commitment if p left the network and cannot receive any messages?
9. Am I in breach of my service commitment if I leave the network (voluntarily, or perhaps not)?

1.4. General observations

Based on the previous two examples, we make some general observations about service commitments:

1. *Service commitments should be unambiguous.* An omniscient observer should be able to decide for sure what a service commitment means and whether it has been kept. Participants, may have only partial or incorrect information so may only be able to approximate this certainty, but a service commitment should be something that *has* a truth-value, even if it may not always be easy or even possible for participants in the network to know for sure what it is.
2. *Service commitments should be time-bounded.* In practice, we care about commitments that are bounded to a specific time interval (e.g. 'the next 10 microseconds' or 'every Tuesday and Wednesday for the next six months').

3. *Service commitments can be probabilistic, and/or subject to rate-limiting and other similar provisos.*
4. *Service commitments may be gossiped in the network.* Participants may communicate amongst themselves the service commitments they know about, and also evaluations of to what extent and how reliably these commitments have been kept.
5. *Service commitments cannot be made on behalf of someone else.* The only commitment that a participant p can make is about its own behaviour. (This is important especially in the presence of gossip; if p says that q made a commitment, this is *not* in and of itself a commitment on q.)

1.5. Research objectives

Our challenge is to build a logical and operational framework within which the intuitions in the previous sections can be formulated and then made operationally useful in practical large scale distributed systems.

2. The logic

2.1. Types and syntax

Remark 2.1.1. We proceed as follows:

1. We will need to talk about cardinalities and proportions, so we start by defining the extended rational numbers in Definition 2.1.3.
2. We will need to talk about data being passed in messages, so we define a simple type system using *base datatypes*, tuples, and powersets, in Definition 2.1.6 and Figure 1.
3. We define the terms and predicates of message logic in Definition 2.1.9 and Figure 2.

2.1.1. Types

Notation 2.1.2. Write $\mathbb{N} = \{0, 1, 2, \dots\}$ for the natural numbers (starting at 0). If $n \in \mathbb{N}$, write $\mathbb{N}_{\geq n}$

Definition 2.1.3. Define the **extended rational numbers** $\mathbb{Q}_\infty$ by

$$\mathbb{Q}_\infty = \mathbb{Q} \cup \{\infty\}.$$

We let $\mathbb{Q}_\infty$ have addition $+$, multiplication $*$, and division $/$, which agree with those on $\mathbb{Q}$ where they coincide and are such that:

1. $\infty + x = x + \infty = \infty$.

$$\alpha, \beta, \gamma ::= (\delta \in \mathit{BaseDatatype}) \mid \alpha \times \cdots \times \alpha \mid \alpha + \cdots + \alpha \mid \mathsf{Pow}(\alpha).$$
$$\zeta ::= \alpha \to \alpha$$

Figure 1. Syntax of datatypes and function types (Definition 2.1.6)

$$\phi ::= \bot \mid \prod^r a : \phi.\phi \mid \mathsf{P}(t) \mid @_t\phi \mid \mathtt{in}_p(t) \mid \mathtt{out}_p(t) \mid \mathtt{BOU}(\phi) \mid \mathtt{EOU}(\phi)$$

$$t ::= (a \in \mathsf{VarSymb}) \mid (d \in \delta \in \mathit{BaseDatatype}) \mid \mathsf{f}(t) \mid \mathtt{here} \mid \mathtt{now} \mid \{a : \alpha \mid \phi\}$$

Figure 2. Syntax of terms and predicates (Definition 2.1.9)

2. $\infty * x = x * \infty = \infty$.
3. $x/0 = \infty$ for any x (including $x = 0$).
4. $x/\infty = 0$ for any x (including $x = \infty$).

Remark 2.1.4. We currently have just one ∞ (rather than a positive or negative one). Topologically, this means that $\mathbb{Q}_\infty$ is a circle, where we can view 0 as being at the bottom of the circle and ∞ is at the top.[2]

Definition 2.1.5. We assume some **base datatypes** $\delta \in \mathit{BaseDatatype}$:

1. The **empty datatype** $\mathsf{Empty} = \{\}$.
2. The **singleton datatype** $\mathsf{Singleton} = \{*\}$.
3. The **bool datatype** $\mathsf{Bool} = \{\top, \bot\}$.
4. The datatype of **extended rationals** $\mathbb{Q}_\infty$.
5. A set of **primitive data** $d \in \mathsf{PrimData}$ (we may freely assume more than one primitive datatype if required).
6. A partially-ordered set of **times** $n \in \mathsf{Time}$ which for simplicity we identify with some (possibly infinite) initial segment of $\mathbb{N}$.[3]
7. A set of **places** $p \in \mathsf{Place}$ (or **processes** or **participants**).
8. A finite set of **message types** $m \in \mathsf{MessageType}$.
9. We assume that the base datatypes are pairwise disjoint, so that (for example) $\mathsf{Time} \cap \mathsf{Place} = \varnothing$.

Definition 2.1.6. Define the syntax of **datatypes** and **function types** inductively as per Figure 1.

[2] The Wikipedia article on the point at infinity (permalink) is clear, and has good references.

[3] An *initial segment* of $\mathbb{N}$ is either $\mathbb{N}$ or a set of the form $\{0, \ldots, n\}$ for some $n \in \mathbb{N}$. That is, it is a subset I of $\mathbb{N}$ that is *down-closed*, meaning that $\forall x, x' \in \mathbb{N}.(x \in I \wedge x' < x) \Rightarrow x' \in I$. There is only one infinite initial segment of $\mathbb{N}$, namely $\mathbb{N}$ itself.

Remark 2.1.7. Note the following:

1. For simplicity, we identify datatypes with their denotation; e.g. we use $\mathbb{Q}_\infty$ both as a syntactic symbol for 'the datatype of extended rationals' and as itself, namely the set of extended rationals. Similarly, function-types *are* also function-spaces. Our intended denotation for types will not change, so it makes sense to identify a (data)type with its intended denotation.
2. We do not nest $\longrightarrow$ in function-types: e.g. $(\alpha \longrightarrow \beta) \longrightarrow \gamma$ is not a legal function-type, as per Definition 2.1.6. Neither is $\alpha \longrightarrow (\beta \longrightarrow \gamma)$, though $(\alpha \times \beta) \longrightarrow \gamma$ is a legal function-type.
   There is no technical obstacle to doing this, but for now, we do not need the power of explicit higher-order functions[4] and it is helpful for clarity to create a distinction between data, and functions on data.
3. In Definition 2.1.5(6) we identify time with a (possibly infinite) initial segment of the natural numbers. In fancy language, time is a discrete, totally ordered, connected partial order with a least element. Generalisations are doubtless possible, but for now this will suffice.

2.1.2. Syntax

Definition 2.1.8. We fix some more data:

1. A set of **function symbols** $\mathsf{f} \in \mathsf{FSymb}$, to each of which is associated a function-type, which we will write $\mathsf{type}(\mathsf{f})$. If $\mathsf{f} \in \mathsf{FSymb}$ then we may write
$$\mathsf{f} : \alpha \longrightarrow \alpha' \quad \text{for the assertion} \quad \mathsf{type}(\mathsf{f}) = (\alpha \longrightarrow \alpha').$$
   We may freely assume standard function symbols, including 0, 1, $+$, $*$, tupling $(\text{-}, \ldots, \text{-})$, and projections $proj_i$, with their usual meanings and types. Meaning will always be clear.
2. For each datatype α we assume a **cardinality** function symbol
$$\# : \mathsf{Pow}(\alpha) \longrightarrow \mathbb{Q}_\infty.$$
   If $P \subseteq \alpha$ then $\#P$ returns the (finite) cardinality of P if P is finite, and $\#P$ returns ∞ if P is infinite.
3. A set of **predicate symbols** $\mathsf{P} \in \mathsf{PSymb}$ to each of which is associated a datatype, which we will write $\mathsf{arity}(\mathsf{P})$ and call the **arity** of P. If $\mathsf{P} \in \mathsf{PSymb}$ then we may write
$$\mathsf{P} : \alpha \quad \text{for the assertion} \quad \mathsf{arity}(\mathsf{P}) = \alpha.$$

[4] This comment is bogus in the sense that $\mathsf{Pow}(\alpha)$ is a legal datatype and it is just as higher-order as $\alpha \longrightarrow \alpha$, but operationally we will use $\mathsf{Pow}(\alpha)$ as a powerset, not a function-space.

We may freely assume standard predicate symbols, such as $=$, $<$, and $\leq$, with their usual meanings. Meaning will always be clear.
For each datatype α we assume a **set membership** predicate symbol $\in : \alpha \times \mathsf{Pow}(\alpha)$.

4. For each datatype α, we assume a disjoint set of **variable symbols** $a, b, c, \cdots \in \mathsf{VarSymb}(\alpha)$.
We may annotate a variable with its type, as $a : \alpha$ or $c : \gamma$. This is especially useful when writing a comprehension term, so that if $a \in \mathsf{VarSymb}(\alpha)$ and ϕ is a predicate then we may write $\{a \mid \phi\}$ as $\{a : \alpha \mid \phi\}$. As typeset strings these look different, but they denote identical abstract syntax — in the latter case, we just remind ourself of the (unique) type of a.[5]

Definition 2.1.9. Define the syntax of **terms** and **predicates** inductively as per Figure 2.

We call a predicate or a term **closed** when it has no free variables.

Notation 2.1.10. Our syntax for datatypes from Figure 1 admits tuple types, and our syntax for terms and predicates includes $\mathsf{P}(t)$ (a predicate symbol applied to a term) and $\mathsf{f}(t)$ (a function symbol applied to a term).

This means that if we have a tuple $(x_1, \ldots, x_n)$ and wish to apply a predicate P or function symbol f to it, then we should write these $\mathsf{P}((x_1, \ldots, x_n))$ and $\mathsf{f}((x_1, \ldots, x_n))$. For our sanity and for that of the reader, we will elide double brackets and shorten this to $\mathsf{P}(x_1, \ldots, x_n)$ and $\mathsf{f}(x_1, \ldots, x_n)$.

Remark 2.1.11.

1. We could easily unify function symbols and predicates, using the base datatype of booleans $\mathsf{Bool} = \{\bot, \top\}$. Boolean connectives like $\wedge$ and $\Rightarrow$ could also become function symbols — but not $\forall$, unless we also admit higher types, in which case $\mathsf{Pow}(\alpha)$ could become $\alpha \to \mathsf{Bool}$.
However, note that if predicates were functions then the type corresponding to $\mathsf{P} : \alpha$ would *not* be $\alpha \to \mathsf{Bool}$; it would be $(\mathsf{Time} \times \mathsf{Place}) \times \alpha \to \mathsf{Bool}$. See Definition 2.2.1(4) and Remark 2.2.2(3).
2. There is some sneaky subtyping going on, because $\mathsf{Time} \subseteq \mathbb{Q}_\infty$. This lets us write things like `now+1` and not worry about edge cases.[6]

[5] If you are asked to buy milk and are provided with a shopping list in which the word 'milk' is circled three times in red ballpoint pen, perhaps because you forgot to buy milk in the past, then this does not change the syntax of the shopping list. The red circle is metadata, intended to help you notice the word 'milk' (and perhaps laden with other meaning, such as judgement on your past failures and warning not to repeat them), but the word 'milk' is still just the word 'milk' and the syntax of the underlying shopping list is unaffected. In the same way, '$: \alpha$' is metadata added to the syntax $\{a \mid \phi\}$ to make sure that we know what the type of a is.

[6] There may be other ways to set things up.

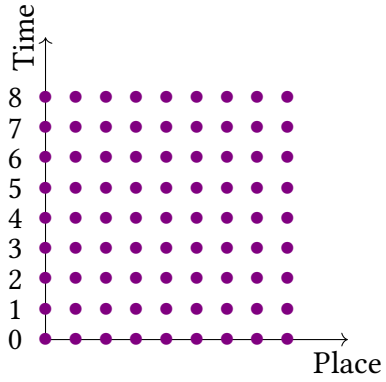

Figure 3. Picture of a model with 10 places and 9 times

2.2. Semantics

Definition 2.2.1. A **model** ς consists of the following data:

1. Sets Time_ς and Place_ς of **times** and **places**. We will usually omit the subscript and just write these Time and Place.
2. Zero, one, or more sets of primitive data as convenient and required.
3. For each function symbol $\mathsf{f} \in \mathsf{FSymb}$ of type $(\alpha \to \alpha') = \mathsf{type}(\mathsf{f})$, an **interpretation** function $\varsigma(\mathsf{f}) : \alpha \to \alpha'$.
   The cardinality function $\# : \mathsf{Pow}(\alpha) \to \mathbb{Q}_\infty$ must be interpreted to map a finite set to its cardinality, and an infinite set to ∞.
4. For each predicate symbol $\mathsf{P} \in \mathsf{PSymb}$ of type $\alpha = \mathsf{type}(\mathsf{P})$, an **interpretation** function $\varsigma(\mathsf{P}) : (\mathsf{Time} \times \mathsf{Place} \times \alpha) \to \mathsf{Bool}$.
   The set membership predicate symbol $\in : \alpha \times \mathsf{Pow}(\alpha)$ must be interpreted as actual set membership, which maps (x, X) to $\top$ if $x \in X$ and to $\bot$ if $x \notin X$.
5. A function $\varsigma(\mathtt{msg}) : (\mathsf{Time} \times \mathsf{Place} \times \mathsf{Place} \times \mathsf{Data}) \to \mathsf{Bool}$.

Remark 2.2.2 (Comments on the model).

1. Intuitively, a model is shaped like a rectangle, with places $p \in \mathsf{Place}$ along the x-axis and times $n \in \mathsf{Time}$ along the y-axis, as illustrated in Figure 3.
2. The meaning $\varsigma(\mathsf{f})$ of a function symbol $\mathsf{f} \in \mathsf{FSymb}$ *does not* vary over time and space, in the sense that if $\mathsf{f} : \alpha \to \alpha'$, then $\varsigma(\mathsf{f})$ is just a function in $\alpha \to \alpha'$.
   Intuitively, this reflects the fact that functions represent things that we do to data (like 'addition' or 'multiplication'). For example, if 3! returns 6 here and now, then 3! should return 6 tomorrow and somewhere else.
3. The meaning $\varsigma(\mathsf{P})$ of a predicate symbol $\mathsf{P} \in \mathsf{PSymb}$ *does* vary over time and space, in the sense that if $\mathsf{P} : \alpha$, then $\varsigma(\mathsf{P})$ is a function in $(\mathsf{Time} \times \mathsf{Place} \times \alpha) \to \mathsf{Bool}$.

$$
\begin{aligned}
[\![d]\!]_{n,p,\varsigma} &= d \\
[\![\mathsf{f}(t_1,\ldots,t_{\mathsf{ar}(\mathsf{f})})]\!]_{n,p,\varsigma} &= \varsigma(\mathsf{f})([\![t_1]\!]_{n,p,\varsigma},\ldots,[\![t_{\mathsf{ar}(\mathsf{f})}]\!]_{n,p,\varsigma}) \\
[\![\texttt{here}]\!]_{n,p,\varsigma} &= p \\
[\![\texttt{now}]\!]_{n,p,\varsigma} &= n \\
[\![\{a:\alpha \mid \phi\}]\!]_{n,p,\varsigma} &= \{x \in \alpha \mid n,p \vDash_\varsigma \phi[a{:=}x]\}
\end{aligned}
$$

Above, f ranges over function symbols (Definition 2.1.8(1)); ϕ ranges over predicates (Figure 2); and x ranges over data (times, places, extended rationals, or any other data).

Figure 4. Meaning of terms (Definition 2.2.5)

Intuitively, this reflects that predicates represent things that we say *about* data, and this can depend on time and place. For example, we might assert 'it's raining' here (in London) and now (today), and then 'it's not raining' tomorrow.[7]

4. Suppose $n \in \mathsf{Time}$ and $p, p' \in \mathsf{Place}$ and $x \in \alpha$. Then intuitively,

$$\varsigma(\texttt{msg})(n,p,p',x) = \mathsf{T}$$

means that at time n, p sends a message to p' with payload x.
We do not need to worry about multiple message types here — we only need one `msg` — because we gave ourselves a datatype MessageType of message types in Definition 2.1.5(8), so α can have the form $\mathsf{MessageType} \times \alpha'$ (if we wish) so that the payload $x = (m, x')$ contains information about what type of message it is.
In this model of message-passing, messages arrive instantly and reliability (no latency; no message loss; no corruption of payload). Obviously, this is a (deliberate) simplification of real-world behaviour.

Definition 2.2.3. Suppose ς is a model (Definition 2.2.1) and $n \in \mathsf{Time}_\varsigma$. Then we define:

1. $\varsigma{\downarrow^n}$ is the model obtained by restricting ς to times until n, including n.
2. $\varsigma{\uparrow_n}$ is the model obtained by restricting ς to times from n, including n.

Remark 2.2.4. *Pedant point:* We distinguish in Figure 5 between *premises* above a line, that are part of the derivation-tree, and *side-conditions* above a line, that are not part of the derivation-tree and which we put in brackets.[8] For example in

[7] Bonus fact: 'rainy London' is a myth. It rains more in Miami and Orlando. Plus, London doesn't get hurricanes.

[8] Sometimes, this distinction can really matter. True story: the first author once got a paper rejected because the referees didn't understand this distinction.

$$\frac{\begin{array}{l}A = \{x \in \alpha \mid n, p \vDash_\varsigma \phi[a{:=}x]\} \\ B = \{x \in \alpha \mid n, p \vDash_\varsigma \phi[a{:=}x] \wedge \psi[a{:=}x]\} \\ (\#B/\#A \geq r)\end{array}}{n, p \vDash_\varsigma \prod^r a : \phi.\psi}\ (\text{PiValid}) \qquad \frac{\varsigma(\mathsf{P})(n, p, [\![t]\!]_{n,p,\varsigma})}{n, p \vDash_\varsigma \mathsf{P}(t)}\ (\text{PValid})$$

$$\frac{\varsigma(\mathtt{msg})(n, p', p, [\![t]\!]_{n,p,\varsigma})}{n, p \vDash_\varsigma \mathtt{in}_{p'}(t)}\ (\text{inValid}) \qquad \frac{\varsigma(\mathtt{msg})(n, p, p', [\![t]\!]_{n,p,\varsigma})}{n, p \vDash_\varsigma \mathtt{out}_{p'}(t)}\ (\text{outValid})$$

$$\frac{[\![t]\!]_{n,p,\varsigma} \in [\![t']\!]_{n,p,\varsigma}}{n, p \vDash_\varsigma t \in t'}\ (\in\text{Valid}) \qquad \frac{(n \geq \#\mathsf{Time})}{n, p \vDash_\varsigma \phi}\ (\text{tOFlow})$$

$$\frac{[\![t]\!]_{n,p,\varsigma}, p \vDash_\varsigma \phi \quad ([\![t]\!]_{n,p,\varsigma} \in \mathbb{N})}{n, p \vDash_\varsigma @_t\phi}\ (\text{tValid}) \qquad \frac{n, [\![t]\!]_{n,p,\varsigma} \vDash_\varsigma \phi \quad ([\![t]\!]_{n,p,\varsigma} \in \mathsf{Place})}{n, p \vDash_\varsigma @_t\phi}\ (\text{pValid})$$

$$\frac{n, p \vDash_{\varsigma\uparrow_n} \phi}{n, p \vDash_\varsigma \mathtt{BOU}(\phi)}\ (\mathtt{BOUValid}) \qquad \frac{n, p \vDash_{\varsigma\downarrow^n} \phi}{n, p \vDash_\varsigma \mathtt{EOU}(\phi)}\ (\mathtt{EOUValid})$$

Figure 5. Valid sequents (Definition 2.2.5)

(tValid), $[\![t]\!]_{n,p,\varsigma}, p \vDash_\varsigma \phi$ is a premise of the rule, whereas $[\![t]\!]_{n,p,\varsigma} \in \mathbb{N}$ is not: it is a side-condition which must be satisfied in order for the instance of the rule to be valid.

Definition 2.2.5. Suppose that:

1. ς is a model.
2. $n \in \mathsf{Time}$ is a time.
3. $p \in \mathsf{Place}$ is a place.

Suppose that t is a closed term and ϕ is a closed predicate (ones having no free variable symbols). Then we define

1. $[\![t]\!]_{n,p,\varsigma}$ the **denotation** of t as per the rules in Figure 4, and
2. the set of **valid sequents** $n, p \vDash_\varsigma \phi$ in Figure 5.

If $n, p \vDash_\varsigma \phi$ is valid, then we say that ϕ **holds** at time n and place p.

Remark 2.2.6 (Comments on the rules). We discuss the rules in Figure 5:

1. *Rule* ($\in$Valid)*:*
   This is routine: $t \in t'$ holds at time n and place p when the denotation of t at time n and place p is an element of the denotation of t' at time n and place p.
2. *Rule* (PValid)*:*
   $\mathsf{P}(t)$ holds at time n and place p when $\varsigma(\mathsf{P})(x)$ returns T at time n and place p, where x is the value denoted by t at time n and place p.
3. *Rules* (inValid) *and* (outValid)*:*
   Rules (inValid) and (outValid) specify that:

   - $\mathtt{in}_{p'}(t)$ holds at time n and place p, when p receives a message from p' at time n with payload t, and
   - $\mathtt{out}_{p'}(t)$ holds at time n and place p, when p sends a message to p' at time n with payload t.

   Thus, the message-passing relation $\varsigma(\mathtt{msg})(n, p, p', x)$ in the model manifests itself in the predicate language as two modalities $\mathtt{in}_p(t)$ and $\mathtt{out}_p(t)$, as per Remark 2.3.1.
4. *Rule* (tOFlow)*:*
   This rule can only be triggered if Time is a *finite* initial segment of $\mathbb{N}$; if Time is infinite, i.e. if time is all of $\mathbb{N}$, then (tOFlow) cannot be used, because $n \geq \#\mathsf{Time}$ is impossible.

If n is a time greater than or equal to $\#\mathsf{Time}$ — e.g. suppose time is finite and we try to evaluate $@_{\mathtt{now+1}}\bot$ on the final day of time — then every predicate ϕ is deemed valid (including $\bot$). We call this a **time overflow**. This is expressed by rule (tOFlow).
In contrast, if n denotes a negative time — as might happen if for example we try to evaluate $@_{\mathtt{now-1}}\top$ on day 0 — then every predicate is deemed invalid (including $\top$). We call this a **time underflow**. This is expressed by the *absence* of any rule that could derive a valid sequent for the form $n, p \vDash_S \phi$ where $n < 0$.
Why should we deem everything to hold after the end of time (even false), and nothing to hold before the beginning of time (even true)? There is a specific technical reason for this: typically service commitments have the form

> 'if something happens today, I will do something in the future'

or

> 'if something happened in the past, then I must do something now'.

By letting time underflows be invalid (even $\top$) and time overflows be valid (even $\bot$), this makes service commitments fail gracefully at the edge cases, in the sense that they become trivially true.
The price paid is that a service commitment of the form 'if something happens tomorrow, I will do something today' may fail ungracefully. If this becomes an issue then we can reevaluate our design choices, but so far pushing ungraceful behaviour into this corner case seems to be helpful, because no examples of interest (that we have seen so far) actually use it.

5. *Rule* (pValid)*:*
Suppose that at time n and place p, t denotes a place p'. Then $@_t\phi$ holds at time n and place p, if ϕ holds at time n and place p'. Thus, $@_t\phi$ 'teleports' us to t.
6. *Rule* (tValid)*:*
Suppose that at time n and place p, t denotes a natural number n'. Then $@_t\phi$ holds at time n and place p, if ϕ holds at n' and place p.
Note the subtlety here that n' need not be a time; n' could be greater than $\#\mathsf{Time}$, or n' could be negative.

   - If n' is negative then by design our rules will not derive a valid sequent, so in this case $@_t\phi$ cannot hold at time n and place p.
   - If n' is greater than $\#\mathsf{Time}$ then $n', p \vDash_S \phi$ holds by rule (tOFlow), so $n, p \vDash_S @_t\phi$ holds.
   - If $n' \in \mathsf{Time}$ then the sequent $n', p \vDash_S \phi$ may or may not be valid, just depending on the rest of the rules.

7. (BOUValid) *and* (EOUValid)*:*

   - EOU is a modality that ends time when it is invoked (thus deleting the future). EOU stands for 'End Of Time' (a riff on 'EOF' for 'End Of File' and 'EOS' for 'End Of String').
   - BOU is a modality that begins time when it is invoked (thus deleting the past). BOU stands for 'Beginning Of Time' (a riff on 'BOF' for 'Beginning of File').

   Both modalities work by truncating (restricting) the model, as per Definition 2.2.3.
8. *Rule* (PiValid)*:*
   Intuitively, $\prod^r a : \phi.\psi$ holds at time n and place p when the assertion

   '$\phi[a{:=}v]$ implies $\psi[a{:=}v]$'

   holds for *at least* $r * 100$ percent of the values v that make $\phi[a{:=}v]$ hold. It may hold for more, but it cannot hold for less.
   So if $\phi[a{:=}v]$ holds for 10 values of v, and from those values $\psi[a{:=}v]$ holds for 9 values, then $\prod^{0.9} a : \phi.\psi$ and $\prod^{0.8} a : \phi.\psi$ hold, and $\prod^{1} a : \phi.\psi$ does not. $\prod^{1} a : \phi.\psi$ is equivalent to what we would normally write just as $\forall a.\phi \Rightarrow \psi$.
   More discussion of this is in Remark 2.4.1.

Remark 2.2.7 (Design choices for time). We consider some design choices in Figure 4 having to do with how time (and place) are treated:

1. We continue a discussion from Remark 2.2.2(2&3).
   The meaning $\varsigma(\mathsf{f})$ of a function symbol f in the clause

   $$[\![\mathsf{f}(t_1,\ldots,t_{\mathsf{ar}(\mathsf{f})})]\!]_{n,p,\varsigma} = \varsigma(\mathsf{f})([\![t_1]\!]_{n,p,\varsigma},\ldots,[\![t_{\mathsf{ar}(\mathsf{f})}]\!]_{n,p,\varsigma})$$

   in Figure 4 does not depend on time or place. Put another way, the clause is *not* this:

   $$[\![\mathsf{f}(t_1,\ldots,t_{\mathsf{ar}(\mathsf{f})})]\!]_{n,p,\varsigma} = \varsigma(\mathsf{f})(n,p,[\![t_1]\!]_{n,p,\varsigma},\ldots,[\![t_{\mathsf{ar}(\mathsf{f})}]\!]_{n,p,\varsigma})$$

   Contrast with rule (PValid) for P in Figure 5, which does let the meaning of P vary in time and place.
2. $[\![t]\!]_{n,p,\varsigma}$ in Figure 4 interprets a term r in the context of a time n, a place p, and an interpretation ς. Thus as per Figure 4

   - $[\![\texttt{here}]\!]_{n,p,\varsigma} = p$, reflecting that 'here' means 'our current location', and
   - $[\![\texttt{now}]\!]_{n,p,\varsigma} = n$, reflecting the fact that 'now' means 'our current time'.

Note that participants in a distributed system may not have access to a global clock (they may have local clocks, but there is no guarantee that these are synchronised [Lam78]), so $[\![\text{-}]\!]_{n,p,\varsigma}$ evaluates a reality which is known to an omniscient observer, but which may not necessarily be directly accessible to local observers within the system.
We could model local clocks of participants in various ways, e.g. just by providing a constant-symbol along with suitable axioms (e.g. an axiom that local time should we within some ϵ of 'real' time).

3. Concepts such as 'today' (for whatever this means to a participant) can be expressed manually by computations on time, e.g. designing a predicate to describe the times between the latest previous midnight and the earliest next midnight (whatever that means).

2.3. The *in* and *out* modalities

Remark 2.3.1. Our predicate syntax from Figure 2 features predicate-formers $\mathtt{in}_p(t)$ and $\mathtt{out}_p(t)$. These are two natural modal operators (arguably *the* two natural modal operators) corresponding to the message-passing primitive $\varsigma(\mathtt{msg})$ in the model from Definition 2.2.1(5).

There is of course much precedent for this: for instance recall how the accessibility relation on possible worlds in Kripke models gives rise to two modalities $\Box$ and $\Diamond$ [BRV01, Definition 1.9].

A simple lemma expresses the connection between 'in' and 'out':

Lemma 2.3.2. *Suppose* $n \in \mathsf{Time}$ *and* $p, p' \in \mathsf{Place}$ *and* t *is a term and* d *is a datum of the same type as* t*, and suppose* ς *is a valuation. Then:*

1. $n, p, \varsigma \vDash \mathtt{out}_{p'}(d)$ *if and only if* $n, p', \varsigma \vDash \mathtt{in}_p(d)$.
2. $n, p, \varsigma \vDash \mathtt{out}_{p'}(t)$ *if and only if* $n, p', \varsigma \vDash \mathtt{in}_p(@_p t)$.

Proof. We consider each part in turn:

1. We check the derivation rules in Figure 5 and see that both hold if and only if $\varsigma(\mathtt{msg})(n, p, p', [\![t]\!]_{n,p,\varsigma})$.
2. From part 1 of this result, noting that $[\![@_p t]\!]_{n,p',\varsigma} = [\![t]\!]_{n,p,\varsigma}$. □

Remark 2.3.3. In view of Lemma 2.3.2, the reader might wonder if we can express *out* using *in*, analogously to how (for example) we can write $P \vee Q = \neg(\neg P \wedge \neg Q)$ or $\Diamond P = \neg\Box\neg P$.

It is true that we can express $\mathtt{out}_q(x)$ using $\mathtt{in}_p(x)$, but this turns out to be quite unnatural:

$$\mathtt{out}_q(x) \quad \text{is} \quad \exists p.p{=}\mathtt{here} \wedge @_q\mathtt{in}_p(x).$$

Above, $\exists p$ is quantification over places, and $=$ is an equality relation on places.

The situation here is not as for $P \vee Q = \neg(\neg P \wedge \neg Q)$ or $\Diamond P = \neg\Box\neg P$, because $\exists p.p{=}\mathtt{here}\wedge@_q\mathtt{in}_p(x)$ is *more complicated* than $\mathtt{out}_q(x)$ (involving two powerful constructs; existential quantification over places and equality) whereas $\neg(\neg P \wedge \neg Q)$ and $\neg\Box\neg P$ are longer than but not more complicated than $P \vee Q$ and $\Diamond P$ respectively.

Thus, we can say that *in* and *out* are both natural primitives in our logic.

By similar arguments we can see why we use the modal-flavoured *in* and *out* instead of referring directly to the underlying message-passing relation in the model. Suppose we assume a binary predicate $\mathtt{msg}$ in the syntax that gets interpreted directly as $\varsigma(\mathtt{msg})$. Then we can express $\mathtt{in}_p(x)$ and $\mathtt{out}_q(x)$ as follows:

$$\begin{array}{l}\mathtt{in}_p(x) \quad \text{is } \exists q.q{=}\mathtt{here} \wedge \mathtt{msg}(\mathtt{now}, p, q, x) \quad \text{and}\\ \mathtt{out}_q(x) \text{ is } \exists p.p{=}\mathtt{here} \wedge \mathtt{msg}(\mathtt{now}, p, q, x).\end{array}$$

One reason that modal logic is useful is that, as we see above, modalities can be clearer and more convenient than direct reference to an underlying relational structure, if we want to talk about properties from the point of view of points in the model — which is what we will mostly want to do in this document.

2.4. Pragmatic implication

Remark 2.4.1. A **pragmatic implication**[9] has the form

$$\textstyle\prod_c^r a : \phi.\psi$$

where

- a is a variable symbol of type α,
- ϕ and ψ are predicates,
- $r \in [0, 1]$ is a **ratio**, and
- $c \geq 0$ is a **threshold**.

To evaluate $n, p \vDash_\varsigma \prod_c^r a : \phi.\psi$, we define

$$\begin{array}{l}A = \{x \in \alpha \mid n, p \vDash_\varsigma \phi[a{:=}x]\} \quad \text{and}\\ B = \{x \in \alpha \mid n, p \vDash_\varsigma \phi[a{:=}x] \wedge \psi[a{:=}d]\} = \{x \in A \mid n, p \vDash_\varsigma \psi[a{:=}x]\}\end{array}$$

and we proceed as follows:

[9] The Π notation quotes the standard notation for dependent product (permalink), which is a generalised implication.

- If $\#A < c$ then we return $\top$.
- If $\#A \geq c$ and $\#B/\#A \geq r$ then we return $\top$.
- If $\#A \geq c$ and $\#B/\#A < r$ then we return $\bot$.

The pragmatic implication treated in Figure 5 lacks the threshold c, but we shall see in Examples 2.4.3(6) that the version with the threshold is expressible using the version without.

Some notation will be useful in Example 2.4.3 and later:

NOTATION 2.4.2. Suppose that $msgtyp \in \mathsf{MessageType}$ is a message type and suppose $p \in \mathsf{Place}$ is a place. Define syntactic sugar $\mathsf{msgtypIn}_p(t)$ and $\mathsf{msgtypOut}_p(t)$ and $\mathsf{msgtypIn}_p()$ and $\mathsf{msgtypOut}_p()$ as follows:

$$\begin{array}{lll} \mathsf{msgtypIn}_p(t) & \text{is} & \mathtt{in}_p(msgtyp, t) \\ \mathsf{msgtypOut}_p(t) & \text{is} & \mathtt{out}_p(msgtyp, t) \\ \mathsf{msgtypIn}_p() & \text{is} & \mathtt{in}_p(msgtyp) \\ \mathsf{msgtypOut}_p() & \text{is} & \mathtt{out}_p(msgtyp). \end{array}$$

EXAMPLE 2.4.3.

1. $\prod^1_0 a : \phi.\psi$ expresses a meaning that we would write in normal predicate logic as $\forall a.\phi \Rightarrow \psi$. Thus, we can express $\phi \Rrightarrow \psi$ just as
$$\phi \Rrightarrow \psi \quad \text{is} \quad \prod^1_0 a : \phi.\psi$$
for a a fresh variable symbol that is not free in ϕ or ψ.
2. $\top$ is just $\bot \Rrightarrow \bot$ and $\neg\phi$ is just $\phi \Rrightarrow \bot$.
3. $\forall a.\phi$ is just $\prod^1_0 a : \top.\phi$ and $\exists a.\phi$ is just $\neg\forall a.\neg\phi$.
4. Suppose ϕ and ϕ' are predicates. Define **disjunction** $\phi \vee \phi'$ and **conjunction** $\phi \wedge \phi'$ by
$$\phi \vee \phi' \quad \text{is} \quad (\neg\phi) \Rrightarrow \psi \qquad\qquad \phi \wedge \phi' \quad \text{is} \quad \neg(\neg\phi \vee \neg\phi').$$
5. Suppose a is a variable symbol with type α and t is a term with type $\mathsf{Pow}(\alpha)$. Then define **bounded existential and universal quantification** $\exists a{\in}t.\phi$ and $\forall a{\in}t.\phi$ by
$$\exists a{\in}t.\phi \quad \text{is} \quad \exists a.a{\in}t \wedge \phi \qquad\qquad \forall a{\in}t.\phi \quad \text{is} \quad \forall a.a{\in}t \Rrightarrow \phi.$$

6. **Threshold.**
   To express $\prod^r_c \phi.\psi$ using $\prod^r \phi.\psi$, we just write

$$\left(\exists a_1, \ldots, a_c. \bigwedge_{1 \le i < j \le c} (\phi[a{:=}a_i] \wedge a_i \neq a_j)\right) \Rightarrow \prod^r_\phi .\psi.$$

   In practice we would want to offer the user the version with the threshold as a primitive, but the above encoding shows that this makes no difference to expressivity.
7. To express that over the next 1000 time steps (not including now) we will respond to a fraction of (at least) 0.9 of ping requests, assuming we get at least 10 ping requests, we could write

$$\mathsf{pingCommit} \;=\; \prod^{0.9}_{10} n', q : \mathtt{now} < n' \le \mathtt{now}{+}1000 \wedge @_{n'} \mathsf{pingIn}_q().@_{n'+1} \mathsf{pongOut}_q().$$

   As per Notation 2.4.2, $\mathsf{pingIn}_q()$ above is shorthand for $\mathtt{in}_q(\mathit{ping})$ where ping : MessageType, and similarly for $\mathsf{pongOut}_q()$. We also assume above that q can send at most one ping message to p per timestamp.
8. To express that over the next 1000 time steps (not including now) we will relay a fraction of 0.9 of messages of type m : MessageType from a particular q, to a particular q', assuming we get at least 10 such requests, we could write

$$\mathsf{relayCommit}_{q \to q'} \;=\; \prod^{0.9}_{10} n' : \mathtt{now} < n' \le \mathtt{now}{+}1000 \wedge @_{n'} \mathsf{mIn}_q(v).@_{n'+1} \mathsf{mOut}_{q'}(v).$$

   As per Notation 2.4.2, $\mathsf{mIn}_q(v)$ is shorthand for $\mathtt{in}_q((M, v))$, and similarly for $\mathsf{mOut}_q(v)$.
9. To express that over the next 1000 time steps (not including now) we will relay a fraction of 0.9 of any type of message from any q, to a particular q', assuming we get at least 10 such requests, we could write

$$\mathsf{relayCommit}_{* \to q'} \;=\; \prod^{0.9}_{10} n', q, v : \mathtt{now} < n' \le \mathtt{now}{+}1000 \wedge @_{n'} \mathtt{in}_q(v).@_{n'+1} \mathtt{out}_{q'}(v).$$

10. **Composition of service commitments:** A service commitment is just a predicate. Composition of service commitments comes 'for free' because predicates can be combined using logical connectives, like $\Rightarrow$.
    For example, to express that if q' will commit to pingCommit then we will provide $\mathsf{relayCommit}_{* \to q'}$, we could write:

$$(@_{q'} \mathsf{pingCommit}) \Rightarrow \mathsf{relayCommit}_{* \to q'}.$$

11. **Rate limiting:** To express that from now onwards, in any 1000 units of time at most 10 pings arrive, we could write:

$$\forall n.n \geq \mathtt{now} \Rightarrow @_n \textstyle\prod_{11}^{1} n' : n \leq n' < n{+}1000 \wedge @_{n'} \exists q.\mathsf{pingIn}_q.\bot.$$

To express that there are at most 10 datums, just use existential quantification and inequality checks.

2.5. Example: the PingCommit service commitment

Some notation will be useful:

Notation 2.5.1. Suppose ϕ is a predicate.

1. Define a **tomorrow** modality $\mathsf{tomorrow}(\phi)$ by:

$$\mathsf{tomorrow}(\phi) \quad \text{is} \quad @_{\mathtt{now}+1}\phi.$$

2. Define a **henceforth** (or **from today**) modality $\mathsf{henceforth}(\phi)$ by:

$$\mathsf{henceforth}(\phi) \quad \text{is} \quad \forall n{\in}\mathsf{Time}.n{\geq}\mathtt{now} \Rightarrow @_n\phi.$$

Example 2.5.2. Let's unpack the meaning of $n, p \vDash_\varsigma \mathsf{tomorrow}(\phi)$:

1. $n, p \vDash_\varsigma \mathsf{tomorrow}(\phi)$ is $n, p \vDash_\varsigma @_{\mathtt{now}+1}\phi$.
2. $n, p \vDash_\varsigma @_{\mathtt{now}+1}\phi$ is $[\![\mathtt{now}{+}1]\!]_{n,p,\varsigma}, p \vDash_\varsigma \phi$.
3. $[\![\mathtt{now}{+}1]\!]_{n,p,\varsigma} = [\![\mathtt{now}]\!]_{n,p,\varsigma}{+}1 = n{+}1$.
4. Thus, $n, p \vDash_\varsigma \mathsf{tomorrow}(\phi)$ is valid just when $n{+}1, p \vDash_\varsigma \phi$.

Thus we can say

$\mathsf{tomorrow}(\phi)$ is true precisely when ϕ is true tomorrow.

Example 2.5.3. We can now express the PingCommit service commitment. It conveniently illustrates how we can use our machinery to make a very simple, but still meaningful, service commitment. We continue Example 2.4.3(7) and assume message types $\mathit{ping}, \mathit{pong} \in \mathsf{MessageType}$. Then for a participant $p \in \mathsf{Place}$:

1. p can express that henceforth, if it receives a ping message from q then it will send a pong message to q' tomorrow, as follows:

$$\mathsf{PingCommit} \;=\; \mathsf{henceforth}\big(\mathtt{in}_q(\mathit{ping}) \Rightarrow \mathsf{tomorrow}(\mathtt{out}_{q'}(\mathit{pong}))\big).$$

We can use our sugar from Notation 2.4.2 to put this more succinctly:

$$\mathsf{PingCommit} \;=\; \mathsf{henceforth}\big(\mathsf{pingIn}_q() \Rightarrow \mathsf{tomorrow}(\mathsf{pongOut}_{q'}())\big).$$

2. To express that p commits to PingCommit at some time n, we assert a judgement

$$n, p \vdash \mathsf{PingCommit}.$$

This judgement (and the commitment it expresses) is *valid* in a model ς when

$$n, p \vDash_{\varsigma} \mathsf{PingCommit}.$$

2.6. Expert mode

Pragmatic implication seems to be enough for now, but users may want to express more subtle notions of implication, e.g. based on information about behaviour that has been gossiped from other participants. For this, we need to be able to collect events, pass them as data, and (most likely) do arithmetic on their cardinalities. Accordingly, our logic allows us to collect instances of particular ϕ as

$$\{d \in \alpha \mid n, p \vDash_{\varsigma} \phi[a{:=}d]\} : \mathsf{Pow}(\alpha).$$

We can recover pragmatic implication $\prod^{r} a : \phi.\psi$ using collection (and the connectives of propositional logic) as follows:

$$\#\{a \mid \phi\} \neq 0 \implies \#\{a \mid \phi \wedge \psi\}/\#\{a \mid \phi\} \geq r.$$

Participants are free to collect sets as data $S : \mathsf{Pow}(\alpha)$, pass these around as data to other participants, for sophisticated participants to 'roll their own' variants of pragmatic implication (or whatever else they find useful, since sets collection is very expressive).

2.7. Using EOU and BOU to evaluate service commitments

Recall from Remark 2.2.6(7) that EOU and BOU truncate the model, deleting the future and the past respectively. This can be useful for expressing that predicates are valid up to a certain point or from a certain point, which is a common practical use-case. For example, the expression 'so far so good' can be expressed as EOU(good) (for a suitable predicate expressing 'good').

Consider participants p and p', a predicate) ϕ, and times $n \leq n'$. Then p' may be interested in validity of the following:

1. p has kept a service commitment that ϕ made at time n *up to now*.
   $n', p \vDash_{\varsigma} \mathtt{EOU}(@_n(\phi))$.
2. p has not kept a service commitment ϕ make at time n.
   $n', p \vDash_{\varsigma} \mathtt{EOU}(\neg @_n(\phi))$.
3. p has mostly (at least 0.9 of the time) kept a service commitment $\mathsf{henceforth}(\phi)$, made at time n.
   $n', p \vDash_{\varsigma} \mathtt{EOU}(\prod^{0.9} n'' : n'' \geq n.@_{n''}\phi)$.

2.8. How to express offers, using axioms

In practice, commitments are not made arbitrarily. Typically, an offer is sent out to some set of eligible participants, and the offer is available only to a limited number of free slots.

A real-life example is: an online retailer advertising 50 items for sale, who will not post abroad. Thus the offer is available to (at most) any fifty from the set of all participants in the same country as the retailer.

It is useful to fix some terminology:

1. A **predicate** is a statement in a formal language that can be evaluated to true or false. In this document, we are specifically interested predicates in the formal language in Figure 2.
2. A **commitment** a predicate for which there is some social or technical agreement that this predicate should evaluate to true.[10]
3. An **offer** is a mechanism by which a predicate is converted into a commitment. Thus, an offer is a formal mechanism for committing to what a predicate expresses.

We now study how we can we express *offers* within message logic. Specifically: we study the example of offering to commit to ϕ, to at most $m \geq 0$ from a set $P \subseteq \mathsf{P}$ participants on a first-come first-served basis.

We assume a tuple of data

$$(a, b, \phi, P, m, \mathsf{accept}_{[\mathsf{a}.\phi]}, \mathsf{offer}_{[a.\phi]})$$

as follows:

1. ϕ is a predicate, $a : \mathsf{Place}$ is a variable of type Place, and $b : \beta$ is a variable of some other type. Here:
   - ϕ represents a commitment which we are offering to make.
   - a, which may be free in ϕ, represents the participant to whom we make the commitment.
   - b, which may be free in ϕ, represents some configuration data for the commitment (for this discussion, we do not care what it is).
2. $P \subseteq \mathsf{Place}$ is a set of **candidate promisees**.
   Intuitively, if $p \in P$ then we are willing to commit to $\phi[a{:=}p]$, subject to availability.

[10] Note the word 'should' here, rather than 'must'. A predicate that *must* be true is an axiom; that is a different idea again.

3. $m : \mathbb{N}_{\geq 0}$ is a number, which we call the **number of free slots**.
   We are willing to commit to ϕ for any m of the elements of P (and possibly fewer, if fewer ask, but no more).
4. $\mathsf{accept}_{[\mathsf{a}.\phi]} : \mathsf{MessageType}$ is a message type, and $\mathsf{offer}_{[a.\phi]} : \mathsf{Pow}(\mathsf{Place}) \times \mathbb{N}_{\geq 0}$ is a predicate symbol.
   Note that $\mathsf{accept}_{[\mathsf{a}.\phi]}$ and $\mathsf{offer}_{[a.\phi]}$ are atomic tokens in our syntax; the subscripts $a.\phi$ are part of how we write them to remind us that we consider these tokens to be associated to ϕ, but this association just exists in our mind. As far as the mathematics is concerned, these are just two symbols.

Then we can impose an axiom on $\mathsf{offer}_{[a.\phi]}$ as follows:

$$\begin{aligned}
&\forall P : \mathsf{Pow}(\mathsf{Place}).\forall m : \mathbb{N}_{\geq 0}.m \geq 1 \implies \\
&\qquad \mathsf{offer}_{[a.\phi]}(P, m) \implies (\exists p{\in}P, y : \beta.\mathtt{in}_p(\mathsf{accept}_{[\mathsf{a}.\phi]}, y) \implies \\
&\qquad\qquad \exists p{\in}P, y : \beta.\Big(\mathtt{in}_p(\mathsf{accept}_{[\mathsf{a}.\phi]}, y) \wedge \\
&\qquad\qquad\qquad \mathsf{tomorrow}\big(\phi[a{:=}p, b{:=}y] \wedge \mathsf{offer}_{[a.\phi]}(P\backslash\{p\}, m\text{-}1)\big)\Big)
\end{aligned}$$

Above, $\implies$ represents (normal logical) implication, and $\exists p{\in}P$ represents bounded quantification, as per Example 2.4.3(1&5). This asserts that at each timestep, if we have offered ϕ and at least one participant sends us a message accepting this offer (with some configuration y), then tomorrow we commit to ϕ for *one* of the participants who messaged in.

We have used tomorrow just as an example: any other time modality (including none at all) would be valid depending on what we want to express. We could also accept multiple offers at a given timestep; again, it would just be a matter of writing up the relevant predicate.

Remark 2.8.1. Even though we have shown how to express an offer as an axiom (about a predicate called $\mathsf{offer}_{[a.\phi]}$), this does not mean that an offer *is* an axiom in the sense of pure mathematics.

In mathematics, if a model does not satisfy the axioms of (say) being a group, then it is not a group. This is by definition; there is no debate about it.

In our model of service commitments, if a model does not satisfy the axiom for $\mathsf{offer}_{[a.\phi]}$, this does not necessarily mean that the offer has not been made. It could also mean that the offer was made but not fulfilled. There is an *additional step* here, where participants in the distributed system decide how serious the breach of this commitment is; conversely even if an offer is made and fulfilled, participants in the distributed system might decide for one reason or another

that they are still not satisfied. See the discussions in Subsections 3.2.6 and 3.3.3 and Remark 3.3.1.

3. The operational framework

So far, we have used message logic to *describe* service commitments: now we will discuss how we could *apply* it to help us implement decision-making and coordination in practical large scale distributed systems.

1. In Subsection 3.1 we outline the differences between our logic and the real world, and explain why the differences make sense.
2. In Subsection 3.2 we explain the seven stages in the life cycle of a service commitment in a distributed system.
3. In Subsection 3.3 we spell out application scenarios, portraying how participants can use service commitments for local decision-making.

3.1. Differences between our logic and the real world

Logic is by design an abstraction of some (usually messy) reality. It is important and useful to abstract — but it is also important to remember what we have abstracted, why, and to what extent realistic details could be put back into the logical model if required. We consider a list of such differences next:

1. *Messages have latency and may get lost:*

   - In the logic, messages arrive instantly and do not go missing.
   - In the real world, messages take time to arrive (i.e., they have latency) and may get lost.

   The logic is a reasonable first approximation to the real world, but a more detailed model could include aspects of latency and possibly message loss. Including this would not be technically difficult.

2. *Observability:*

   - In the logic, predicates are evaluated by an omniscient observer standing outside the model's notion of time and space.
   - In the real world, participants are not omniscient. They can only see local and partial information and are trapped in the model's notion of time and space.

An omniscient observer can stand outside the model and make specifications about what *should* happen. Participants can then reason locally about whether such a specification has actually been met, based on their local knowledge. Building infrastructure to facilitate such local reasoning remains future work.

3. *Network communication:*

    - In the logic, any participant can directly send a message to any other participant. In effect, all participants in the network are connected to each other (i.e., everyone is each other's peer).
    - In the real world, participants can only directly send messages to a small number of other nodes; longer-range communication happens through gossiping.

    Gossiping could be folded into the logic quite easily. This is future work.

4. *Information flow control:*

    - In the real world, nodes have the flexibility to express who should be able to read what part of a message by means of encryption/decryption.
    - In the logic, the omniscient observer knows everything, including any decryption keys that may be required. Thus information flow control is inherently not represented in the logic as its currently designed.

    Folding this into the logic is possible, but would add more complexity. Future work could explore information flow control in our logic model.

5. *Evaluation:*

    In Subsection 2.8 we distinguish between a *predicate*, a *commitment* (= a predicate for which there is agreement that it should be true), and an *offer* (= an offer to commit). Offers may be made to many participants, and actual commitments made to only some of them (as for example in a sale of goods that is available 'while stocks last').

    However, deciding whether breaking a commitment, or keeping it, is a good or a bad thing, may vary.[11] See the discussion of *quality evaluation* in

[11] This seems to be a significant difference between pure logic, where if we assert a predicate as true and it isn't, then that's bad; and applied logic, where we may assert a predicate as true and then, for whatever reason, change our mind.

Subsection 3.2, and the discussion with examples in Subsection 3.3 (and in particular the discussion in Subsection 3.3.3). Deciding how much of this evaluation stage to express in the logic, and how much to leave external to it, is a design question for future work.

3.2. The seven stages of a service commitment

We will now split the life cycle of a service commitment — as it works in a real-life distributed system — into seven stages as follows:

1. Offer creation.
2. Offer dissemination.
3. Offer negotiation.
4. Commitment activation.
5. Consumption.
6. Quality evaluation.
7. Termination.

We discuss each stage in turn:

3.2.1. Service offer creation

Recall from Definition 1.1.1 that a service commitment is a commitment to some pattern of messaging behavior. Any participant p can create a service commitment SC about its own future behaviour, but usually first it offers to do so:

Definition 3.2.1. A **service offer** is an announcement that a participant is willing to commit to a service commitment (Definition 1.1.1).

We need service offers for service commitments because in practice

1. a service commitment might be one half of a trade ('I'll give you a donut if you'll give me a dollar'), or
2. a service commitment, even if free, might have limited availability (e.g. 'free doughnuts! only while stocks last!').

So in practice, before p commits to SC, it usually creates a corresponding service offer SO to let other participants know what behavior p can commit to, and the terms of agreement for that commitment.

Concretely, a service offer SO is a data tuple

$$(\mathit{Promiser}, \mathit{Promisees}, \mathit{Content}, \mathit{ActivationStatus}, \mathit{ExpirationTime}, \mathit{Identifier}), \quad (1)$$

as we now describe:

1. *Promiser:* The promiser has to be p (the participant making the offer). It is important to note that p cannot make offers or commitments on behalf of others.

2. *Promisees:* The participants for whom the commitment encoded in SO is intended. This set is identified (in a standard way) by a function that inputs a participant identifier and outputs 'yes' or 'no'.[12]

3. *Content:* A predicate in message logic identifying the specific constraint on behaviour that p commits to, and the terms of agreement for a commitment.

4. *Activation Status:* The current availability status of the service offer (*active* or *inactive*). The activation status can differ per promisee.

5. *Expiration Time:* The deadline after which SO becomes invalid. The expiration time can differ per promisee.

6. *Identifier:* A unique identifier by which p or any other participant may refer to the service commitment so at p.

To make the service offer SO active (rather than it being just some data on disk), p adds SO to a special field in its memory called its **configuration**.

3.2.2. Service offer dissemination

Once a participant p has created a service offer SO to express an offer of some commitment SC, it can communicate SO to other participants as a message using established gossiping and pub-sub protocols.

Concretely, the message consists of a tuple of the fields in equation (1) in Subsection 3.2.1. p can choose to cryptographically encrypt (some of) the fields. For example, p may not want to disclose the list of promisees and encrypt the content such that only the target promisee can decrypt it (rather than all promisees or other participants). The message is cryptographically signed by p: write this $sig_p(\mathsf{SO})$.[13]

There is no restriction that $sig_p(\mathsf{SO})$ can only be communicated to participants in the *promisees* of so — such a restriction would make no sense on a gossip network! — but when $sig_p(\mathsf{SO})$ does reach a promisee q, q can update its local configuration by storing the pair (SO, p). q can also subscribe to changes in

[12] The 'always yes' function is a promise to everyone; the 'always no' function is a (vacuous) promise to nobody; a *singleton function* that returns 'yes' on just a single participant represents a promise to that one participant; and so on.

[13] We assume that signatures cannot be forged.

the service offer (e.g., deactivation or expiration time modifications) or to stay informed of any updates to the offer's state.

In some cases, participants may need to discover existing service commitments dynamically. This typically involves querying the network for commitments that match certain criteria, such as commitments involving a specific promiser, promisee, or service. The exact query language is considered future work, but one possibility would be to use Anoma intents as a basis for discovery.

3.2.3. Service offer negotiation

When a service offer SO from promiser p reaches a promisee q, q will choose precisely one of the following options:

1. *Do nothing:* Promisee q may not be interested in consuming the service offer from p.

2. *Negotiate:* Promisee q may want to negotiate the terms of the offer. This phase is valuable for q if it can get the same service for lower costs,[14] and this also provides pricing information for p to use in future offers.

3. *Request for service commitment activation:* Promisee q may agree on the terms of the service offer and request p to commit to the promised behavior.

3.2.4. Service commitment activation

When promiser p receives a service commitment activation request message from a promisee q, p can decide to:

1. *Accept the request:* Promiser p creates a service commitment SC with the same fields as the service offer SO. p sends a signed commitment activation message $sig_p(\mathsf{SC})$ to q (or initiates a three way handshake). When q receives the activation message, it updates its local configuration with SC, and adds a *Performance Status* field indicating to what extent q assesses the service commitment has been kept, or violated.

2. *Reject the request:* Promiser p simply ignores the request message.

3.2.5. Service consumption

Consuming a service commitment refers to the act of (not) engaging with the promised behavior. A promisee is not obligated to use the promised service. For example, a promisee may not be happy with the quality of a provided relay service, and decide to interact with other promisers.

[14] could be financial and/or non-financial costs

3.2.6. Service commitment evaluation

Evaluation occurs as the participant monitors whether the terms of the service commitment are being met. The system checks whether the promiser has fulfilled the obligations within the specified time frame and according to the predefined quality parameters (if any). If the promiser p successfully completes the commitment, promisee q marks service commitment as 'fulfilled'. If the promiser p fails to fulfill its commitment, q will modify the *Performance Status* field mentioned in item 1 in Subsection 3.2.4 accordingly.

We propose to use service commitments evaluation data as a 'unit of trust' in Anoma's Node Trust Architecture. More concretely: promisers will have a reputation that reflects how reliable their commitments are. Promisers that breach their commitments may lose reputation and find that nobody invokes their services in the future.

3.2.7. Service commitment termination

Service commitments are considered to be terminated in the following ways:

1. *Gracefully:* The service commitment SC is considered terminated when its expiration deadline has passed, after which phase 1 can start again.

2. *Not gracefully:* Promiser p leaves the network while it has active commitments to other participants. We rely on existing networking protocols to learn that a participant has left the network and to prevent that promisers with bad reputation can whitewash their reputation [FPCS04].

3.3. Application scenario

We now discuss application scenarios to illustrate how service commitments can be used for decision-making by participants in a distributed system.

3.3.1. An extremely simple application scenario

Consider the simple network displayed in Figure 6, having four participants: Alice, Bob, Carol and Dave.

Alice is connected to Bob and Carol; Bob is connected to Dave; and Carol is also connected to Dave. Furthermore, Bob has committed to a pragmatic implication that he will relay at least 95% of any messages he receives to Dave within 10ms, and Carol has committed to a pragmatic implication that she will relay all messages of any messages she receives to Dave within 1000ms.

Suppose that Alice wants to get a message to Dave. She has two choices:

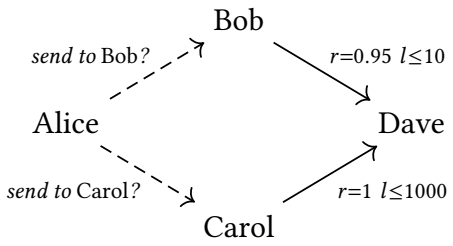

Figure 6. An extremely simple application scenario

1. She can send a message to Bob, who (if he keeps his commitment) will probably relay it to Dave quite quickly, though Bob has reserved the right to drop up to 5% of the messages.
2. She can send a message to Carol, who (if she keeps his commitment) will certainly relay it to Dave, but only commits to do so within a second.

In spite of the simplicity of this scenario, there are a few points to note:

1. The ratios $r = 0.95$ and $r = 1$ are not probabilities. It is not the case that Bob will forward messages with 95% probability. He has promised to forward at least 95% of messages, but he might forward more.
2. In the scenario as presented in Figure 6, there is no guarantee of fairness; e.g. if Bob is getting one message every second from each of 20 participants, he would fulfill his commitment if he forwards all messages from 19 of the participants and drops all messages from one of them.[15]
   We highlight this because probabilistic behaviour is assumed to be fair, however, as just discussed pragmatic implication is *not* probabilistic.
3. Bob and Carol may have committed to relay messages, but they are not necessarily forced to meet their commitments.
   We are presented with a model, and some predicates which for social reasons we may consider 'should' be true, and for which in a real-life system there may be incentives to keep them true, but this does not necessarily imply that they *must* be true. This is a feature, not a bug, which reflects the fact that in real life, commitments may be made but not met, sometimes for legitimate reasons which are understood and recognised as legitimate by the community of participants on the network, as we discuss in Remark 3.3.1.

Remark 3.3.1. A toy example of how commitments may be broken for good reasons is given in Subsection 3.3.3.

[15] More sophisticated commitments, corresponding to more elaborate predicates, are certainly possible; we just have not written them out.

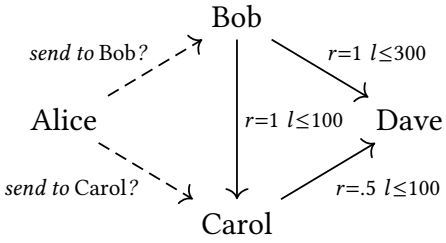

Figure 7. A simple application scenario

An example from real life is a force majeure exclusion in contracts, which frees parties from liability or obligation when an extraordinary event or circumstance beyond the control of the parties interferes with delivery of a service. Crucially, what constitutes 'extraordinary event' and 'beyond the control of the parties' is a *social* decision, not a purely legalistic, logical, or technical one. So essentially, the *force majeure* clause just expresses "we commit to this contract, unless society decides that there is a good reason not to", which is more-or-less the same idea as the service commitment evaluation step from Subsection 3.2.6.[16]

So it is important to understand that we are using logic to model service commitments, and offers of service commitments, but an interesting added dose of 'real world' is implicit in our narrative and lurking behind the logical formalism.[17]

There are predicates and axioms in our formalism, but they are *more* than *just* that.

3.3.2. A simple application scenario

This case is similar to the previous one, but here Bob has committed to forward messages to both Dave (within 300ms) and Carol (within 100ms). Again, this illustrates that the r-labels are not probabilities; they are commitments.

In this case, if Alice has to choose to whom to send a message — and assuming she has no other information than what is illustrated in the diagram, and assuming that she would prefer that more than half of her messages arrive — then she has a single best strategy: send everything to Bob. If Bob keeps all of his commitments, then Alice's messages will arrive with Dave — once (perhaps) within 200ms, and again (certainly) within 300ms.

[16] There is also a dual to this: the law recognises cases where the letter of a contract is fulfilled, and but not its spirit, and depending on circumstances this too may be taken very seriously. For instance: a retail customer might challenge a contract with a large company that contains some fine print, on the grounds that the significance of that fine print was not made sufficient clear at the time of entering into the contract.

[17] Well, interesting for a logician like the first author. To an industry practitioner like the second author, this is all rather obvious — and, if anything, it's pure mathematical axiom systems that may seem a little odd.

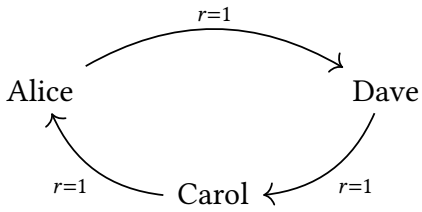

Figure 8. A silly application scenario

3.3.3. A silly case

In the silly application scenario in Figure 8, if Alice sends a message to Dave then he will relay it to Carol, who will relay it to Alice, who will relay it to Dave, and so on forever. This is clearly a pathological situation: the question is how to understand it within our logic.

There are various ways to do so, including:

1. Blame it on the fact that the agents committed with certainly (i.e. with $r = 1$) to forward their messages. That is, we pin the error on the 1; perhaps it should be (say) $r = 0.9$ instead.
   The problem with this is that we do not distinguish between 'silly' messages that should be dropped, and 'sensible' messages that should be forwarded.
2. Accept that sometimes commitments will be broken and that this may be a good thing.
   Essentially, this pushes design effort out of the logical system and into the evaluation mechanisms which agents may use in the real world to decide whether some violation of a service commitment is bothersome to them. Promises may be made and broken without penalty, if circumstances justify it.[18]
3. Extend the logic such that agents can trace the path a message has followed. Then the service commitment could be refined so that (for example) Alice commits to forward messages *that have not already passed through* Dave.

4. Conclusions

4.1. Summary

Message logic is a simple way to formally describe and reason about messages passed around a network. This is interesting and useful because message-passing

[18] This is not a hypothetical; it happens all the time in the real world. For example: if you leave your baby with a babysitter and say "Don't leave the flat", and the babysitter promises not to — and then there's a fire in the flat, then you would be furious with the babysitter if they *did* keep their promise. The point here is not that (surprise, surprise!) common sense exists; it's that the raw material of our commitments to one another, including those expressed using Message Logic, is embedded, and known to be embedded, within a social framework within which they are intended to be understood.

is the relevant observable of a distributed system, so this logical model captures a fundamental and useful aspect of reality.

Our model as it is currently set up is (deliberately) simplistic — e.g. messages are passed with zero latency and zero loss — but often that is a reasonable first approximation to real life (a detailed discussion is in Subsection 3.1).[19]

In our model:

1. Service commitments are modelled by predicates, that are valid in a model if and only if the service commitment is kept. See the example in Subsection 2.5.
2. Offers become axiomatisations, as outlined in Subsection 2.8.

This is embedded in a rich real-life context of discovery protocols, evaluation of whether commitments have been kept and corresponding reputation systems, and so on. With time and further research, we will see how much of this we want to include in the logic and how much is inherently not to do with logical formalisation and should be left as external to the model (e.g. we may want to abstract over the gossip or pub-sub protocol used).

We envisage that message logic could be applied as follows:

1. *Informally*, as a rigorous logical language for specifying service commitments (think: first-order logic for distributed systems).
2. *Formally*, for machine verification. A system of service commitments and offers could be expressed in (say) Isabelle or Agda; and correctness properties could be asserted and proved (or refuted).
3. *Programmatically*. In this scenario, Message Logic is used to express service commitments directly in an implementation. We wrote earlier that service commitments are 'modelled' by predicates, but in a programmatic setting, the service commitment would *be* a predicate in Message Logic.

Message logic is designed to seem simple, because we want it to be easy to use; however, it is also a powerful and expressive logic.

4.2. Related work

Message Logic draws inspiration from various fields in distributed systems, formal methods, and service-oriented architectures. Relevant related work is as follows:

[19] If we make a phone call, to a first approximation we assume that data will be delivered effectively with zero latency and zero loss. Sometimes that assumption is violated; the connection becomes laggy and/or unreliable. At this point we become frustrated, precisely *because* we had this assumption.

1. *Promise theory*

   Promise theory [Bur15] provides a conceptual framework for reasoning about cooperative systems through promises made by agents, which include guarantees of behavior and performance. Message Logic builds on this notion by formalising service commitments as predicates, providing a logical foundation for verifying and analyzing such commitments in distributed systems.

2. *Service coordination in distributed systems*

   Research on service coordination mechanisms, such as service-oriented architectures [PVDH07, SBD+10], protocols for consensus [Kwo14, SWvRM20, BG17], and service discovery services [PTT12, MRPM08], intersects with our work. While these protocols aim to ensure reliable coordination among distributed components, Message Logic abstracts these mechanisms into a logical framework, allowing for the formal reasoning of commitments and offers independent of implementation-specific details.

3. *Temporal and modal logics for distributed systems*

   Logics such as Linear Temporal Logic (LTL) [DLH01] and Computational Tree Logic (CTL) [BH05] have been extensively used to model and verify distributed system behaviors. These frameworks capture the temporal aspects of distributed systems but lack the expressiveness to directly model service commitments and their violations. Message Logic complements these approaches by focusing on service-level guarantees within distributed systems.

4.3. Future work

We see two natural ways to extend Message Logic: we can extend *the logic*, and we can *build infrastructure* with which to apply it.[20]

4.3.1. Extending Message Logic for real-world scenarios

The current logical framework makes some simplifying assumptions (e.g. zero-latency and no message loss; see Subsection 3.1). This is of course unrealistic, so we can consider expanding the logic in different ways (depending on what we care to model more precisely), including:

[20] Or perhaps both; e.g. a practical application may require new tooling, and it might motivate a specific logic extension.

1. *Incorporating Latency and Message Loss*: Including bounded or unbounded delays and probabilistic message loss, to reflect the inherent uncertainties of distributed environments.

2. *Local Reasoning for Participants*: Currently, an omniscient observer determines whether commitments are upheld. If we want to explicitly reason within the logic about what participants know, believe, or know or believe that other participants know or believe (and so on), then epistemic modalities are well-studied and could be added [RSW24].

3. *Dynamic Service Commitment Evolution*: Mechanisms to handle changes to service commitments (such as updates to offers or modifications of commitments during execution) would make the logic more adaptable to evolving systems.

4. *Integration with Reputation Systems*: Formalising how service commitment evaluation feeds into reputation or trust systems could enhance the practical relevance of Message Logic for real-world distributed systems with decentralised decision-making.

Note that these extensions would probably not increase expressive power in principle; they would simply make some things easier to express.

4.3.2. Building infrastructure around Message Logic

To apply Message Logic in practice, we would require infrastructure and tools. Potential directions include:

1. *Specification and verification frameworks*: Developing tools that allow practitioners to specify service commitments in Message Logic and verify their correctness automatically using theorem provers like Isabelle, Coq, or SMT solvers.

2. *Protocol representation and composition*: Building data structures and libraries that allow the expression of complete protocols (e.g., consensus algorithms, mempool mechanisms) within the logic. This would demonstrate the practical applicability of Message Logic to design and implement distributed system protocols.

3. *Simulation and monitoring Tools*: Implementing simulation environments to test distributed systems against formalised commitments. Monitoring tools could also dynamically evaluate service-level compliance during system operation.

4. *Programmatic integration*: Extending programming languages or middleware to natively support Message Logic predicates, enabling direct use of service commitments as part of distributed system implementations.

References

BG17. Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017. URL: https://arxiv.org/abs/1710.09437. (cit. on p. 34.)

BH05. Alexander Bell and Boudewijn R Haverkort. Sequential and distributed model checking of Petri nets. *International Journal on Software Tools for Technology Transfer*, 7:43–60, 2005. doi:10.1016/S1571-0661(05)80391-3. (cit. on p. 34.)

BRV01. Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001. (cit. on p. 16.)

Bur15. Mark Burgess. *Thinking in promises: designing systems for cooperation*. O'Reilly Media, Inc., 2015. (cit. on p. 34.)

DLH01. Falk Dietrich, Xavier Logean, and J-P Hubaux. Modeling and testing object-oriented distributed systems with linear-time temporal logic. *Concurrency and Computation: Practice and Experience*, 13(5):385–420, 2001. doi:10.1002/cpe.571. (cit. on p. 34.)

FPCS04. Michal Feldman, Christos Papadimitriou, John Chuang, and Ion Stoica. Free-riding and whitewashing in peer-to-peer systems. In *Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*, pages 228–236, 2004. (cit. on p. 29.)

Kwo14. Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1(11):1–11, 2014. (cit. on p. 34.)

Lam78. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978. doi:10.1145/359545.359563. (cit. on p. 16.)

MRPM08. Elena Meshkova, Janne Riihijärvi, Marina Petrova, and Petri Mähönen. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. *Computer networks*, 52(11):2097–2128, 2008. (cit. on p. 34.)

PTT12. Giuseppe Pirrò, Domenico Talia, and Paolo Trunfio. A DHT-based semantic overlay network for service discovery. *Future Generation Computer Systems*, 28(4):689–707, 2012. (cit. on p. 34.)

PVDH07. Mike P Papazoglou and Willem-Jan Van Den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB journal*, 16:389–415, 2007. doi:10.1007/s00778-007-0044-3. (cit. on p. 34.)

RSW24. Rasmus Rendsvig, John Symons, and Yanjing Wang. Epistemic Logic. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2024 edition, 2024. (cit. on p. 35.)

SBD+10. Jan Sacha, Bartosz Biskupski, Dominik Dahlem, Raymond Cunningham, René Meier, Jim Dowling, and Mads Haahr. Decentralising a service-oriented architecture. *Peer-to-Peer Networking and Applications*, 3:323–350, 2010. (cit. on p. 34.)

SWvRM20. Isaac Sheff, Xinwen Wang, Robbert van Renesse, and Andrew C Myers. Heterogeneous paxos: Technical report. *arXiv preprint arXiv:2011.08253*, 2020. URL: https://arxiv.org/abs/2011.08253. (cit. on p. 34.)

Acknowledgements. We thank Tobias Heindel for comments on a draft, and for asking good questions.