TU WIEN Informatics

An Extension Framework for Epistemic Reasoning in Byzantine Distributed Systems

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Technische Informatik

eingereicht von

Thomas Schlögl, BSc
Matrikelnummer 01125213

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Prof. Ulrich Schmid
Mitwirkung: Roman Kuznets, Ph.D.

Wien, 30. Jänner 2020

Thomas Schlögl	Ulrich Schmid

TU WIEN Informatics

An Extension Framework for Epistemic Reasoning in Byzantine Distributed Systems

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

by

Thomas Schlögl, BSc
Registration Number 01125213

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Ulrich Schmid
Assistance: Roman Kuznets, Ph.D.

Vienna, 30th January, 2020

Thomas Schlögl	Ulrich Schmid

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Thomas Schlögl, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. Jänner 2020

Thomas Schlögl

Danksagung

Ich möchte mich bei meinen Betreuern Prof. Ulrich Schmid und Roman Kuznets herzlichst bedanken für Ihre Unterstützung durch wertvolle Rückmeldungen und Verbesserungsvorschläge zu dieser Arbeit.

Diese Arbeit wurde vom österreichischen Fonds zur Förderung der wissenschaftlichen Forschung (FWF) unter den Projekten ADynNet (P28182) and RiSE/SHiNE (S11405) gefördert.

Acknowledgements

I would like to express my deepest thanks to my supervisors Prof. Ulrich Schmid and Roman Kuznets for providing me with valuable suggestions and feedback on this work.

This thesis has been supported by the Austrian Science Fund FWF under the projects ADynNet (P28182) and RiSE/SHiNE (S11405).

Kurzfassung

Diese Diplomarbeit präsentiert ein Erweiterungs-Framework für die leistungsfähige epistemische Modellierungs- und Analyse-Umgebung für Multiagentensysteme mit byzantinisch fehlerhaften Agenten, die 2019 von Kuznets el. al. publiziert wurde. Es erweitert die ursprünglich nur für asynchrone Agenten und asynchrone Kommunikation formulierte Umgebung in einer generischen Art und Weise, die die Formulierung und Kombination zusätzlicher Systemannahmen in modularer Form erlaubt. Unter den spezifischen Erweiterungen, die in der Arbeit enthalten sind, finden sich unter anderem zuverlässige Kommunikation, zeitbeschränkte Kommunikation, Multicasting, synchrone und lockstep-synchrone Agenten und sogar koordinierte Aktionen von Agenten, wie sie in der Modellierung und Analyse von fehlertoleranten verteilten Systemen oftmals anzutreffen sind. Für die zentralen Erweiterungen der synchronen und lockstep-synchronen Agenten werden auch elementare Eigenschaften des resultierenden Gesamtmodells abgeleitet, wie etwa lokale und globale Fehlererkennungsmöglichkeiten und die Existenz/Nichtexistenz eines “brain-in-the-vat”-Szenarios.

Abstract

In this Master thesis, we provide an extension framework for the powerful epistemic reasoning framework for multi-agent systems with byzantine faulty agents published by Kuznets et. al. in 2019, which is currently restricted to asynchronous agents and asynchronous communication. We enrich the existing framework by a generic way to add extensions, which allow to encode and safely combine different system assumptions in a modular way. Among the particular extensions provided in this thesis are reliable communication, time-bounded communication, multicasting, synchronous and lock-step synchronous agents and even agents with coordinated actions, which are commonly used in modeling and analysis of fault-tolerant distributed systems. For the pivotal cases of synchronous and lock-step synchronous agents, we also analyze the basic properties of the resulting model, such as local and global fault detection abilities of the agents and the possibility of a "brain-in-a-vat" scenario.

Contents

CHAPTER 1

Introduction

Distributed systems (DS) consist of networked processors without a central control that need to achieve a common goal. *Distributed algorithms* allow the processors to exchange information and reason about the evolving state of their local knowledge, in order to infer sufficient knowledge of the global system state to achieve their common goal. However, various sources of uncertainty like varying execution speeds, unpredictable transmission delays and partial failures make it notoriously difficult to infer the global state of a distributed system. Consequently, the literature on distributed algorithms is abundant, and many landmark results both on solvability of distributed computing problems like consensus [LSP82] and on impossibility results [FLP85] were established in the past. Almost all correctness proofs of distributed algorithms are hand-crafted, however, with automatic or at least computer-assisted proofs done from the start (not to speak of synthesis [LKWB18]) being a rare exception [DHJ+16]. Overall, it is fair to say that the design of fault-tolerant distributed algorithms is still more of an art than solid engineering.

The situation is even worse in the case of *fault-tolerant* distributed systems, in particular, those where processors can misbehave in an arbitrary (*byzantine*) [LSP82] way, e.g., by sending inconsistent information to different respondents. The uncertainty added by byzantine faulty processors creates another level of complexity in the design and analysis of distributed algorithms, which severely complicates these tasks even for relatively simple problems like clock synchronization [ST87, WL88] or byzantine agreement [LSP82].

Existing epistemic languages, such as the popular runs and systems framework [HM90], already provide a protocol-independent formal high-level description of knowledge states, even for agents that may lose messages and/or crash [DM90]. This led to very general results, e.g., the correspondence between distributed agreement and common knowledge, which is the ultimate cause of the impossibility to coordinate actions without reliable communication [Gra78]. One of the primary tools in such analyses is the causal cone based on Lamport's happened-before relation [Lam78], used, e.g., to prove the necessity of nested knowledge for ordered responses [bM14] and the ability of a silent choir [GM18] to enable communication-by-time in crash-prone systems.

However, existing epistemic reasoning frameworks cannot handle agents that are not simply allowed to become mute but may also disseminate erroneous information, e.g., by lying about past

events. However, to the best of our knowledge, a comprehensive epistemic reasoning framework that also allows byzantine agents did not exist before [KPS+19b, KPSF19a, KPSF19b, Fru19].

1.1 Epistemic logic

At least since the ground-breaking work by Halpern and Moses [HM90], the knowledge-based approach [FHMV95a] is known as a powerful tool for analyzing distributed systems. In a nutshell, it uses epistemic logic [Hin62] to reason about knowledge and belief in multi-agent systems. Standard epistemic logic relies on a Kripke model M that describes the possible global states ("possible worlds") the agents can be in, where certain atomic propositions (facts like "variable x_i in the state of agent i is zero" or "external event e occurred on agent i") hold true or not, along with an indistinguishability relation $s \sim_i s'$ that tells that agent i cannot distinguish locally whether it is in global state s or s'. Knowledge of some fact φ about the system in global state s is primarily captured by a modal *knowledge operator* K_i, used in formal expressions like $(M, s) \models K_i\varphi$. It captures the intuition that, being in the global state s of model M, agent i knows φ iff φ holds in every global state s' that is indistinguishable from s for i.

In the *interpreted runs and systems* framework for reasoning about distributed and other multi-agent systems [FHMV95a, HM90], the set of all possible runs r (executions) of a system I determines the set of Kripke models, formed by the evolution of the global state $r(t)$ in all runs $r \in I$ over time $t \in \mathbb{N}$. Note that time is modeled as discrete for simplicity, without necessarily being available to the agents. Two global states $r(t)$ and $r'(t')$ are indistinguishable for agent i iff i has the same local state in both, formally, $r_i(t) = r'_i(t')$. Therefore, i knows some fact φ in run $r \in I$, formally, $(I, r, t) \models K_i\varphi$, iff in every $r' \in I$ and for every t' with $r_i(t) = r'_i(t')$ it holds that $(I, r', t') \models \varphi$. Note that φ can be a formula containing arbitrary atomic propositions, as well as other knowledge operators and temporal modalities like $\Diamond$ (eventually) and $\Box$ (always), combined by standard logical operators negation $\neg$, conjunction $\wedge$, disjunction $\vee$, and implication $\Rightarrow$. For example, $(I, r, t) \models \Diamond K_i \mathit{occurred}(e)$ states that there is some time $t' \geq t$ when i will know that event e will have occurred somewhere. Important additional modalities are *mutual knowledge* $E_G\varphi \equiv \bigwedge_{i \in G} K_i\varphi$ in a group G of agents, and *common knowledge* $C_G\varphi$, which can usually be viewed as an infinite conjunction of nested mutual knowledge of arbitrary depth $C_G\varphi \equiv E_G\varphi \wedge E_G(E_G\varphi) \wedge \ldots$; informally, this means that every agent in G knows that every agent in G knows that ... every agent in G knows φ.

The knowledge-based approach based on runs and systems has been used for studying several distributed computing problems in systems with uncertainty but no failures. In [bM14], Ben-Zvi and Moses considered the simple *ordered response* problem in distributed systems, where the agents had to respond to an external START event by executing a special one-shot action FIRE in a given order $i_1, i_2, \ldots$. The authors showed that, in every correct solution, agent i_k has to establish the nested knowledge $(I, r, t) \models K_{i_k} K_{i_{k-1}} \ldots K_{i_1} \mathit{occurred}(\text{START})$ before it can issue FIRE and that this nested knowledge is also sufficient. Moreover, they showed that every correct solution in systems with known bounded message delays necessarily requires a communication structure called a *centipede* in the *agent-time graph*.[1] In the conference

[1] The standard agent-time graph consists of pairs (i, t) of agent i and time t and of directed edges that represent Lamport's happens-before relation [Lam78]: edges $(i, t) \to (i, t+1)$ represent the local information

version [BZM10] of [bM14], the authors also considered the *simultaneous response* problem, where all agents had to issue FIRE at the same time. It requires the group G of firing agents to establish common knowledge $C_G occurred(\text{START})$ [HM90]. This work was later extended to responses that are not simultaneous but tightly coordinated in time [BZM13, GM13]. A refined characterization of the causal cone in systems with communication delay bounds is provided in [bM18].

The knowledge-based approach has also been successfully applied to fault-tolerant synchronous distributed systems. Agents suffering from some restricted types of failures, such as crash or omission failures, have already been studied in [MT88], primarily in the context of agreement problems [DM90, HMW01], which require some form of common knowledge. An important ingredient here are indexical sets of correct agents and a related belief operator $B_i\varphi \equiv K_i(correct_i \Rightarrow \varphi)$ [MS93], which states that agent i knows φ to be true in all runs where i is correct. More recent results are unbeatable consensus algorithms in synchronous systems with crash failures [CGM14], and the discovery of the importance of *silent choirs* [GM18] for message-optimal protocols in crash-resilient systems.

Even though several of the above papers have "byzantine" in their title, it is nevertheless the case that they solely consider benign faults, like crashes and message omissions. This is also true for [GM19], where silent choirs are used for speeding up byzantine consensus in fault-free runs. We are not aware of any attempt to extend epistemic reasoning to systems with truly byzantine faults, except for the PhD thesis [Mic89], where faulty agents may deviate from their protocols also by sending wrong messages etc. However, even faulty agents may not really behave arbitrarily there, as they need to exhibit a behavior that could have been observed in some correct execution as well.

Fault-tolerance has also been considered in the context of *multi-agent systems* (MAS), where temporal-epistemic languages like CTLK [FHMV95a] and even symbolic (but not parametrized, e.g., w.r.t. the number of agents n) model checkers like MCMAS [LQR09] exist. For a given concrete interpreted systems model, it is hence possible to specify and automatically verify arbitrary temporal-epistemic properties. For systems that may suffer from faults, replication-based fault-tolerance techniques [FD02], diagnosis-based approaches [KK07], and even lying agents [dD14b, dD14a] have been considered in the past, albeit in very specific settings. In the more generic approach described in [EL17], automatic fault injection techniques [Iye95] are used for mutating a given interpreted systems model of some application, in order to allow agents to misbehave in some way, and to verify the properties of the resulting model using MCMAS.

The first steps towards an epistemic reasoning framework for MAS with byzantine agents, conducted in my supervisor's *Embedded Computing Systems Group* at TU Wien, started out from a Master's thesis by Patrik Fimml [Fim18], which eventually led to the comprehensive report [KPS+19b] that also forms the basis of this thesis. Part of [KPS+19b] has already been published at FroCoS 2019, TARK 2019, and ESSLLI 2019: In [KPSF19b], it has been proved that byzantine agents allow for a "brain-in-a-vat" scenario, which makes it impossible for a faulty agent to perceive its correctness. In [KPSF19a], the byzantine analog of the causal cone [bM14] has been developed. *Multipedes* were identified as the crucial communication structure for reliably detecting the occurrence of certain events in the system, and a close

flow at agent i while edges $(i,t) \to (j,t')$ correspond to information transfer from i to j by means of a message sent at time t and received at time t'.

relation to a novel modality *hope* that is a promising candidate for replacing belief in byzantine fault-tolerant systems [MS93] has been established. Finally, in [Fru19], a sound and complete axiomatization of hope has been provided.

1.2 General methodological approach

The primary basis for this thesis is the epistemic framework for byzantine asynchronous agents [KPS+19b, KPSF19a], which needs to be extended to cover synchronous agents, coordinated actions, broad- and multicasts (hard- and software), etc. Moreover, the currently very limited set of generic and protocol-specific lemmas and theorems need to be extended appropriately.

This comprehensive undertaking is doable due to the fact that the existing framework [KPS+19b] has been designed to be modular, e.g., by carefully separating the multiple roles played by the environment (message delivery, inert physical medium, and adversary) into disjoint phases. Thus, one phase can be modified independently from the others. The second crucial design choice was giving the environment undivided control over byzantine behavior. First results regarding the causal cone for byzantine asynchronous agents both prove the potential of the framework and provide persuasive parallels with existing treatments of knowledge of infallible agents: for example, the necessity to have a causal chain is strengthened to a causal chain consisting of correct agents [KPSF19a]. At the same time, our preliminary findings show that in byzantine settings it is necessary to relativize the notion of knowledge modulo the agent's correctness.

The remainder of this thesis is structured as follows:

- Chapter 2 copies the required parts of [KPS+19a] for convenience of the reader.
- Chapter 3 makes up the core of this thesis.
- Chapter 4 presents some conclusions and directions of further research.

CHAPTER 2

The Byzantine Message-Passing Framework [KPS+19a]

The content of this whole chapter in its entirety was copied from [KPS+19a] for easier reference. The original content of this master thesis starts at Chapter 3.

2.1 Agents and States

We consider distributed multi-agent systems with agents viewed as abstract processes, which can represent humans, computers, or more basic devices.

Definition 2.1.1 (Agents and nodes)**.** We consider a non-empty finite set $\mathcal{A}$ of **agents**. Without loss of generality, we assume that the agents are numbered: $\mathcal{A} = [\![1; n]\!]$ for some integer $n > 1$.[1] (The case of a distributed system of only one agent is considered degenerate as it does not exhibit typical properties of a distributed system.) **Local timestamps**, or simply **nodes**, are identified by pairs $(i, t) \in \mathcal{A} \times \mathbb{N}$ of an agent i and a **timestamp** t. For a set $X \subseteq \mathcal{A} \times \mathbb{N}$ of local timestamps, we define the set

$$\mathcal{A}(X) := \{i \mid (\exists t \in \mathbb{N})(i, t) \in X\}$$

of involved agents

In our distributed model, non-negative integer timestamps $0, 1, 2, \ldots$ are used exclusively for snapshots of the system's state. All actions are performed during the open intervals in between, called **rounds**: $]0; 1[,]1; 2[,]2; 3[, \ldots$[2] We use the abbreviation $t.5$ to denote the round $]t; t + 1[$. For instance, it is equivalent to discuss one agent initiating the sending of a message at timestamp t and another agent receiving this message by timestamp $t + 1$, on the one hand, or to discuss the message sent and received during the round $t.5$, on the other hand.

The system begins with each agent in one of its *initial states*.

[1] We use the notation $[\![k; m]\!]$ to denote the set of integers from k to m, i.e., $\{i \in \mathbb{N} \mid k \leq i \leq m\}$.

[2] We use the notation $]k; m[$ to denote the set of real numbers strictly between k and m, i.e., $\{x \in \mathbb{R} \mid k < x < m\}$.

Definition 2.1.2 (Initial states)**.** Each agent $i \in \mathcal{A}$ has a set Σ_i of **local initial states**. A **joint initial state**, or **global initial state**, is a tuple of local initial states from

$$\mathscr{G}(0) := \prod_{i \in \mathcal{A}} \Sigma_i.$$

An agent's state can be modified due to *internal actions* of the agent itself and/or *external events* triggered by the **environment**, represented as a designated agent ϵ, which is *not* considered a member of $\mathcal{A}$. Since we do not assume agents to be of the same type, their sets of initial states, available internal actions, and observable external events may differ. An example of a local internal action is incrementing a local counter. An example of a local external event is receiving input from a motion-detection sensor.

Definition 2.1.3 (Local internal actions and events)**.** Int_i denotes the set of all **local internal actions** of agent $i \in \mathcal{A}$. Ext_i denotes the set of all **local external events** it can observe. We use

- $a, a', a'', \ldots, a_1, a_2, \ldots$, for local internal actions,
- $e, e', e'', \ldots, e_1, e_2, \ldots$, etc. for local external events, and
- $o, o', o'', \ldots, o_1, o_2, \ldots, u, u', u'', \ldots, u_1, u_2, \ldots$ as a generic notation for both events and actions.

We consider a message-passing system whereby agents communicate exclusively by messages. Unlike other actions, which may be specific to an agent, messages should be understandable for both the sender and receiver. Hence, we assume one uniform set of messages comprehensible for all agents.

Definition 2.1.4 (Messages)**.** We denote by *Msgs* the (possibly infinite) set of **messages** that agents can send to each other. For any two agents $i, j \in \mathcal{A}$, a message $\mu \in Msgs$ can be

- sent by agent i to agent j (possibly in multiple copies), which constitutes an internal action of i and is recorded[3] in agent i's history as $send(j, \mu_k)$ for the kth copy of the message; we consider $send(j, \mu_0)$ to be the *master copy* and denote it simply by $send(j, \mu)$ in protocols where multiple copies are not necessary;
- received by agent j from agent i, which constitutes an external event for j and is recorded in agent j's history as $recv(i, \mu)$ (note that j is not aware whether multiple copies of this message have been sent by i to j and cannot tell which copy it has received).

Remark 2.1.5 (Channel implementation)**.** We chose not to model communication channels explicitly. The way messages are delivered from one agent to another is governed by the environment ϵ but this process remains a black box with certain postulated guaranteed properties, e.g., we will present tools capable of ensuring that successfully delivered messages arrive in the same round they have been sent. The most important property of our message-passing system is that it is incorruptible. Agents do not have access to it. In particular,

[3]The exception is the case when sending of the message was a Byzantine action. Then the record of sending can be missing or corrupted, in particular, it can look like another action.

Byzantine agents cannot in any way affect message delivery. The most they can do is to send false information, but cannot pretend that this false information comes from a reliable source.

This might be viewed as a restriction on their Byzantine power. Fortunately, the model is flexible enough to also implement agents impersonating other agents. The underlying assumption of our model is that two communicating agents would necessarily recognize each other (this is sometimes referred to as "oral messages"). Thus, in order to mask one's identity, one simply needs to transfer messages through a relay station. Thus, the unfalsifiable identity of the last sender of the message will belong to such a relay station, whereas the identity of the originator of the message would have to be part of the message content and, hence, malleable. In cases where there is no obvious physical manifestation for such a relay station, e.g., for wireless communication, we can still use a relay station to represent the medium that makes the communication possible. Without going into the technical details, we also remark that relay stations can be implemented without the increase in latency in message delivery.

Remark 2.1.6 (Global ids and global view)**.** Allowing multiple copies of the same message serves only one purpose: to enable sending several duplicates of the same message in the same round. Generally, these copies need not be used and, conversely, their use does not have any effect on the sender's behavior. In particular, there is no obligation to use consecutive numbers for copies nor to always use a fresh copy number in following rounds.[4] Thus, despite having a possibility to distinguish all sent messages, agents are under no obligation to do so.

Thus, the same copy k of the same message μ can be sent from i and received by j multiple times with the sent messages $send(j, \mu_k)$ and received messages $recv(i, \mu)$ all looking the same for agents i and j respectively. The environment ϵ, which also plays the role of the delivery system, must, however, be able to distinguish among identical copies of messages with identical contents but sent/received at different times. This is modelled by a **global message identifier** $id \in \mathbb{N}$, or simply **GMI**, which can be compared to a tracking number used by the environment to uniquely identify each message. Agents never observe this GMI.

Further, while agent i only observes messages sent by itself and its own actions and events, the environment should distinguish between a message μ sent to j from i and a message with the same content μ sent to j by another agent i', between action a by i and the same action a by j, between event e observed by i and the same event e observed by j. Thus, the environment represents

- copy k of a message μ sent from i to j and assigned GMI id in the format $gsend(i, j, \mu, id)$ with the information of the copy number k transferred to the GMI id,
- the same copy of the same message when received by j in the format $grecv(j, i, \mu, id)$,
- action a by agent i in the format $A = internal\,(i, a)$,
- event e observed by i in the format $E = external\,(i, e)$.

We will refer to this as the *global* or *environment's view*, as opposed to the *local view* $send(j, \mu_k)$ and $recv(i, \mu)$ of the sending and receiving agents respectively, a of the acting agent, and e of the observing agent. We denote

[4] However, within one round each new copy requires a fresh number. Otherwise, it will be conflated with the same-numbered copy because messages form a set.

	Correct	Byzantine
Observing the round	$go(i)$	$sleep\,(i)$
Not being aware of the round		$hibernate\,(i)$

Table 2.1: Possibilities regarding agent's actions in a round.

- globally presented actions by A, A', A_1, etc.,
- globally presented events by E, E', E_1, etc., and
- globally presented either by $O, O', O_1, \ldots, U, U', U_1, \ldots$.

We assume that a'' and A'' or e_{13} and E_{13}, etc. represent the same action/event presented locally and globally respectively.

After we define protocols and the normative, by-the-protocol behavior for agents, we will introduce agents with Byzantine behavior, i.e., behavior defying their protocols.

Remark 2.1.7 (Modelling asynchronous agents)**.** Asynchronous agents do not have access to the global clock of the system. In particular, they should not be able to count the rounds passed from the beginning. This is implemented by letting agents skip one or more rounds completely. For each round, the environment ϵ controls whether an agent is to be awoken to implement its protocol and/or observe some external events or is to skip the round.

This choice for agent i is implemented by three system events: $go(i)$, $sleep\,(i)$, and $hibernate\,(i)$. None of these events are registered by the agents. The $go(i)$ event, unless countermanded by $sleep\,(i)$ or $hibernate\,(i)$ causes the agent to implement its protocol for this round. Even if the protocol prescribes to stand by and do nothing, the agent would still record the passage of time. Independently, the agent records the passage of time whenever it observes any event[5] other than a Byzantine event perceived as the special **no-op** action 🗓.

Events $sleep\,(i)$ and $hibernate\,(i)$ model situations when the agent fails to act due to a malfunction. They differ in whether, despite not acting, the agent is supposed to notice the passing round or not. Note that we are distinguishing between the agent *actively doing nothing* (observing the round passing without any action) and *passively not doing anything* (not being aware of the passing round at all). For either option, we present a possible malfunction resulting in it (in the absence of any actions or events).

Remark 2.1.8 (Sufficient conditions for message transfer)**.** As can be seen from the preceding remark, it may happen that despite an agent's protocol prescribing it to send a message, this action is thwarted by the environment not waking up the agent. Additionally, a properly functioning agent should not be able to receive a message that was not yet sent (i.e., sent no later than in the same round). This causally motivated restriction on the model will be implemented by a special filter (see Def. 2.2.19) that uses the GMI of the message to distinguish among multiple duplicate messages if need be. This filter can be viewed as part of the environment.

[5] It might seem that waking up agents to receive messages interferes with asynchronicity. However, just like $go(i)$, the delivery of messages is controlled by the environment. In particular, the environment can make the agent skip rounds by postponing all message deliveries.

Because of the active role agents play in actions, there are more ways to violate rules while performing them. We discuss Byzantine events and actions separately.

Remark 2.1.9 (Modelling Byzantine events)**.** In the spirit of the distributed paradigm, i.e., in the absence of a global observer, we do not model global events. Even if one event is observed by several agents, we model these observations independently and do not generally postulate the event to be registered by all agents in the same round. In other words, the fact that one agent failed to observe event e or observed it too late, despite another agent having observed it (earlier) does not generally make the first agent's behavior incorrect.[6]

Instead the faults are agent-centric. It is not an event itself that causes the fault, it is the agent's perception of the event (with the exception of system events). Consequently, the only difference we model is between the agent recording an event e that happened to it vs. the agent recording e even though e did not take place. Note that the two versions are incompatible. Consequently, they never happen simultaneously. More precisely, the environment never attempts simultaneously a correct and faulty event that leave the same trace in the agent's local history.

Remark 2.1.10 (Modelling Byzantine actions)**.** Unlike events, which are not controlled by the agent, actions depend on its will. Accordingly, apart from recording its own actions incorrectly, the agent can also perform wrong actions, i.e., a set of actions not envisioned by its protocol. It is also clear that whether the action is correct and whether it is correctly recorded are independent of each other. Accordingly, there are four possibilities:

- a correct action is correctly recorded;
- a correct action is mistaken for another action or possibly not recorded at all;
- an incorrect action is correctly recorded;
- an incorrect action is mistaken for another (possibly correct) action or possibly not recorded.

Clearly, all but the first case represent faulty agents. It is especially important to distinguish which action took place and which action (if any) the agent thinks took place for the action of sending a message. We assume that all messages actually sent, whether correctly or otherwise and independent of whether the agent retains a record of sending it, are treated by the environment in the same way. In particular, the environment always assigns a correct GMI for all messages actually sent meaning that Byzantine agents are not able to deceive the environment as to the identity of their messages or, indeed, as to the Byzantine nature of these messages. However, agent i may, for instance, think that a message was sent to agent j, whereas in fact it was sent to k. The GMI for such a message and all the delivery procedures will correspond to the true state of affairs. In particular, the environment would not be allowed to deliver this message to j in a correct manner. Similarly, the agent sending a message without being aware of it is modeled by the send action being mistaken for the special Byzantine **no-op** action 🗓. More generally, any action may remain unregistered by the agent when

[6] If it is absolutely necessary for correct agents to observe a particular event, it is still possible to model agents guilty of not observing it by creating a special Byzantine event *distracted*.

mistaken for 🗓. Conversely, the agent may believe to have acted despite not doing anything if 🗓 is mistaken for this action. The latter situation still falls under "actively doing nothing," at least from the agent's point of view. Thus, 🗓 is used as a Byzantine action that has no consequences and leaves no record. In particular, if 🗓 is the only action/event of agent i during a round (independently of which actions it erroneously performed in reality), then the local history of the agent remains unchanged, unless $sleep(i)$ forces the agent to mark time.

Remark 2.1.11 (The culprit of Byzantaneity)**.** Despite the Byzantine agents acting any way they like, they should be able to reason about their actions the same way as uncorrupted agents. This is especially crucial for agents who are corrupted due to malfunction rather than malfeasance. This need is at the heart of our decision to shift the control of Byzantine behavior from corrupted agents to the environment. Thus, formally, a Byzantine "action" of an agent, including sending a Byzantine message, is not modelled as the agent's action, but is instead an external event imposed on the agent by the environment, an event specifying both which action took place and which action the agent thinks took place. The agent is unable to distinguish between itself performing some action a and the environment imposing some action that looks like a to the agent. In particular, as will be shown later, the only way for the agent to learn that it is corrupted is by observing the discrepancies between its actions and those demanded by its protocol, more precisely, those discrepancies the agent is able to perceive.

Remark 2.1.12 (How environment communicates)**.** Our model does not provide for message passing between an agent and the environment because such messages would be redundant. On the one hand, the environment is considered to be omniscient and already knows everything the agents might want to communicate to it. As for the information flow in the opposite direction, the environment is not viewed as a conscious entity trying to communicate (though, in principle, information can be delivered to an agent from the environment by means of external events).

We now flesh out the formal definitions for the concepts just discussed:

Definition 2.1.13 (Global message identifier function)**.** We fix a function for computing GMIs to be any computable one-to-one total function $id\colon \mathcal{A} \times \mathcal{A} \times Msgs \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.[7]

Note that the two functions retrieving the arguments t and k from $id(i, j, \mu, k, t)$ respectively would always be computable due to the injectivity and totality of $id(\cdot)$. However, it is beneficial to choose $id(\cdot)$ so as to additionally make these two functions computable efficiently.

Definition 2.1.14 (Internal actions)**.** From the point of view of agent $i \in \mathcal{A}$, its correct **internal actions**, which can be prescribed by its protocol, consist of the *send* actions from Def. 2.1.4 and local internal actions $a \in Int_i$ (Def. 2.1.3):

$$\overline{Actions}_i := \{send(j, \mu_k) \mid j \in \mathcal{A}, \mu \in Msgs, k \in \mathbb{N}\} \sqcup Int_i. \tag{2.1}$$

The same actions from the point of view of the environment look like

$$\overline{GActions}_i := \{gsend(i, j, \mu, id) \mid j \in \mathcal{A}, \mu \in Msgs, id \in \mathbb{N}\} \sqcup \{internal\,(i, a) \mid a \in Int_i\} \tag{2.2}$$

[7] A simple though not necessarily the most efficient possibility is to use $2^i \cdot 3^j \cdot 5^{\lceil \mu \rceil} \cdot 7^k \cdot 11^t$, where $\lceil \mu \rceil$ represents the numerical code of the message μ according to some arbitrary but fixed coding scheme.

We also define the sets of all (correct) internal actions that at least one of the agents can take:

$$\overline{Actions} := \bigcup_{i \in \mathcal{A}} \overline{Actions_i} \tag{2.3}$$

$$\overline{GActions} := \bigsqcup_{i \in \mathcal{A}} \overline{GActions_i} \tag{2.4}$$

Remark 2.1.15. Note that the union in (2.4) is disjoint because each action viewed globally necessarily specifies the acting agent. Even if the same local internal action $a \in Int_i \cap Int_j$ can be performed by several agents, the representations $internal\,(i, a) \neq internal\,(j, a)$ of the action for these agents $i \neq j$ are distinct.

Definition 2.1.16 (External events)**.** From the point of view of agent $i \in \mathcal{A}$, **correct external events** that it can observe consist of the *recv* actions from Def. 2.1.4 and local external events $e \in Ext_i$ (Def. 2.1.3):

$$\overline{Events_i} := \{recv(j, \mu) \mid j \in \mathcal{A}, \mu \in Msgs\} \sqcup Ext_i \tag{2.5}$$

The same events from the point of view of the environment look like

$$\overline{GEvents_i} := \{grecv(i, j, \mu, id) \mid j \in \mathcal{A}, \mu \in Msgs, id \in \mathbb{N}\} \sqcup \{external\,(i, e) \mid e \in Ext_i\} \tag{2.6}$$

For each correct external event $E \in \overline{GEvents_i}$ of agent i, there is a matching **Byzantine external event**

$$fake\,(i, E)$$

representing the agent being mistaken about observing the (local version of the) event E. For $A, A' \in \{🗓\} \sqcup \overline{GActions_i}$, each of which is either a correct global action of agent i or **no-op** 🗓, there is a matching **Byzantine external event**

$$fake\,(i, A \mapsto A')$$

representing the situation when the agent performs (the local version of) action A but thinks that it performed (the local version of) action A'. When the agent faithfully records the performed Byzantine action, we abbreviate

$$fake\,(i, A \mapsto A) = fake\,(i, A).$$

Note that $fake\,(i, 🗓)$ acts as a malfunction without any action or any trace in the local history. Hence, we abbreviate it as

$$fake\,(i, 🗓) = fail\,(i).$$

Correct actions are always recorded faithfully. (It can, however, happen that one or several faulty actions are merged with a correctly performed action in the local history.) We do not impose any *a priory* restrictions on E, A, or A', i.e., a Byzantine agent can mistakenly observe any event, mistakenly perform any action, and mistake any performed action or inaction for any other action or for inaction.

In particular, for received Byzantine messages, the environment creates the message "out of thin air" as if it has been delivered. Such a message will be supplied with a GMI for uniformity's sake but such GMIs is not assumed to carry any information. Indeed, this GMI

will play no role whatsoever as it will be stripped from the message upon delivery. It can even violate the uniqueness of GMIs. Similarly, a Byzantine sent message A is created already with a well-formed GMI, unlike the correctly sent messages, which are supplied with a GMI in a separate step after creation. However, the delivery of messages A with GMI does not depend on whether they originate as Byzantine or correct ones. At the same time, if the agent is mistaken about having sent a message A', its GMI is again immaterial as the agent will not see it nor the environment will ever deliver this "message."

We use $\mathit{fake}(i, U)$ to denote an arbitrary fake event $\mathit{fake}(i, E)$ or action $\mathit{fake}(i, A \mapsto A')$ and denote the set of all such Byzantine events of agent i by

$$\mathit{BEvents}_i := \left\{\mathit{fake}(i, E) \mid E \in \overline{\mathit{GEvents}_i}\right\} \sqcup \left\{\mathit{fake}(i, A \mapsto A') \mid A, A' \in \{\boxed{}\} \sqcup \overline{\mathit{GActions}_i}\right\} \tag{2.7}$$

In addition to correct and Byzantine events, the environment uses system events to determine which agents can act in each round:

- $\mathit{go}(i)$ approves i's actions prescribed by its protocol;
- $\mathit{sleep}(i)$ instructs i to forfeit acting in this round but wakes it up anyways, marking it Byzantine;
- $\mathit{hibernate}(i)$ instructs i to forfeit acting in this round without waking it, marking it Byzantine.

System events $\mathit{SysEvents}_i := \{\mathit{go}(i), \mathit{sleep}(i), \mathit{hibernate}(i)\}$ are not (directly[8]) observable by agents and, hence, are not part of $\overline{\mathit{Events}_i}$.[9]

The complete set of events affecting agent i that the environment can trigger is

$$\mathit{GEvents}_i := \overline{\mathit{GEvents}_i} \sqcup \mathit{BEvents}_i \sqcup \mathit{SysEvents}_i \tag{2.8}$$

We also define the sets of external events affecting all agents as

$$\overline{\mathit{Events}} := \bigcup_{i \in \mathcal{A}} \overline{\mathit{Events}_i}, \qquad \overline{\mathit{GEvents}} := \bigsqcup_{i \in \mathcal{A}} \overline{\mathit{GEvents}_i},$$
$$\mathit{GEvents} := \bigsqcup_{i \in \mathcal{A}} \mathit{GEvents}_i, \qquad \mathit{BEvents} := \bigsqcup_{i \in \mathcal{A}} \mathit{BEvents}_i,$$
$$\mathit{SysEvents} := \bigsqcup_{i \in \mathcal{A}} \mathit{SysEvents}_i.$$

Remark 2.1.17. Since action A' in $\mathit{fake}(i, A \mapsto A')$ is a complete fiction only existing in i's imagination, we could have written it in the format $\mathit{fake}(i, A \mapsto a')$ for the action a' already presented from i's local point of view. The only reason we do not do that is to preserve uniformity and avoid mixing local and global points of view.

Remark 2.1.18 (Modelling Byzantine inaction)**.** As can be seen from Table 2.1, these three events governing actions have the following meanings:

[8] Events $\mathit{go}(i)$ can, under certain circumstances, be detected by the acting agent based on the fact that it is acting. However, mere acting does not generally imply $\mathit{go}(i)$ by itself since the actions can also be Byzantine.

[9] Another group of events not observed by the agents are $\mathit{fake}\left(i, A \mapsto \boxed{}\right)$.

- the event $\mathit{hibernate}(i)$ prevents the agent from acting or from waking up other than to observe other events and makes the agent Byzantine;
- the event $\mathit{sleep}(i)$ wakes the agent up but prevents it from acting and makes it Byzantine;
- the event $\mathit{go}(i)$ wakes up the agent and prompts it to fulfill its protocol;
- finally, without any of the three events, the agent would not act, would not wake up unless to observe other events and would not become Byzantine unless other events cause it.

Since $\mathit{hibernate}(i)$, $\mathit{sleep}(i)$, and $\mathit{go}(i)$ are generally pairwise incompatible, the environment never issues more than one event from $\mathit{SysEvents}_i$, much like she never attempts both a correct event $E \in \overline{\mathit{GEvents}_i}$ and its fake version $\mathit{fake}(i, E)$.

Note that $\{\mathit{hibernate}(i)\}$ is similar in its effect to $\{\mathit{fail}(i)\}$ in the absence of $\mathit{go}(i)$: namely, the Byzantine inaction when the agent does not act, does not mark time (unless because of other events), and becomes Byzantine. However, the similarity is not perfect: $\mathit{fail}(i)$ is a generic failure due to the lack of some action, whereas $\mathit{hibernate}(i)$ signifies the specific failure to act in the round altogether. Accordingly, $\mathit{fail}(i)$ is compatible with $\mathit{go}(i)$, whereas $\mathit{hibernate}(i)$ is not. Thus, having this separation makes sense for diagnostic purposes. We generally intend to have a full taxonomy of possible failures: failures based on events, based on actions, based on the absence of actions, and based on failure to follow system instruction $\mathit{go}(i)$.

Remark 2.1.19. It is possible to define the correct version $\mathit{internal}(i, 🗓)$ of $\mathit{fake}(i, 🗓)$ but, given that 🗓 is not recorded in the local history, this would be a wholly redundant operation.

Remark 2.1.20. There can be two main causes for Byzantine behavior:

- the agent may want to subvert the correct procedures and actively engages in sabotage;
- the agent is malfunctioning, which can result in incorrect sensor data being recorded, actions performed in reaction to this erroneous data, and/or actions in response to correct data but not correctly implemented.

Our formal model is attuned to the latter case, where the malfunctions are imposed by the environment. While the former case remains faithfully represented as the environment is free to implement any malicious intent by the agent, the epistemic approach is primarily relevant in the setting with malfunctioning agents. Indeed, if the agent has a complete freedom to misbehave to achieve a goal different from the stated protocol, then its protocol need not contain any self-diagnostic tools whereas for other agents its Byzantine actions are indistinguishable from random actions. By contrast, a malfunctioning agent that is capable of detecting its own faultiness can make necessary adjustments or shut down.

Remark 2.1.21 (Perfect recall)**.** We consider agents capable of perfect recall. More precisely, they do have perfect recall while acting correctly and their memory remains stable once recorded. However, Byzantine agents may misremember some of their events/actions. In particular, Byzantine agents may not remember any actions they performed. If perfect recall is desired for all agents, then Byzantine actions of the type $\mathit{fake}(i, A \mapsto A')$ with $A \neq A'$ should be prohibited.

Perfect recall imposes certain requirements on the memory available to the agent. The choice between perfect recall and history-free agents is akin to the difference between Turing machines and finite-state automata, i.e., the choice between expressivity and efficiency. In this report, we concentrate on the formalism that is expressive. Hence, the state of an agent is defined as its local *history*. In the string of following definitions as well as in the following statements and proofs, we assume that a set $\mathcal{A} = [\![1; n]\!]$ of agents, sets Σ_i of initial states, sets Int_i of local internal actions and sets Ext_i of local external events for each agent $i \in \mathcal{A}$, as well as a set *Msgs* of messages are arbitrary but fixed and we do not repeat this list every time.

Definition 2.1.22 (Agent's history)**.** A **history** h_i **of agent** $i \in \mathcal{A}$, or its **local state**, is a non-empty sequence

$$h_i = [\lambda_m, \ldots, \lambda_1, \lambda_0]$$

for some $m \geq 0$ such that $\lambda_0 \in \Sigma_i$ and $\forall j \in [\![1; m]\!]$ we have $\lambda_j \subset \overline{Actions_i} \sqcup \overline{Events_i}$. In this case m is called the **length of history** h_i and denoted $|h_i|$. We say that a set $\lambda \subset \overline{Actions_i} \sqcup \overline{Events_i}$ **is recorded** in the history h_i of agent i and write $\lambda \subset h_i$ iff $\lambda = \lambda_j$ for some $j \in [\![1; m]\!]$. We say that $o \in \overline{Actions_i} \sqcup \overline{Events_i}$ **is recorded** in the history h_i and write $o \in h_i$ iff $o \in \lambda$ for some set $\lambda \subset h_i$.

Definition 2.1.23 (Environment's history)**.** A **history** h **of the system** with n agents, or the **global state**, is a tuple

$$h := (h_\epsilon, h_1, \ldots, h_n)$$

where the **history of the environment** is a sequence

$$h_\epsilon = [\Lambda_m, \ldots, \Lambda_1]$$

for some $m \geq 0$ such that $\forall j \in [\![1; m]\!]$ we have $\Lambda_j \subseteq \overline{GActions} \sqcup GEvents$ and h_i is a local state of each agent $i \in [\![1; n]\!]$. In this case m is called the **length of history** h and denoted $|h| := |h_\epsilon|$, i.e., the environment has the true global clock. We say that a set $\Lambda \subset \overline{GActions} \sqcup GEvents$ **happens** in the environment's history h_ϵ or in the system history h and write $\Lambda \subset h_\epsilon$ iff $\Lambda = \Lambda_j$ for some $j \in [\![1; m]\!]$. We say that $O \in \overline{GActions} \sqcup GEvents$ **happens** in the environment's history h_ϵ or in the system history h and write $O \in h_\epsilon$ iff $O \in \Lambda$ for some set $\Lambda \subset h_\epsilon$.

Definition 2.1.24 (Sets of local and global states)**.** $\mathscr{L}_i$ is the set of **local states** of agent i, i.e., the set of all histories of agent i. $\mathscr{L}_\epsilon$ is the set of histories of the environment. $\mathscr{L} := \prod_{i \in \mathcal{A}} \mathscr{L}_i$ is the set of **joint local states**. $\mathscr{G}$ is the set of **global states**.

The global state of the system contains both the local states of all agents and the omniscient view of the environment. It provides a complete snapshot of the system at a specific time, including the real picture of events and how these events are perceived by agents.

The following is an exhaustive list of the notation we use to describe main stages in a lifecycle of events and actions. For completeness purposes, this list also mentions protocols that will be formally introduced later. In this list, $i, j \in \mathcal{A}$ are agents, $\mu \in Msgs$ is a message, and $id \in \mathbb{N}$ is a GMI.

Correct internal actions and their Byzantine copies:

- $a \in Int_i$ represents the following:
  - in i's protocol, this prescribes agent i to perform the local internal action a if it is woken up for the round; this command by itself does not affect whether the agent marks the passage of time: that depends on whether the agent is woken up;
  - in i's local history, this means that agent i marked the passage of time and thinks it performed a, but does not necessarily know whether it was really performed and, if so, whether it was performed according to the protocol or in a Byzantine fashion.
- $\mathit{internal}\,(i, a)$ for $a \in Int_i$ represents the following:
  - in the environment's history, this means that i performed a according to i's protocol and marked the passage of time.
- $\mathit{fake}\,(i, \mathit{internal}\,(i, a) \mapsto A')$ for $a \in Int_i$ and $A' \in \overline{GActions_i}$ represents the following:
  - in the environment's protocol, this prescribes agent i to perform a in a Byzantine fashion (i.e., irrespective of both the protocol and whether the agent is woken up) but believe that the local version of A' was performed instead; if approved, this event causes the agent to mark the passage of time;
  - in the environment's history, this means that i performed a in a Byzantine fashion but believed that the local version of A' was performed instead and marked the passage of time.
- $\mathit{fake}\,(i, \mathit{internal}\,(i, a) \mapsto \text{📅})$ for $a \in Int_i$ represents the following:
  - in the environment's protocol, this prescribes agent i to perform a in a Byzantine fashion but forget about it; this command does not affect whether the agent marks the passage of time;
  - in the environment's history, this means that i performed a in a Byzantine fashion without recording it in its local history; whether i marked the passage of time depends exclusively on other actions/events of the round.
- $\mathit{fake}\,(i, A' \mapsto \mathit{internal}\,(i, a))$ for $a \in Int_i$ and $A' \in \overline{GActions_i}$ represents the following:
  - in the environment's protocol, this prescribes agent i to perform the local version of A' but believe that a was performed instead while marking the passage of time;
  - in the environment's history, this means that i performed the local version of A' in a Byzantine fashion but believed that a was performed instead and marked the passage of time.
- $\mathit{fake}\,(i, \text{📅} \mapsto \mathit{internal}\,(i, a))$ for $a \in Int_i$ represents the following:
  - in the environment's protocol, this prescribes agent i to mistake inaction for performing a while marking the passage of time;
  - in the environment's history, this means that i did not do anything but believes to have performed a and marked the passage of time.

Sending messages, correctly or in a Byzantine way:

- $send(j, \mu_k)$ represents the following:
  - in i's protocol, this prescribes agent i to send kth copy of a message μ to j if it is woken up for the round; in most cases, only one copy, the master copy is sent, which is denoted μ_0 or simply μ; this command by itself does not affect whether the agent marks the passage of time: that depends on whether the agent is woken up;
  - in i's local history, this means that agent i marked the passage of time and thinks it sent kth copy of a message μ to j, but does not necessarily know whether it was really sent and, if so, whether it was sent according to the protocol or in a Byzantine fashion.
- $gsend(i, j, \mu, id)$ represents the following:
  - in the environment's history, this means that i marked the passage of time, sent a message μ to j according to i's protocol, and the message was assigned the GMI id (which contains information about the copy number).
- $fake\,(i, gsend(i, j, \mu, id) \mapsto A')$ for $A' \in \overline{GActions_i}$ represents the following:
  - in the environment's protocol, this prescribes agent i to send a message μ to j in a Byzantine fashion and the environment to assign the GMI id (which contains information about the copy number) to the message and is computed correctly with respect to the current timestamp;, but i would believe that the local version of A' was performed instead; if approved, this event causes the agent to mark the passage of time;
  - in the environment's history, this means that i marked the passage of time, sent a message μ to j in a Byzantine fashion and the message was assigned the GMI id, but i believes that the local version of A' was performed instead; notwithstanding this belief, μ can be correctly received by j.
- $fake\,(i, gsend(i, j, \mu, id) \mapsto \text{🗓})$ represents the following:
  - in the environment's protocol, this prescribes agent i to send a message μ to j in a Byzantine fashion and the environment to assign the GMI id to the message and forget about it; the GMI is computed correctly with respect to the current timestamp; this command does not affect whether the agent marks the passage of time;
  - in the environment's history, this means that i sent a message μ to j in a Byzantine fashion and the message was assigned the GMI id, but forgot about it; whether i marked the passage of time depends exclusively on other actions/events of the round; notwithstanding, μ can be correctly received by j.
- $fake\,(i, A \mapsto gsend(i, j, \mu, id))$ for $A \in \overline{GActions_i}$ represents the following:
  - in the environment's protocol, this prescribes agent i to perform the local version of A but mistakenly believe that it is sending a message μ to j (GMI id does not play a role); if approved, this event causes the agent to mark the passage of time;

  - in the environment's history, this means that i marked the passage of time and performed the local version of A but mistakenly believes to have sent copy μ to j; despite i's belief, this event does not enable j to receive μ correctly.

- $fake\,(i, \text{📅} \mapsto gsend(i, j, \mu, id))$ represents the following:

  - in the environment's protocol, this prescribes agent i to mistakenly believe that it is sending a message μ to j; if approved, this event causes the agent to mark the passage of time;
  - in the environment's history, this means that i marked time and mistakenly believes to have sent a message μ to j; despite i's belief, this event does not enable j to receive μ correctly.

Byzantine inaction

- $fake\,(i, \text{📅})$ represents the following:

  - in the environment's protocol, this prescribes agent i to not do anything in a Byzantine fashion; this command does not affect whether the agent marks the passage of time;
  - in the environment's history, this means that i became Byzantine if it wasn't already; whether i marked the passage of time depends exclusively on other actions/events of the round.

Correct external events and their Byzantine copies:

- $e \in Ext_i$ represents the following:

  - in i's local history, it means that i marked the passage of time and believes to have observed e happened, but does not necessarily know whether it really happened;

- $external\,(i, e)$ for $e \in Ext_i$ represents the following:

  - in the environment's protocol, it prescribes the environment to impose e on agent i; it is incompatible with $fake\,(i, external\,(i, e))$; if approved, this event causes the agent to mark the passage of time;
  - in the environment's history, it means that i marked the passage of time and observed e.

- $fake\,(i, external\,(i, e))$ for $e \in Ext_i$ represents the following:

  - in the environment's protocol, it prescribes agent i to mistakenly believe to have observed e; it is incompatible with $external\,(i, e)$; if approved, this event causes the agent to mark the passage of time;
  - in the environment's history, it means that i marked the passage of time and mistakenly believes to have observed e.

Receiving messages, correctly or in a Byzantine fashion:

- $recv(j, \mu)$ represents the following:
  - in i's local history, it means that i marked the passage of time and believes to have received a message μ from agent j, but does not necessarily know whether the receipt of the message really happened.
- $grecv(i, j, \mu, id)$ represents the following:
  - in the environment's protocol, it prescribes to deliver to i a message μ sent earlier with GMI id by j; it is incompatible with $fake\,(i, grecv(i, j, \mu, id'))$ even if the fake id' is different; if approved, this event causes the agent to mark the passage of time;
  - in the environment's history, it means that i marked the passage of time and received a message μ sent earlier by j with GMI id.
- $fake\,(i, grecv(i, j, \mu, id))$ represents the following:
  - in the environment's protocol, it prescribes agent i to falsely believe to have received a message μ from j (GMI id does not play a role); it is incompatible with $grecv(i, j, \mu, id')$ even if the correct id' is different; if approved, this event causes the agent to mark the passage of time;
  - in the environment's history, it means that i marked the passage of time and falsely believes to have received μ from j.

Environment controlling agents' actions:

- $go(i)$ represents the following:
  - in the environment's protocol, it prescribes agent i to wake up and perform some set of actions prescribed by i's protocol; it is incompatible with $sleep\,(i)$ and $hibernate\,(i)$; if approved, this event causes the agent to mark the passage of time (even if no actions are prescribed by the protocol);
  - in the environment's history, it means that i was woken up, marked the passage of time, and performed some set of actions prescribed by i's protocol.
- $sleep\,(i)$ represents the following:
  - in the environment's protocol, it prescribes agent i to malfunction by marking the passage of time but skipping whatever actions prescribed by i's protocol; it is incompatible with $go(i)$ and $hibernate\,(i)$; if approved, this event causes the agent to mark the passage of time;
  - in the environment's history, it means that i malfunctioned, was woken up and marked the passage of time but was not allowed to perform any actions prescribed by i's protocol.
- $hibernate\,(i)$ represents the following:

– in the environment's protocol, it prescribes agent i to malfunction by skipping whatever actions prescribed by i's protocol; it is incompatible with $go(i)$ and $sleep\,(i)$; if approved, this event prevents marking the passage of time due to actions; however, the passage of time may be triggered by other events;

– in the environment's history, it means that i malfunctioned, was not woken up and did not perform any actions prescribed by i's protocol; moreover, the passage of time was not marked due to $go(i)$ or $sleep\,(i)$ but may have been marked due to other events.

2.2 Transition Function

There are multiple consistency restrictions to be imposed on the histories to ensure that information from a local history h_i, incomplete as it might be, does not contradict what is recorded by the environment objectively and omnisciently in h_ϵ. We now start introducing these restrictions, dividing them into several types according to the parts of the framework responsible for upholding them.

Our general ideology is that (correct) agents act to achieve a particular goal, for which the agent's protocol takes the responsibility. The environment plays a triple role. Firstly, it is an impartial physical medium enforcing the consistency of histories and the laws of causality. In particular the environment increments all histories, local and global, in a coherent way and filters out events that are considered "physically" impossible, such as a (non-Byzantine) delivery of a message that was never sent. Secondly, the environment is the source of external unbiased indeterminacy: it simply records all possibilities the future can have in store. "Unbiased" here means that possibilities should not be omitted in the interests of short-term expedience, such as achieving the worst-case scenario. The latter is done using the third part of the environment that performs the non-deterministic choice and is designated the *adversary*. In other words, it should not be possible to guarantee avoiding good (or any other possible) choices but it should be possible to avoid them by chance.

Since agents are assumed not to have the complete overview of the system, agent i's protocol P_i can only rely on i's local view, i.e., its local state at the moment. In particular, it is crucial for implementing asynchronous agents that P_i cannot use timestamp t as a parameter.

Conversely, the protocol P_ϵ of the omniscient environment can use timestamp t, in fact using t is necessary to correctly forge GMIs for sent Byzantine messages. At the same time, P_ϵ should not depend on the current (global) state to preserve the unbiased representation of the physical laws and to facilitate proofs of properties of our framework.

For instance, a (synchronous-communication) receiver can be modeled by an environment protocol that "listens on all frequencies", i.e., attempts to deliver all messages at all times. However, all messages that have not been sent are filtered out and not delivered. Thus, the behavior of the system does depend on the global state, but the environment's protocol does not.

Definition 2.2.1 (Coherent events)**.** Let $t \in \mathbb{N}$ be a timestamp. A set $S \subset GEvents$ of events is called t**-coherent** if it satisfies the following conditions:

1. for any $fake\,(i, gsend(i, j, \mu, id) \mapsto A) \in S$, the GMI $id = id(i, j, \mu, k, t)$ for some $k \in \mathbb{N}$;

2. for any $i \in \mathcal{A}$ at most one of $go(i)$, $sleep\,(i)$, and $hibernate\,(i)$ is present in S;

3. for any $i \in \mathcal{A}$ and any $e \in Ext_i$ at most one of $external\,(i, e)$ and $fake\,(i, external\,(i, e))$ is present in S;

4. for any $grecv(i, j, \mu, id_1) \in S$, no event of the form $fake\,(i, grecv(i, j, \mu, id_2))$ belongs to S for any $id_2 \in \mathbb{N}$;

5. for any $fake\,(i, grecv(i, j, \mu, id_1)) \in S$, no event of the form $grecv(i, j, \mu, id_2)$ belongs to S for any $id_2 \in \mathbb{N}$;

Remark 2.2.2. It is possible that two copies $grecv(i, j, \mu, id_1)$ and $grecv(i, j, \mu, id_2)$ of the same message, possibly sent at different rounds, arrive simultaneously. While the receiving agent i would only know that the message μ from j is received, without being aware of various copies or their multiplicity, the ability to receive multiple copies is important, for instance, when message delivery has to be reliable.

We also leave a possibility of $fake\,(i, grecv(i, j, \mu, id_1))$ and $fake\,(i, grecv(i, j, \mu, id_2))$ at the same time making agent i falsely think that it received the message μ from j. While one such error makes all further ones redundant, there is no material difference in the system's behavior when two or more of such errors are present. Hence, to avoid unnecessary technical work, we leave this as a possibility. Needless to say, this possibility can always be precluded in specific protocols.

Lemma 2.2.3. *Any subset of a t-coherent set is itself t-coherent.*

Definition 2.2.4 (Protocol)**.**

1. A (non-deterministic) **protocol for agent** $i \in \mathcal{A}$ is any function

$$P_i \colon \mathscr{L}_i \to 2^{2^{\overline{Actions_i}}} \setminus \{\varnothing\} \tag{2.9}$$

   For a local state $h_i \in \mathscr{L}_i$ of agent i, each member $S \in P_i\,(h_i)$ is a subset of $\overline{Actions_i}$ and represents one of non-deterministic choices prescribing a set of actions for i in this local state. Note that $P_i\,(h_i) \neq \varnothing$ means that an agent always has at least one such choice S, which might be to perform no actions if $S = \varnothing$.

2. Given individual agents' protocols $P_1, \ldots, P_n$, their **joint protocol** is a function of global states that returns a tuple of action sets computed according to agent's protocols, one set per agent: for a global state $h = (h_\epsilon, h_1, \ldots, h_n)$,

$$P\,(h) := (P_1\,(h_1), \ldots, P_n\,(h_n)) \tag{2.10}$$

3. A (non-deterministic) **protocol for the environment** is any function

$$P_\epsilon \colon \mathbb{N} \longrightarrow 2^{2^{GEvents}} \setminus \{\varnothing\} \tag{2.11}$$

such that every $S \in P_\epsilon(t)$ is t-coherent. In other words, for each $t \in \mathbb{N}$, each member $S \in P_\epsilon(t)$ is a t-coherent subset of $GEvents$, i.e.,

$$\begin{aligned} S \quad \subset \quad & \{grecv(i,j,\mu,id) \mid i,j \in \mathcal{A}, \mu \in Msgs, id \in \mathbb{N}\} \quad \sqcup \quad \{go(i) \mid i \in \mathcal{A}\} \quad \sqcup \\ & \{external(i,e) \mid i \in \mathcal{A}, e \in Ext_i\} \quad \sqcup \quad \{sleep(i) \mid i \in \mathcal{A}\} \quad \sqcup \\ & \{fake(i,E) \mid i \in \mathcal{A}, E \in \overline{GEvents_i}\} \quad \sqcup \quad \{hibernate(i) \mid i \in \mathcal{A}\} \quad \sqcup \\ & \left\{fake(i, A \mapsto A') \mid i \in \mathcal{A}, A, A' \in \{\text{🗓}\} \sqcup \overline{GActions_i}\right\}, \end{aligned}$$

and represents one of the non-deterministic possibilities for what can happen in the system at time t.5. The two conditions mean that no correct event can be accompanied by its faulty version and any faulty send has a correctly computed GMI. Note that $P_\epsilon(t) \neq \varnothing$ means that the environment always has at least one such choice S, which might be to impose no events if $S = \varnothing$.

Definition 2.2.5. For $\sigma \in \{0,1\}$ and a set X we define

$$X^\sigma := \begin{cases} X & \text{if } \sigma = 1, \\ \varnothing & \text{if } \sigma = 0. \end{cases}$$

Notation 2.2.6. We denote by $\mathscr{C}$ the set of all joint protocols.

Notation 2.2.7. We denote by $\mathscr{C}_\epsilon$ the set of all environment protocols.

Remark 2.2.8. All sets produced by protocols in $\mathscr{C}_\epsilon$ at time t are t-coherent.

Remark 2.2.9. Depending on the intended strength and type of Byzantine agents, we may further restrict the set of functions allowed as protocols of the environment.

Remark 2.2.10 (Time sensitive actions)**.** The dependence of the environment's protocol on time enables modelling of time-sensitive actions. For instance, such a protocol can implement a global prohibition on message delivery during designated quiet time.

Remark 2.2.11 (Life must go on)**.** Both the environment and each of the agents always has at least one (possibly empty) set of actions/events at its disposal (*no-apocalypse clause*). The situation when the agents crash and cannot proceed further can still be represented, e.g., by designating a special crash action.

As we saw, Byzantine send and receive events, as well as correct receive events, are both initiated and performed by the environment, which is why we chose to represent these events as fully formed, i.e., supplied with a GMI, from the very beginning. In fact, correct receive events must contain a GMI to determine whether a message with such a GMI was sent earlier, which is part of the definition of its correctness. The situation with correct send actions is different: they are initiated by an agent, who must remain unaware of a GMI, but the message is propagated to the recipient by the environment. Thus, when an active agent sends a message, it must first be transformed from the local to the global view. This task is performed by the *labeling functions* $label_i$ for each $i \in \mathcal{A}$. Similarly, when the message, correct or fake, is delivered or a fake event occurs for an agent, the event must be transformed into its local format before being recorded in the local history. This is done by the "*reverse*" function $label^{-1}$.

Definition 2.2.12 (Labeling functions)**.** For an agent $i \in \mathcal{A}$, we define a function

$$label_i \colon \overline{Actions_i} \times \mathbb{N} \longrightarrow \overline{GActions_i}$$

converting the local representation of actions to the global format as follows:

$$label_i\,(a,t) := \begin{cases} gsend(i,j,\mu,id(i,j,\mu,k,t)) & \text{if } a = send(j,\mu_k) \\ internal\,(i,a) & \text{if } a \in Int_i \end{cases}$$

We collect all these functions into one tuple $label := (label_1, \ldots, label_n)$.

We also define a function converting actions and events from the global format into the local ones. This function is applied after all fake events are already turned into their benign counterparts and 📅's are removed by a separate function. Thus, this function does not deal with fake events or no events.

$$label^{-1} \colon \overline{GActions} \sqcup \overline{GEvents} \longrightarrow \overline{Actions} \sqcup \overline{Events}$$

as follows:

$$label^{-1}\,(U) := \begin{cases} send(j,\mu_k) & \text{if } U = gsend(i,j,\mu,id(i,j,\mu,k,t)) \\ send(j,\mu_0) & \text{if } U = gsend(i,j,\mu,M) \text{ and } M \neq id(i,j,\mu,k,t) \text{ for any } k,t \in \mathbb{N} \\ recv(j,\mu) & \text{if } U = grecv(i,j,\mu,id) \\ a & \text{if } U = internal\,(i,a) \\ e & \text{if } U = external\,(i,e) \end{cases}$$

Function $label^{-1}$ extends to sets in the standard way: $label^{-1}(X) := \{label^{-1}(U) \mid U \in X\}$. For the functions $label_i$, we distribute the timestamp parameter to all elements of the set: $label_i\,(X,t) := \{label_i\,(a,t) \mid a \in X\}$.

Remark 2.2.13. The injectivity of the function id used in $label_i$ ensures that each message is unique from the point of view of the environment.

Remark 2.2.14. The second clause in the definition of $label^{-1}\,(U)$ is mostly cosmetic: we make GMIs id unforgeable, and, hence, this clause will never be used. It is added solely to make the function $label^{-1}\,(U)$ total, thus, avoiding irrelevant complications stemming from the use of potentially partial functions.

Definition 2.2.15 (Non-deterministic choice for protocols)**.** Given a global history $h = (h_\epsilon, h_1, \ldots, h_n) \in \mathscr{G}$ and protocols $P_\epsilon \in \mathscr{C}_\epsilon$ for the environment and $(P_1, \ldots, P_n) \in \mathscr{C}$ for the agents, we obtain, for agent $i \in \mathcal{A}$ and timestamp $t \in \mathbb{N}$, the sets of global actions and events to be attempted at the global state h at t, i.e., in the round $t.5$, as follows:

1. Events imposed by the environment are a t-coherent set

$$\alpha_\epsilon^t = X_\epsilon \tag{2.12}$$

for some set $X_\epsilon \in P_\epsilon\,(t)$ non-deterministically chosen by the adversary.

2. Actions agent $i \in \mathcal{A}$ would perform if woken up are a set

$$\alpha_i^{h,t} = \mathit{label}_i\,(X_i, t) \tag{2.13}$$

for some set $X_i \in P_i\,(h_i)$ non-deterministically chosen by the adversary.

3. All these choices are combined in the **joint attempted action**

$$\alpha^{h,t} := (\alpha_\epsilon^t, \alpha_1^{h,t}, \ldots, \alpha_n^{h,t}).$$

Among the events α_ϵ^t we distinguish the following subsets for each agent $i \in \mathcal{A}$:

1. *Regular events* for $i \in \mathcal{A}$

$$\overline{\alpha}_{\epsilon_i}^t := \alpha_\epsilon^t \cap \overline{\mathit{GEvents}}_i = \{\mathit{grecv}(i, j, \mu, id) \in \alpha_\epsilon^t \mid j \in \mathcal{A}, \mu \in \mathit{Msgs}, id \in \mathbb{N}\} \sqcup \{\mathit{external}\,(i, e) \in \alpha_\epsilon^t \mid e \in \mathit{Ext}_i\} \tag{2.14}$$

2. Instructions regarding waking up for $i \in \mathcal{A}$

$$\alpha_{g_i}^t = \alpha_\epsilon^t \cap \mathit{SysEvents}_i \tag{2.15}$$

3. Fake events for agent $i \in \mathcal{A}$, including those mimicking the agent's actions:

$$\alpha_{b_i}^t := \alpha_\epsilon^t \cap \mathit{BEvents}_i = \left\{\mathit{fake}\,(i, A \mapsto A') \in \alpha_\epsilon^t \mid A, A' \in \{\boxminus\} \sqcup \overline{\mathit{GActions}}_i\right\} \sqcup \{\mathit{fake}\,(i, E) \in \alpha_\epsilon^t \mid E \in \overline{\mathit{GEvents}}_i\} \tag{2.16}$$

4. Instructions making agent i Byzantine:

$$\alpha_{f_i}^t := \alpha_{b_i}^t \sqcup \left(\alpha_\epsilon^t \cap \{\mathit{sleep}\,(i), \mathit{hibernate}\,(i)\}\right) \tag{2.17}$$

Note that $\mathit{sleep}\,(i)$ and $\mathit{hibernate}\,(i)$ may be present in both $\alpha_{g_i}^t$ and $\alpha_{f_i}^t$ Finally, we define

$$\overline{\alpha}_\epsilon^t := \bigsqcup_{i \in \mathcal{A}} \overline{\alpha}_{\epsilon_i}^t \qquad \alpha_g^t := \bigsqcup_{i \in \mathcal{A}} \alpha_{g_i}^t \qquad \alpha_b^t := \bigsqcup_{i \in \mathcal{A}} \alpha_{b_i}^t \qquad \alpha_f^t := \bigsqcup_{i \in \mathcal{A}} \alpha_{f_i}^t$$

Lemma 2.2.16. *Given a global history $h \in \mathscr{G}$ and protocols P_ϵ for the environment and $P_1, \ldots, P_n$ for the agents, for agent $i \in \mathcal{A}$ and for timestamp $t \in \mathbb{N}$, we have that*

$$\alpha_\epsilon^t \subset \mathit{GEvents} \qquad \text{and} \qquad \alpha_i^{h,t} \subset \overline{\mathit{GActions}}_i.$$

The environment should not create impossible situations. Most of them are prohibited by the definition of environment's protocol. For instance, an event recorded by an agent cannot both happen and not happen. Accordingly, the environment can only try one of these two possibilities but not both at the same time.

There is, however, one common type of "causal" impossibility that is not restricted to environment alone or a particular moment in time: a message *cannot* be delivered without

being previously[10] sent. Due to our necessity to make the environment's protocol independent of the global history, which is crucial for many proofs, the environment's protocol cannot check whether the message was actually sent in previous rounds (based on the global history) or in the current round (based in part on the actions chosen by the adversary for the sending agent, the presence of the *go* command for it, and other events imposed on this agent). But correctly receiving an unsent message would break the laws of causality. Since these events cannot be handled by the environment alone, we create a special filter that weeds them out.

To simplify notation we introduce the following abbreviation:

Definition 2.2.17 (Active/passive, aware/unaware)**.** For a set $X \subseteq GEvents$ we define

$$active(i, X) := \begin{cases} t & \text{if } X \cap SysEvents_i = \{go(i)\}, \\ f & otherwise. \end{cases} \tag{2.18}$$

$$aware(i, X) := \begin{cases} t & \text{if } \varnothing \neq X \cap SysEvents_i \in \{\{go(i)\}, \{sleep\,(i)\}\}, \\ f & otherwise. \end{cases} \tag{2.19}$$

For readability's sake we write $active(i, X)$ instead of $active(i, X) = t$ and $passive(i, X)$ instead of $active(i, X) = f$, as well as $aware(i, X)$ instead of $aware(i, X) = t$ and $unaware(i, X)$ instead of $aware(i, X) = f$.

Corollary 2.2.18. *Given that for any $S \in P_\epsilon\,(t)$, at most one of system actions can be present*

$$passive(i, S) \iff S \cap SysEvents_i \in \{\varnothing, \{sleep\,(i)\}, \{hibernate\,(i)\}\}, \tag{2.20}$$

$$unaware(i, S) \iff S \cap SysEvents_i \in \{\varnothing, \{hibernate\,(i)\}\}. \tag{2.21}$$

$$active(i, X) \implies aware(i, X) \tag{2.22}$$

$$unaware(i, S) \implies passive(i, S) \tag{2.23}$$

Definition 2.2.19 (Event and action filter functions)**.** We define an **event filter function for Byzantine agents**

$$filter_\epsilon^B : \mathscr{G} \times 2^{GEvents} \times 2^{\overline{GActions}_1} \times \cdots \times 2^{\overline{GActions}_n} \longrightarrow 2^{GEvents}$$

as follows. Given a global history h, a set X_ϵ of events attempted by the environment (chosen by the adversary) and sets X_i of actions to be performed by the agents (also chosen by the adversary), the function returns the set of all attempted events that are "causally" possible as the set of events to be actually performed by the environment. Formally, for a set $X_\epsilon \subset GEvents$, sets $X_i \subset \overline{GActions}_i$ for each agent $i \in \mathcal{A}$, and a global history $h = (h_\epsilon, h_1, \ldots, h_n) \in \mathscr{G}$, we define

$$\begin{aligned} filter_\epsilon^B\,(h, X_\epsilon, X_1, \ldots, X_n) \quad := \\ X_\epsilon \quad \setminus \quad \Big\{ grecv(j, i, \mu, id) \quad | \quad gsend(i, j, \mu, id) \notin h_\epsilon \quad \wedge \\ (\forall A \in \{\text{📅}\} \sqcup \overline{GActions}_i)\, fake\,(i, gsend(i, j, \mu, id) \mapsto A) \notin h_\epsilon \quad \wedge \\ (gsend(i, j, \mu, id) \notin X_i \vee passive(i, X_\epsilon)) \quad \wedge \\ (\forall A \in \{\text{📅}\} \sqcup \overline{GActions}_i)\, fake\,(i, gsend(i, j, \mu, id) \mapsto A) \notin X_\epsilon \Big\} \end{aligned} \tag{2.24}$$

[10] Here *previously* sent means sent in one of the preceding rounds or in the same round, whether correctly or in a Byzantine fashion. On the other hand, when an agent mistakenly thinks the message was sent, it is not considered sent previously.

In addition, we define **action filter functions for Byzantine agents** $i \in \mathcal{A}$

$$filter_i^B : 2^{\overline{GActions}_1} \times \cdots \times 2^{\overline{GActions}_n} \times 2^{GEvents} \longrightarrow 2^{\overline{GActions}_i}$$

as follows. Given a set of actions $X_j \subset \overline{GActions}_j$ prescribed for each agent $j \in \mathcal{A}$ by its protocol (as chosen by the adversary) and a set of events $X_\epsilon \subset GEvents$ that are performed by the environment, we define an all-or-nothing

$$filter_i^B\left(X_1, \ldots, X_n, X_\epsilon\right) = \begin{cases} X_i & \text{if } active(i, X_\epsilon) \\ \varnothing & \text{otherwise} \end{cases} \tag{2.25}$$

It is obvious from these definitions that these are indeed filter functions on X_ϵ and X_i respectively:

$$filter_\epsilon^B\left(h, X_\epsilon, X_1, \ldots, X_n\right) \subset X_\epsilon \tag{2.26}$$

$$filter_i^B\left(X_1, \ldots, X_n, X_\epsilon\right) \subset X_i \tag{2.27}$$

Thus, after protocols provided a range of possible event/action collections $P_\epsilon(t)$ and $P_i(h_i)$ and the adversary chose the collection α_ϵ^t of events to be attempted by the environment and collections $\alpha_i^{h,t}$ of actions to be performed by each agent if it is awoken, the filter functions determine which of these events and actions are to actually happen during the round. For this second stage, the resulting sets are called β-sets by analogy with α-sets.

Remark 2.2.20. While it is clear that an agent observing an event that actually happens cannot be mistaken about it, the situation with actions is more complex because the agent may not be certain about the exact action it performs. We chose to leave the agent with the widest variety of possibilities:

- several faulty actions can be mistaken for one (including the no-op action 🗓 or some action that actually was performed): $fake\,(i, A_1 \mapsto A)$, ..., $fake\,(i, A_m \mapsto A)$ are generally compatible;
- one faulty action can be mistaken for several actions: $fake\,(i, A \mapsto A_1)$, ..., $fake\,(i, A \mapsto A_m)$ are generally compatible (this can also happen when A was actually performed or when A is the no-op action 🗓).

In other words, agents can be confused not only regarding which actions they have performed but also regarding how many have been performed. Further, the agent may think it has done a lot without doing anything or vice versa may think it has done nothing despite frantic activity.

Remark 2.2.21. For the case of general Byzantine agents, we could directly define a local version of the **action filter function** $filter_i^B : 2^{\overline{GActions}_i} \times 2^{GEvents} \longrightarrow 2^{\overline{GActions}_i}$ because the choices of other agents do not affect the filtering for general Byzantine agents. But we will need the definition of other variants of $filter_i$ to be $(n+1)$-ary functions to refine the model to implement, for example, *rendez-vous communication*.

Definition 2.2.22. For a global history $h \in \mathscr{G}$, a timestamp $t \in \mathbb{N}$, a tuple of requested actions and events $\alpha^{h,t} = (\alpha_\epsilon^t, \alpha_1^{h,t}, \ldots, \alpha_n^{h,t})$, and agent $i \in \mathcal{A}$,

1. $\beta_\epsilon^{h,\alpha^{h,t}} := filter_\epsilon^B\left(h, \quad \alpha_\epsilon^t, \quad \alpha_1^{h,t}, \quad \ldots, \quad \alpha_n^{h,t}\right)$
2. $\beta_i^{h,\alpha^{h,t}} := filter_i^B\left(\alpha_1^{h,t}, \ldots, \alpha_n^{h,t}, \beta_\epsilon^{h,\alpha^{h,t}}\right)$
3. $\beta^{h,\alpha^{h,t}} := (\beta_\epsilon^{h,\alpha^{h,t}}, \beta_1^{h,\alpha^{h,t}}, \ldots, \beta_n^{h,\alpha^{h,t}})$

As for α_ϵ^t we also distinguish the following subsets of $\beta_\epsilon^{h,\alpha^{h,t}}$ for each agent $i \in \mathcal{A}$:

1. *Regular events* for agent i:

$$\overline{\beta}_{\epsilon_i}^{h,\alpha^{h,t}} := \beta_\epsilon^{h,\alpha^{h,t}} \cap \overline{GEvents_i} = \{grecv(i,j,\mu,id) \in \beta_\epsilon^{h,\alpha^{h,t}} \mid j \in \mathcal{A}, \mu \in Msgs, id \in \mathbb{N}\} \sqcup \{external\,(i,e) \in \beta_\epsilon^{h,\alpha^{h,t}} \mid e \in Ext_i\} \subset \overline{\alpha}_{\epsilon_i}^t \quad (2.28)$$

2. Instructions regarding waking up agent i:

$$\beta_{g_i}^{h,\alpha^{h,t}} := \beta_\epsilon^{h,\alpha^{h,t}} \cap SysEvents_i \subset \alpha_{g_i}^t \quad (2.29)$$

3. Fake events for agent i, including those mimicking the agent's actions:

$$\beta_{b_i}^{h,\alpha^{h,t}} := \beta_\epsilon^{h,\alpha^{h,t}} \cap BEvents_i = \left\{fake\,(i, A \mapsto A') \in \beta_\epsilon^{h,\alpha^{h,t}} \mid A, A' \in \{\text{📅}\} \sqcup \overline{GActions_i}\right\} \sqcup \{fake\,(i,E) \in \beta_\epsilon^{h,\alpha^{h,t}} \mid E \in \overline{GEvents_i}\} \subset \alpha_{b_i}^t \quad (2.30)$$

4. Instructions making agent i Byzantine:

$$\beta_{f_i}^{h,\alpha^{h,t}} := \beta_{b_i}^{h,\alpha^{h,t}} \sqcup \left(\beta_\epsilon^{h,\alpha^{h,t}} \cap \{sleep\,(i), hibernate\,(i)\}\right) \subset \alpha_{f_i}^t \quad (2.31)$$

Finally, we define

$$\overline{\beta}_\epsilon^{h,\alpha^{h,t}} := \bigsqcup_{i\in A} \overline{\beta}_{\epsilon_i}^{h,\alpha^{h,t}} \subset \overline{\alpha}_\epsilon^t \qquad \beta_g^{h,\alpha^{h,t}} := \bigsqcup_{i\in A} \beta_{g_i}^{h,\alpha^{h,t}} \subset \alpha_g^t$$
$$\beta_b^{h,\alpha^{h,t}} := \bigsqcup_{i\in A} \beta_{b_i}^{h,\alpha^{h,t}} \subset \alpha_b^t \qquad \beta_f^{h,\alpha^{h,t}} := \bigsqcup_{i\in A} \beta_{f_i}^{h,\alpha^{h,t}} \subset \alpha_f^t$$
$$\beta_{\epsilon_i}^{h,\alpha^{h,t}} := \overline{\beta}_{\epsilon_i}^{h,\alpha^{h,t}} \sqcup \beta_{g_i}^{h,\alpha^{h,t}} \sqcup \beta_{b_i}^{h,\alpha^{h,t}}$$

Remark 2.2.23. The filtering is split into two steps: first filtering events $\beta_\epsilon^{h,\alpha^{h,t}}$ and then filtering actions based on the results of event filtering $\beta_i^{h,\alpha^{h,t}} = filter_i\left(\alpha_1^{h,t}, \ldots, \alpha_n^{h,t}, \beta_\epsilon^{h,\alpha^{h,t}}\right)$. Such two-step filtering enables us to represent communication scenarios that rely on coordination among agents by making it possible to filter out *go* events that violate the coordination requirements.

Remark 2.2.24. Consider a global history $h \in \mathscr{G}$, a timestamp $t \in \mathbb{N}$, and a tuple of requested actions and events $\alpha^{h,t} = (\alpha_\epsilon^t, \alpha_1^{h,t}, \ldots, \alpha_n^{h,t})$. Then

$$\beta_\epsilon^{h,\alpha^{h,t}} = \overline{\beta}_\epsilon^{h,\alpha^{h,t}} \sqcup \beta_g^{h,\alpha^{h,t}} \sqcup \beta_b^{h,\alpha^{h,t}}.$$

Proposition 2.2.25. *Consider a global history $h \in \mathscr{G}$, a timestamp $t \in \mathbb{N}$, a tuple of requested actions and events $\alpha^{h,t} = (\alpha_\epsilon^t, \alpha_1^{h,t}, \ldots, \alpha_n^{h,t})$, and an agent $i \in \mathcal{A}$. Then action filter function $filter_i$ for agent i ensures that*

$$\begin{aligned} \beta_i^{h,\alpha^{h,t}} \neq \varnothing \quad &\Longrightarrow \quad active(i, \beta_\epsilon^{h,\alpha^{h,t}}); \\ \beta_i^{h,\alpha^{h,t}} \neq \varnothing \quad &\Longrightarrow \quad aware(i, \beta_\epsilon^{h,\alpha^{h,t}}). \end{aligned}$$

Proof. The first statement follows directly from Def. 2.2.22(2) and equation (2.25). The second statement follows from the first and (2.22). □

Once again, it is easy to see that

Lemma 2.2.26. *Given a global history $h \in \mathscr{G}$, a timestamp $t \in \mathbb{N}$, a tuple of requested actions and events $\alpha^{h,t} = (\alpha_\epsilon^t, \alpha_1^{h,t}, \ldots, \alpha_n^{h,t})$, and agent $i \in \mathcal{A}$*

$$\beta_\epsilon^{h,\alpha^{h,t}} \subset GEvents \qquad \text{and} \qquad \beta_i^{h,\alpha^{h,t}} \subset \overline{GActions_i}$$

It is important to separate the complete knowledge required of the environment to perform the transition from state to state from the limited local view that the agents have. In particular, it is a central assumption of distributed systems in general and of the proposed framework in particular that agents should not be able to tell the difference between an external event they actually observed and a fake external event their sensors mistakenly registered, nor between performing action A' and thinking they have performed A' when A was the actual action performed, as represented by $fake\,(i, A \mapsto A')$. In this respect, the agents can be viewed as malfunctioning drones rather than scheming moles: They always mean well but are sometimes prevented by the environment from behaving correctly. In such cases, they can juxtapose their own intentions with their perception of the resulting actions and events but cannot directly detect the environment's meddling.

Formally, this means that the local histories must be purged of

(1) *fake* modifiers,

(2) GMIs,

(3) controlling commands $go(i)$, $sleep\,(i)$, and $hibernate\,(i)$.

These tasks are performed by the localization function σ: both on the action/event level and on the set level:

Definition 2.2.27 (Localization function)**.** The function

$$\sigma\colon 2^{\overline{GActions}\sqcup GEvents} \longrightarrow 2^{\overline{Actions}\sqcup\overline{Events}}$$

is defined as follows

$$\sigma(X) := label^{-1}\Big((X \cap (\overline{GActions} \sqcup \overline{GEvents}))\cup \{E \mid (\exists i)\, fake\,(i,E) \in X\}\cup \{A' \neq \text{🗓} \mid (\exists i)(\exists A)\, fake\,(i, A \mapsto A') \in X\}\Big) \quad (2.32)$$

If $U \in \overline{GActions} \sqcup GEvents$ and $\sigma(\{U\}) \neq \varnothing$, i.e., U is not one of $go(i)$, $sleep\,(i)$, $hibernate\,(i)$, nor is a $fake\,(i, A \mapsto \text{🗓})$ for some i and A, we also write $\sigma(U)$ to denote the only element of the set $\sigma(\{U\})$, i.e., $\sigma(\{U\}) = \{\sigma(U)\}$.

The following lemma directly follows from the definitions of P_ϵ (Def. 2.2.4(3) and of t-coherence (Def. 2.2.1).

Lemma 2.2.28. *Given a global history $h \in \mathscr{G}$ and protocols $P_\epsilon \in \mathscr{C}_\epsilon$ for the environment and $P_1, \ldots, P_n$ for the agents, for any agent $i \in \mathcal{A}$ and for timestamp $t \in \mathbb{N}$,*

$$\sigma(\alpha^t_{b_i}) \cap label^{-1}\left(\overline{\alpha}^t_{\epsilon_i}\right) = \varnothing \quad (2.33)$$

$$|\alpha^t_{g_i}| \leq 1 \quad (2.34)$$

$$fake\,(i, gsend(i,j,\mu,id) \mapsto A) \in \alpha^t_{b_i} \implies (\exists k \in \mathbb{N})\, id = id(i,j,\mu,k,t) \quad (2.35)$$

The following properties are inherited from the α-sets because filtering does not add new things. The next lemma follows from Lemma 2.2.3.

Lemma 2.2.29. *Given a global history $h \in \mathscr{G}$ and protocols $P_\epsilon \in \mathscr{C}_\epsilon$ for the environment and $P_1, \ldots, P_n$ for the agents, for any agent $i \in \mathcal{A}$ and for timestamp $t \in \mathbb{N}$, the set $\beta^{h,\alpha^{h,t}}_\epsilon$ is t-coherent, in particular,*

$$\sigma(\beta^{h,\alpha^{h,t}}_{b_i}) \cap label^{-1}\left(\overline{\beta}^{h,\alpha^{h,t}}_{\epsilon_i}\right) = \varnothing \quad (2.36)$$

$$|\beta^{h,\alpha^{h,t}}_{g_i}| \leq 1 \quad (2.37)$$

$$fake\,(i, gsend(i,j,\mu,id) \mapsto A) \in \beta^{h,\alpha^{h,t}}_{b_i} \implies (\exists k \in \mathbb{N})\, id = id(i,j,\mu,k,t) \quad (2.38)$$

Remark 2.2.30. Since $\left|\beta^{h,\alpha^{h,t}}_{g_i}\right| \leq 1$ in all cases, we write $go(i) \in \beta^{h,\alpha^{h,t}}_{g_i}$ instead of the equivalent statement $\beta^{h,\alpha^{h,t}}_{g_i} = \{go(i)\}$.

The last piece of the puzzle is state update functions that record the events and actions performed in a round into all the histories.

Definition 2.2.31 (State update functions)**.** Given a global history $h = (h_\epsilon, h_1, \ldots, h_n) \in \mathscr{G}$, a tuple of performed actions/events $X = (X_\epsilon, X_1, \ldots, X_n) \in 2^{GEvents} \times 2^{\overline{GActions_1}} \times \cdots \times$

$2^{\overline{GActions_n}}$, we use the following abbreviation $X_{\epsilon_i} = X_\epsilon \cap GEvents_i$ for each $i \in \mathcal{A}$. Agents i's update function

$$update_i \colon \mathscr{L}_i \times 2^{\overline{GActions_i}} \times 2^{GEvents} \to \mathscr{L}_i$$

outputs a new local history from $\mathscr{L}_i$ based on i's actions X_i and environment-controlled events X_ϵ as follows:

$$update_i\left(h_i, X_i, X_\epsilon\right) := \begin{cases} h_i & \text{if } \sigma(X_{\epsilon_i}) = \varnothing \text{ and } unaware(i, X_\epsilon) \\ \left[\sigma\left(X_{\epsilon_i} \sqcup X_i\right)\right] : h_i & \text{otherwise} \end{cases} \tag{2.39}$$

where : represents sequence concatenation. Similarly, the environment's state update function

$$update_\epsilon \colon \mathscr{L}_\epsilon \times \left(2^{GEvents} \times 2^{\overline{GActions_1}} \times \cdots \times 2^{\overline{GActions_n}}\right) \to \mathscr{L}_\epsilon$$

outputs a new state of the environment based on X_ϵ:

$$update_\epsilon\left(h_\epsilon, X\right) := \left(X_\epsilon \sqcup X_1 \sqcup \cdots \sqcup X_n\right) \colon h_\epsilon \tag{2.40}$$

Thus, the global state is modified as follows:

$$update\left(h, X\right) := \left(update_\epsilon\left(h_\epsilon, X\right), update_1\left(h_1, X_1, X_\epsilon\right), \ldots, update_n\left(h_n, X_n, X_\epsilon\right)\right) \tag{2.41}$$

Remark 2.2.32. The first clause in (2.39) corresponds to the situation when the agent is not woken up by actions or events. In particular, $unaware(i, X_\epsilon)$ states that i is denied both actions in the round and even awareness of the round itself. Virtually always function $update_i$ will be applied to $X_\epsilon = \beta_\epsilon^{h,\alpha^{h,t}}$, which is a t-coherent set. Thus, the condition $unaware(i, \beta_\epsilon^{h,\alpha^{h,t}})$ is equivalent to $\beta_\epsilon^{h,\alpha^{h,t}} \cap SysEvents_i \in \{\varnothing, \{hibernate\,(i)\}\}$, in other words,

$$unaware(i, \beta_\epsilon^{h,\alpha^{h,t}}) \quad \Longleftrightarrow \quad \beta_\epsilon^{h,\alpha^{h,t}} \cap SysEvents_i \subset \{hibernate\,(i)\}. \tag{2.42}$$

Following [FHMV95b], we define transition functions as follows:

Definition 2.2.33 (Transition function)**.** For agents' protocols $P = (P_1, \ldots, P_n)$ and a protocol P_ϵ of the environment, we define a **Byzantine transition function**

$$\tau^B_{P_\epsilon,P} \quad : \quad 2^{GEvents} \times 2^{\overline{GActions_1}} \times \cdots \times 2^{\overline{GActions_n}} \quad \to \quad (\mathscr{G} \quad \to \quad \mathscr{G})$$

as a function that outputs a **global state transformer function**

$$\tau^B_{P_\epsilon,P}(Y) \colon \mathscr{G} \to \mathscr{G}$$

from global states to global states given joint attempted actions/events

$$Y \quad \in \quad 2^{GEvents} \times 2^{\overline{GActions_1}} \times \cdots \times 2^{\overline{GActions_n}}$$

defined as follows. For a global state $h = (h_\epsilon, h_1, \ldots, h_n) \in \mathscr{G}$ and such joint attempted actions/events Y we consider two possibilities:

- if $Y = \alpha^{h,|h|} = \left(\alpha_\epsilon^{|h|}, \alpha_1^{h,|h|}, \ldots, \alpha_n^{h,|h|}\right)$ for some $\alpha_\epsilon^{|h|} \in P_\epsilon\left(|h|\right)$ and some $X_i \in P_i\left(h_i\right)$ for each $i \in \mathcal{A}$ such that $\alpha_i^{h,|h|} = label_i\left(X_i, |h|\right)$ then we define

$$\tau^B_{P_\epsilon,P}(Y)(h) := update\left(h, \quad \beta^{h,\alpha^{h,|h|}}\right) \tag{2.43}$$

  where the β-sets are computed from $\alpha^{h,|h|}$ by Def. 2.2.22 and the *update* function is defined in (2.41);

- otherwise, we define $\tau^B_{P_\epsilon,P}(Y)(h) = h$.[11]

Remark 2.2.34. By a slight abuse of notation, we write $h' \in \tau^B_{P_\epsilon,P}(h)$ to mean that there is a protocol-conformant set of joint actions $\alpha^{h,|h|}$ satisfying the first clause of the above definition such that $\tau^B_{P_\epsilon,P}(\alpha^{h,|h|})(h) = h'$.

2.3 Runs and Contexts

As already mentioned, integer timestamps are used exclusively to take snapshots of the local and global states. A sequence of such snapshots as the time progresses is called a *run*. Our goal is to model systems that are, in general, asynchronous, meaning that the agents can neither know the global time nor count the number of rounds since the beginning of the run. Without loss of generality, we consider runs that encompass the whole infinite set $\mathbb{N}$ of timestamps.

Definition 2.3.1 (Run)**.** A **run** is a function that assigns a global state to each integer timestamp.

$$r : \mathbb{N} \longrightarrow \mathscr{G} \tag{2.44}$$

We denote the set of all runs by R. The part of the run that an agent i can see is called i's **local view**. It is a function that assigns i's local state to each integer timestamp.

$$r_i \colon \mathbb{N} \longrightarrow \mathscr{L}_i \tag{2.45}$$

It is clear that each local view r_i is uniquely determined by the run r:

$$r_i\left(t\right) := \pi_{i+1} r\left(t\right)$$

where π_j is the jth projection function for tuples/sequences. Similarly, we define the environment's history

$$r_\epsilon \colon \mathbb{N} \longrightarrow \mathscr{L}_\epsilon$$

to be

$$r_\epsilon\left(t\right) := \pi_1 r\left(t\right)$$

Definition 2.3.2. For a set $X \subset \mathcal{A} \times \mathbb{N}$ of nodes we define the upper **time bound** $T(X)$ to be the largest $T \in \mathbb{N}$ such that $(i, T) \in X$ for some agent $i \in \mathcal{A}$ if such a T exists. $T(\varnothing)$ is defined to be 0. A set X is called **bounded** if it has a time bound or **unbounded** otherwise.

[11] The latter case will never be used and is only provided to make the transition function total.

Each agent initially starts off in a correct state and may become Byzantine when its actions stop being dictated by the protocol or its perception of events is compromised. Thus, it is more precise to talk about Byzantine states of agents, i.e., about Byzantine nodes $(i,t) \in \mathcal{A} \times \mathbb{N}$ instead of announcing the agents themselves to be universally Byzantine. Local timestamps (nodes) directly resulting from a violation of the agent's protocol or from a Byzantine event, including a Byzantine system event, are called *Bad*. All nodes of an agent starting from the first *Bad* local timestamp are called *Failed*. Recall that time in global histories h is represented by $|h|$.

Definition 2.3.3. Consider a global history $h = (h_\epsilon, h_1, \ldots, h_n) \in \mathscr{G}$ of length $|h|$, so that $h_\epsilon = [\Lambda_{|h|}, \ldots, \Lambda_1]$. We define the sets of *Bad* and *Failed* nodes

$$\begin{aligned} Bad\,(h) &:= \{(i,t) \in \mathcal{A} \times \mathbb{N} \mid \Lambda_t \cap (BEvents_i \sqcup \{sleep\,(i), hibernate\,(i)\}) \neq \varnothing\} \\ Failed\,(h) &:= \{(i,t) \in \mathcal{A} \times \mathbb{N} \mid (\exists t' \leq t)\,(i,t') \in Bad\,(h)\} \end{aligned}$$

if $|h| > 0$ and $Bad\,(h) = Failed\,(h) := \varnothing$ otherwise.

Remark 2.3.4. For any global history $h \in \mathscr{G}$, $Bad\,(h) \subset Failed\,(h)$. Indeed, the former represents nodes that experienced a malfunction in the immediately preceding round, whereas the latter is comprised of nodes with some malfunction possibly further in the past.

Definition 2.3.5. For a run $r \in R$, timestamp $t \in \mathbb{N}$, and bounded set $X \subset \mathcal{A} \times \mathbb{N}$ of nodes, we define

$$\begin{aligned} Bad\,(r,t) &:= Bad\,(r\,(t)) & Bad_X\,(r) &:= X \cap Bad\,(r, T(X)) \\ Failed\,(r,t) &:= Failed\,(r\,(t)) & Failed_X\,(r) &:= X \cap Failed\,(r, T(X)) \end{aligned}$$

For an unbounded set $X \subset \mathcal{A} \times \mathbb{N}$ of nodes, we define

$$Bad_X\,(r) := X \cap \left(\bigcup_{t=1}^{\infty} Bad\,(r,t)\right) \qquad Failed_X\,(r) := X \cap \left(\bigcup_{t=1}^{\infty} Failed\,(r,t)\right)$$

Remark 2.3.6. The unions in the unbounded case begin from $t = 1$ because for any run r, we have $Bad\,(r,0) = Failed\,(r,0) = \varnothing$.

Remark 2.3.7. The definition of $Bad_X\,(r)$ and $Failed_X\,(r)$ for unbounded sets X is compatible with that for bounded sets, when applied to bounded sets X. The benefit of the latter definition is that it is efficiently computable.

Remark 2.3.8. For agents' protocols P and the environment's protocol P_ϵ, we are mostly interested in runs $r \in R$ that are built according to these protocols by some transition function. For the time being, we use the Byzantine transition function $\tau^B_{P_\epsilon,P}$, i.e., consider runs that begin from a proper initial state and such that for each timestamp $t \in \mathbb{N}$,

$$r\,(t+1) \in \tau^B_{P_\epsilon,P}\,(r\,(t)) \tag{2.46}$$

Sometimes we call such runs $\tau^B_{P_\epsilon,P}$-**transitional**, or simply **transitional**. For such a transitional run r, we denote its initial state by $r\,(0)$ and the global state after round $(t-1).5$ is $r\,(t)$.

It immediately follows from (2.43), (2.40), and Def. 2.3.2 that

Proposition 2.3.9. *For any transitional run r,*

$$Bad\,(r,t) \subset Bad\,(r,t+1) \qquad \textit{and} \qquad Failed\,(r,t) \subset Failed\,(r,t+1).$$

In addition, for $X \subset X'$,

$$Bad_X\,(r) \subset Bad_{X'}\,(r) \qquad \textit{and} \qquad Failed_X\,(r) \subset Failed_{X'}\,(r),$$

independent of whether both sets are bounded, X is bounded while X' is not, or both sets are unbounded.

Proof. The crucial observation is that, for transitional runs, $r\,(t+1)$ is either equal to $r\,(t)$ or obtained by prepending it. In either case, $r\,(t+1)$ contains $r\,(t)$ without modifications. □

Remark 2.3.10. In the interests of generality and modularity of concepts, we defined the transition function in terms of arbitrary histories. For the case of histories comprising a transitional run, the notation can be simplified. We will now provide a concise digest of one step of transition for transitional runs (see also Fig. 2.1) with the dual purpose: to give a compact summary of the procedure and introduce a simpler notation mostly used in the rest of our work. This will also form a crucial part of the notion of *(weak) consistency with a context and a joint protocol* in Def. 2.3.27 after we introduce all parts comprising a context.

In Def. 2.2.33, we defined the basic Byzantine transition function $\tau^B_{P_\epsilon,P}$ based on protocols P of the agents and P_ϵ of the environment. As already mentioned, we will sometimes need to change the filtering phase of the transition function. Hence, we leave the exact details of the transition as a parameter τ that converts protocols into a transition function.

Definition 2.3.11 (Transition template)**.** A **transition template**

$$\tau\colon \mathscr{C}_\epsilon \times \mathscr{C} \to \left(2^{GEvents} \times 2^{\overline{GActions_1}} \times \cdots \times 2^{\overline{GActions_n}} \;\to\; (\mathscr{G} \;\to\; \mathscr{G})\right) \tag{2.47}$$

is a two-place function that takes a protocol $P_\epsilon \in \mathscr{C}_\epsilon$ of the environment and a joint agents' protocol $P \in \mathscr{C}$ and outputs a **transition function** $\tau(P_\epsilon, P)$, which we denote by $\tau_{P_\epsilon,P}$

$$\tau_{P_\epsilon,P} \;:\; 2^{GEvents} \times 2^{\overline{GActions_1}} \times \cdots \times 2^{\overline{GActions_n}} \;\to\; (\mathscr{G} \;\to\; \mathscr{G})$$

Thus $\tau^B_{P_\epsilon,P}$ is only one possible transition function, the *Byzantine* transition function, obtained from protocols P_ϵ and P: namely, the one resulting from using the Byzantine filter functions $filter^B_i$ and $filter^B_\epsilon$.

Whichever filtering is used, one round of transition consists of the following phases:

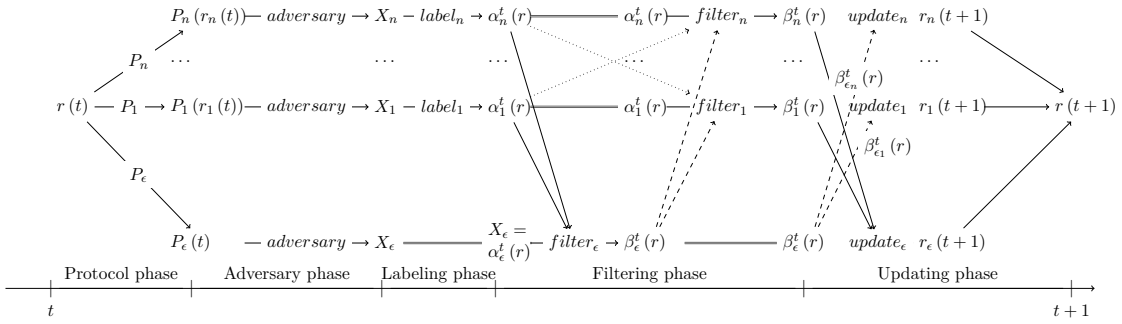

Figure 2.1: The evolution of states in round $t.5$ (from timestamp $t \in \mathbb{N}$ to $t+1$) inside a run r constructed according to the transition function $\tau_{P_\epsilon,P}$. Different communication models require changes to the filtering functions $filter_\epsilon$ and $filter_i$.

One step of $\tau_{P_\epsilon,P}$-transition for runs One transition made according to a transition function $\tau_{P_\epsilon,P}$ consists of five consecutive phases, which are visually represented in Fig. 2.1:

1. *Protocol phase* (protocols are explicit arguments to the transition template τ)
   First, the protocol P_i for each agent i lays out a range $P_i(r_i(t))$ of possible sets of i's actions in the round based on the local state $r_i(t)$ of the agent. Similarly, the protocol P_ϵ of the environment lays out a range $P_\epsilon(t)$ of possible t-coherent sets of events in the round based on time t.

2. *Adversary phase* (this phase is stable: it does not change from template to template or from protocol to protocol)
   From these ranges, the adversary non-deterministically picks one set
   $$X_i \in P_i(r_i(t)) \tag{2.48}$$
   of actions for each agent i and a set
   $$X_\epsilon \in P_\epsilon(t) \tag{2.49}$$
   of events for the environment. These are actions the agents intend to perform and events the environment intends to impose in the round. Note that $X_i \subset \overline{Actions_i}$ and $X_\epsilon \subset GEvents$.

3. *Labeling phase* (this phase is stable: it does not change from template to template or from protocol to protocol)
   The environment processes the intended actions X_i of each agent i converting them into the global format, in particular, assigning GMIs to message send requests from agents. We denote the resulting sets
   $$\alpha_i^t(r) := label_i(X_i, t). \tag{2.50}$$
   The set of environmental events X_ϵ is already in the global format and requires no modifications:
   $$\alpha_\epsilon^t(r) := X_\epsilon. \tag{2.51}$$
   Note that $\alpha_i^t(r) \subset \overline{GActions_i}$ and $\alpha_\epsilon^t(r) \subset GEvents$.

4. *Filtering phase* (this phase depends on the filtering functions $filter_\epsilon$ and $filter_i$, which are considered to be part of the template)
In this phase, intended actions and events that are deemed "causally impossible" in the underlying communication model are filtered out: though they may be requested by the agents/environment, they are not performed and not recorded in histories. Thus, the exact nature of filtering depends on the intended model, and different filtering functions produce different transition functions. The following requirements are imposed on filtering functions that can be used in any template:

$$passive(i, X_\epsilon) \implies filter_i(X_1, \ldots, X_n, X_\epsilon) = \varnothing \tag{2.52}$$
$$filter_i(X_1, \ldots, X_n, X_\epsilon) \subset X_i \tag{2.53}$$
$$filter_\epsilon(h, X_\epsilon, X_1, \ldots, X_n) \subset X_\epsilon \tag{2.54}$$

In other words, no actions by agent i are allowed by the environment unless $go(i)$ is issued and filtering is a non-increasing function with respect to the relevant argument. Note that some environment's events may also be "causally impossible", such as, e.g., receiving a message that was never sent. The filtering phase is further divided into two subphases:

a) first impossible environment events are filtered out by the function $filter_\epsilon$ based on the intended environment's events X_ϵ and intended actions $\alpha_i^t(r)$ of all agents, resulting in the set $\beta_\epsilon^t(r)$ of performed environment's events:

$$\beta_\epsilon^t(r) := filter_\epsilon\left(r(t), \quad \alpha_\epsilon^t(r), \quad \alpha_1^t(r), \quad \ldots, \quad \alpha_n^t(r)\right), \tag{2.55}$$

b) then for each agent i, the filtering function $filter_i$ performs the same task on the agents' actions, but taking into account the already filtered events $\beta_\epsilon^t(r)$ and intended actions $\alpha_j^t(r)$ of all agents $j \in \mathcal{A}$. The resulting sets of actions actually performed by agents are denoted $\beta_i^t(r)$:

$$\beta_i^t(r) := filter_i\left(\alpha_1^t(r), \quad \ldots, \quad \alpha_n^t(r), \quad \beta_\epsilon^t(r)\right) \tag{2.56}$$

Note that $\beta_i^t(r) \subset \alpha_i^t(r) \subset \overline{GActions_i}$ and $\beta_\epsilon^t(r) \subset \alpha_\epsilon^t(r) \subset GEvents$ by (2.53) and (2.54) and that the latter is always a t-coherent set.

5. *Updating phase* (this phase is stable: it does not change from template to template or from protocol to protocol)
The events $\beta_\epsilon^t(r)$ and actions $\beta_i^t(r)$ actually happening in the round are faithfully recorded into the global history and are translated into the simplified local form for being recorded into the local histories of each agent by the update functions. The crucial point of this translation is stripping out the GMIs and any information that would allow an agent to easily distinguish a correct event from a faulty one. Once again, the local history of each agent i is only affected by the actions $\beta_i^t(r)$ it performs and environment's events $\beta_{\epsilon_i}^t(r)$ it observes, whereas the global history is modified based on the complete information about all events and actions performed in the round.

$$r_i(t+1) := update_i\left(r_i(t), \quad \beta_i^t(r), \quad \beta_\epsilon^t(r)\right) \tag{2.57}$$
$$\beta^t(r) := \left(\beta_\epsilon^t(r), \quad \beta_1^t(r), \quad \ldots, \quad \beta_n^t(r)\right), \tag{2.58}$$
$$r_\epsilon(t+1) := update_\epsilon\left(r_\epsilon(t), \quad \beta^t(r)\right) \tag{2.59}$$

We will routinely use (2.48)–(2.59) to prove properties of transitional runs. However, it should be noted that in this respect β sets have a different status from α sets and X sets. Indeed, by (2.59), all β sets can be easily retrieved from a given transitional run. We do not even have to assume the transitionality of a run to define the β sets, though this definition would make sense mostly for transitional runs. In compliance with (2.40), for an arbitrary global history h, we define

$$\beta_i(h) := \pi_1 h_\epsilon \cap \overline{GActions}_i \tag{2.60}$$
$$\beta_\epsilon(h) := \pi_1 h_\epsilon \cap GEvents \tag{2.61}$$

For the case of runs

$$\beta_i^t(r) = \beta_i(r(t+1)) = \pi_1 r_\epsilon(t+1) \cap \overline{GActions}_i \tag{2.62}$$
$$\beta_\epsilon^t(r) = \beta_\epsilon(r(t+1)) = \pi_1 r_\epsilon(t+1) \cap GEvents \tag{2.63}$$

The latter set we further partition (we only show the notation for the case of runs, the case of histories is processed analogously):

- correct external events: $\overline{\beta}_\epsilon^t(r)$ observed by all agents and $\overline{\beta}_{\epsilon_i}^t(r)$ observed specifically by agent i;
- system events (*go*, *sleep*, and *hibernate*): $\beta_g^t(r)$ imposed on all agents and $\beta_{g_i}^t(r)$ imposed specifically on agent i;
- Byzantine events: $\beta_b^t(r)$ observed by all agents and $\beta_{b_i}^t(r)$ observed specifically by agent i;
- all external events that make the agent Byzantine until the end of the run, including faultily skipped rounds $sleep(i)$ and $hibernate(i)$: $\beta_f^t(r)$ imposed on all agents and $\beta_{f_i}^t(r)$ imposed specifically on agent i.

On the other hand, parts of the X and α sets are filtered out and have no effect on the transitions in the run. Hence, given a transitional run, it is not generally possible to retrieve the exact X and α sets used in each transition. All that is required is that there exist a collection of sets $X_1, \ldots, X_n, X_\epsilon$ satisfying (2.48)–(2.49) that eventually generate $\beta_1^t(r), \ldots, \beta_n^t(r), \beta_\epsilon^t(r)$ from (2.62)–(2.63) according to (2.50)–(2.56). Despite this subtlety, we still use function-like notation for α sets to keep the notation uniform with β sets. In particular, we use this uniformity to identify the same parts of the α sets by the same subscripts, e.g., $\overline{\alpha}_{\epsilon_i}^t(r)$ represents the set of correct internal events the environment is intending to impose on agent i, which will be filtered and become $\overline{\beta}_{\epsilon_i}^t(r)$, the set of correct external events actually imposed by the environment on agent i during round $t.5$ of the run r.

If we write $\beta^t(r), \beta_\epsilon^t(r), \beta_1^t(r), \ldots, \beta_n^t(r)$, etc., it means that we assume the run r to be transitional and use $\alpha_\epsilon^t(r), \alpha_1^t(r), \ldots, \alpha_n^t(r)$, etc. for one possible choice of sets that could lead to such sets β according to a transition function $\tau_{P_\epsilon, P}$.

Transitional runs have several useful properties:

Remark 2.3.12 (Global total recall)**.** Not only $\beta^t(r)$ but also all $\beta^{t'}(r)$ for $t' \leq t$ can be extracted from $r(t+1)$, or even from $r_\epsilon(t+1)$ in a $\tau_{P_\epsilon,P}$-transitional run r: for instance,

$$\beta_\epsilon^{t'}(r) = \pi_1 r_\epsilon(t'+1) \cap GEvents = \pi_{t-t'+1} r_\epsilon(t+1) \cap GEvents$$

Lemma 2.3.13 (Objective global time)**.** *For any $\tau_{P_\epsilon,P}$-transitional run r and any timestamp t,*

$$|r(t)| = t.$$

The following properties rely on the restrictions imposed on the environment's protocol as well as on the filtering functions implementing general Byzantine agents. In order to make this and further results more general we define pointwise order on filtering functions and formulate many of the statements for any filter up to the general Byzantine one.

Definition 2.3.14. We say that a filter $filter_\epsilon^1$ is **stricter** than a filter $filter_\epsilon^2$ or that $filter_\epsilon^2$ is more **liberal** than $filter_\epsilon^1$ and we write $filter_\epsilon^1 \subset filter_\epsilon^2$ if the inclusion holds pointwise

$$filter_\epsilon^1(h, X_\epsilon, X_1, \ldots, X_n) \subset filter_\epsilon^2(h, X_\epsilon, X_1, \ldots, X_n)$$

for any global history $h \in \mathscr{G}$, any $X_\epsilon \subset GEvents$, and arbitrary $X_i \subset \overline{GActions}_i$ for each $i \in \mathcal{A}$.

Lemma 2.3.15 (GMIs are correct)**.** *For any $\tau_{P_\epsilon,P}$-transitional run r for some $P_\epsilon \in \mathscr{C}_\epsilon$, for $i, j \in \mathcal{A}$, $\mu \in Msgs$, $t \in \mathbb{N}$, $A \in \overline{GActions}_i \sqcup \{\text{📅}\}$, and $id \in \mathbb{N}$*

$$gsend(i,j,\mu,id) \in \beta_i^t(r) \implies id = id(i,j,\mu,k,t) \text{ for some } k \in \mathbb{N} \tag{2.64}$$

$$fake(i, gsend(i,j,\mu,id) \mapsto A) \in \beta_{b_i}^t(r) \implies id = id(i,j,\mu,k,t) \text{ for some } k \in \mathbb{N} \tag{2.65}$$

In addition, for any transition template τ with a $filter_\epsilon \subset filter_\epsilon^B$ and for any $\tau_{P_\epsilon,P}$-transitional run r

$$grecv(i,j,\mu,id) \in \overline{\beta}_{\epsilon_i}^t(r) \implies id = id(j,i,\mu,k,t') \text{ for some } k \in \mathbb{N} \text{ and } t' \leq t \tag{2.66}$$

Proof. By (2.56), (2.53), (2.50), and Def. 2.2.12, the id for correct *gsend* instructions is supplied by the function $label_i$, which guarantees the first statement.

Similarly, for the second statement, by (2.55), (2.51), Def. 2.2.4(3), the id's for Byzantine *gsend* instructions are created by the environment in a manner that guarantees the second statement.

Finally, for the third statement, by the same (2.55) and (2.24), if a *grecv* command was not filtered out by $filter_\epsilon$, it was not filtered by the more liberal $filter_\epsilon^B$, hence, the matching *gsend* command or its Byzantine version must have occurred at the latest by the same round. In other words, taking into account (2.55)–(2.56), there are four possibilities:

1. $gsend(j,i,\mu,id) \in r_\epsilon(t) \implies gsend(j,i,\mu,id) \in \beta_j^{t'}(r)$ for some $t' < t$
2. $gsend(j,i,\mu,id) \in \beta_j^t(r)$

3. $fake\,(j, gsend(j,i,\mu,id) \mapsto A) \in r_\epsilon\,(t)$ for some $A \in \{\text{📅}\} \sqcup \overline{GActions}_j \implies$
$fake\,(j, gsend(j,i,\mu,id) \mapsto A) \in \beta^{t'}_{b_j}\,(r)$ for some $t' < t$ and $A \in \{\text{📅}\} \sqcup \overline{GActions}_j$

4. $fake\,(j, gsend(j,i,\mu,id) \mapsto A) \in \beta^{t}_{b_j}\,(r)$ for some $A \in \{\text{📅}\} \sqcup \overline{GActions}_j$

In other words,

$$gsend(j,i,\mu,id) \in \beta^{t'}_j\,(r) \text{ or } fake\,(j, gsend(j,i,\mu,id) \mapsto A) \in \beta^{t'}_{b_j}\,(r) \text{ for an } A \in \{\text{📅}\} \sqcup \overline{GActions}_j$$

for some $t' \leq t$. It remains to use the already proved (2.64) or (2.65) respectively. □

Corollary 2.3.16. *For any $\tau_{P_\epsilon,P}$-transitional run r with a $filter_\epsilon \subset filter^B_\epsilon$, for $i,j \in \mathcal{A}$, $\mu \in Msgs$, $t \in \mathbb{N}$, and $id \in \mathbb{N}$*

$$grecv(i,j,\mu,id) \in \overline{\beta}^t_{\epsilon_i}\,(r) \implies (\exists t' \leq t)\Big(gsend(j,i,\mu,id) \in \beta^{t'}_j\,(r) \quad or \quad (\exists A \in \{\text{📅}\} \sqcup \overline{GActions}_j)\, fake\,(j, gsend(j,i,\mu,id) \mapsto A) \in \beta^{t'}_{b_j}\,(r)\Big) \tag{2.67}$$

Corollary 2.3.17 (GMIs are unique)**.** *For any transitional run r, for $i,j,k,l \in \mathcal{A}$, $\mu,\mu' \in Msgs$, $t,t' \in \mathbb{N}$, $id \in \mathbb{N}$, $A \in \{\text{📅}\} \sqcup \overline{GActions}_i$, and $A' \in \{\text{📅}\} \sqcup \overline{GActions}_k$:*

$$\begin{cases} gsend(i,j,\mu,id) \in \beta^t_i\,(r) \\ gsend(k,l,\mu',id) \in \beta^{t'}_k\,(r) \end{cases} \implies \begin{cases} k = i \\ l = j \\ t' = t \\ \mu' = \mu \end{cases} \tag{2.68}$$

$$\begin{cases} fake\,(i, gsend(i,j,\mu,id) \mapsto A) \in \beta^t_{b_i}\,(r) \\ fake\,(k, gsend(k,l,\mu',id) \mapsto A') \in \beta^{t'}_{b_k}\,(r) \end{cases} \implies \begin{cases} k = i \\ l = j \\ t' = t \\ \mu' = \mu \end{cases} \tag{2.69}$$

$$\begin{cases} gsend(i,j,\mu,id) \in \beta^t_i\,(r) \\ fake\,(k, gsend(k,l,\mu',id) \mapsto A') \in \beta^{t'}_{b_k}\,(r) \end{cases} \implies \begin{cases} k = i \\ l = j \\ t' = t \\ \mu' = \mu \end{cases} \tag{2.70}$$

In other words, the GMI id completely determines the sender, the recipient, the sent message and the time of sending for both correct and Byzantine messages processed by the environment.

Proof. The statements follow from Lemma 2.3.15 (from (2.64) for the first statement, from (2.65) for the second one, and from both (2.64) and (2.65) for the third one) and the injectivity of $id(\cdot)$ from Def. 2.1.13. □

Corollary 2.3.18 (Send–receive causality)**.** *For any $\tau_{P_\epsilon,P}$-transitional run r with a $filter_\epsilon \subset filter^B_\epsilon$, for $i,j,k,l \in \mathcal{A}$, $\mu,\mu' \in Msgs$, $t,t' \in \mathbb{N}$, $A \in \{\text{📅}\} \sqcup \overline{GActions_i}$, and $id \in \mathbb{N}$:*

$$\begin{cases} gsend(i,j,\mu,id) \in \beta^t_i(r) \\ grecv(k,l,\mu',id) \in \overline{\beta}^{t'}_{\epsilon_k}(r) \end{cases} \implies \begin{cases} k = j \\ l = i \\ t' \geq t \\ \mu' = \mu \end{cases} \tag{2.71}$$

$$\begin{cases} fake(i, gsend(i,j,\mu,id) \mapsto A) \in \beta^t_{b_i}(r) \\ grecv(k,l,\mu',id) \in \overline{\beta}^{t'}_{\epsilon_k}(r) \end{cases} \implies \begin{cases} k = j \\ l = i \\ t' \geq t \\ \mu' = \mu \end{cases} \tag{2.72}$$

In other words, whether a message is sent correctly or faultily, the receipt of the message cannot happen before it was sent and the senders/recipients/content at the time of receipt must match those at the time of sending.

Proof. The statements follow from Lemma 2.3.15 and the injectivity of $id(\cdot)$ from Def. 2.1.13. □

Remark 2.3.19. While GMIs for sent messages are unforgeable for all transition templates, the correctness of GMIs for correctly received messages relies on the filtering performed by the general Byzantine environment filter and any stricter filters. It could be argued that Byzantine behavior can be strengthened and/or reliability of the communication channel can be weakened to enable Byzantine agents to forge GMIs, but this is outside the scope of this report, especially given that the man in the middle attack can be represented without forged GMIs (see Remark 2.1.5 for details).

In order to discuss what it means to behave the same way at a node $(i,t) \in \mathcal{A} \times \mathbb{N}$ in two distinct runs r and r' we define the notion of *agreement*:

Definition 2.3.20 (Agreement on a node)**.** For two runs r and r' from R, we say that r and r' **agree on** a node $(i,t) \in \mathcal{A} \times \mathbb{N}$ iff

1. $r_i(t) = r'_i(t)$
2. $\beta^t_{\epsilon_i}(r) = \beta^t_{\epsilon_i}(r')$
3. $\beta^t_i(r) = \beta^t_i(r')$

We extend this notion to sets of nodes $X \subset \mathcal{A} \times \mathbb{N}$: runs r and r' agree on X iff

$$(\forall (i,t) \in X) \;\; r \text{ and } r' \text{ agree on } (i,t)$$

Remark 2.3.21. In the above definition, Requirement 1 states that the local states of i at t are identical in both runs. Requirement 3 expresses that actions of i chosen by the adversary based on the protocol for the upcoming round $t.5$ are identical in both runs. Requirement 2 ensures three properties: correct external events imposed on i in round $t.5$ are identical, there is no difference as to whether i is awoken for round $t.5$ or not, faulty behavior of i in round $t.5$ is identical.

Lemma 2.3.22. *For two transitional runs r and r' and a node $(i,t) \in \mathcal{A} \times \mathbb{N}$*

$$r \text{ and } r' \text{ agree on } (i,t) \qquad \text{implies} \qquad r_i(t+1) = r'_i(t+1)$$

Proof. By (2.57) and (2.39). □

Remark 2.3.23. For two transitional runs r and r' and a node $(i,t) \in \mathcal{A} \times \mathbb{N}$,

$$r \text{ and } r' \text{ agreeing on } (i,t) \quad \text{is strictly stronger than} \quad \begin{cases} r_i(t) = r'_i(t) \\ r_i(t+1) = r'_i(t+1) \end{cases}$$

The most important case when local states remain in sync between timestamps t and $t+1$ despite different things happening during round $t.5$ is when a correct action/event in run r is replaced with its Byzantine version in run r'.

While it is preferrable to directly build desired properties of runs into the transition functions, in a manner of speech, to hardwire them, there are characteristics that cannot be implemented on a round-by-round basis. The most familiar of them is the *liveness condition* that requires that certain things happen *eventually* in a run. If no bound on the delay is given, this requirement cannot be translated into local terms because this is the property of the whole infinite run. Therefore, to enforce such properties we have to restrict the set of runs being considered.

Definition 2.3.24 (Admissibility condition)**.** An **admissibility condition** Ψ is any subset of the set R of all runs.

Now we have all the ingredients to define sets of runs for particular communication models. A **context** is essentially an extended environment where a **joint protocol** is executed.

Definition 2.3.25 (Context)**.** A **context**

$$\gamma = (P_\epsilon, \mathscr{G}(0), \tau, \Psi) \tag{2.73}$$

consists of an environment protocol $P_\epsilon \in \mathscr{C}_\epsilon$, a set of global initial states $\mathscr{G}(0)$, a transition template τ, and an admissibility condition Ψ.

Definition 2.3.26 (Agent-context)**.** Given a context γ and joint protocol P, we can combine them in an **agent-context** $\chi = (\gamma, P)$.

Definition 2.3.27 (Consistency)**.** For a context $\gamma = (P_\epsilon, \mathscr{G}(0), \tau, \Psi)$ and a joint protocol P, we define the set of runs **weakly consistent** with P in γ (or weakly consistent with $\chi = (\gamma, P)$), denoted $R^{w\chi} = R^{w(\gamma,P)}$, to be the set of $\tau_{P_\epsilon,P}$-transitional runs that start at some global initial state from $\mathscr{G}(0)$:

$$R^{w(\gamma,P)} := \{ r \in R \mid r(0) \in \mathscr{G}(0) \text{ and } (\forall t \in \mathbb{N})\, r(t+1) \in \tau_{P_\epsilon,P}(r(t)) \} \tag{2.74}$$

A run r is called **strongly consistent**, or simply **consistent**, with P in γ (or with χ) if it is weakly consistent with P in γ and, additionally, satisfies the admissibility condition: $r \in \Psi$. We denote the system of all runs consistent with P in γ by

$$R^{(\gamma,P)} := R^{w(\gamma,P)} \cap \Psi. \tag{2.75}$$

We say that an agent-context $\chi = (\gamma, P)$ is **non-excluding** if any prefix of a run weakly consistent with P in γ can be extended to a run strongly consistent with P in γ.

Definition 2.3.28 (Non-excluding agent-context)**.** For an agent-context χ, χ is non-excluding iff

$$R^{\chi} \neq \varnothing \quad \text{and} \quad (\forall r \in R^{w\chi})(\forall t \in \mathbb{N})(\exists r' \in R^{\chi})(\forall t' \leq t)\, r'(t') = r(t')$$

The full formalism will be introduced in Sect. 2.4.

A local state of a run is called *coherent* if the agent could have arrived at the same local state without exhibiting any Byzantine behavior.

Definition 2.3.29 (Coherence)**.** For an agent-context χ, a run $r \in R^{\chi}$, and a node $(i,t) \in \mathcal{A} \times \mathbb{N}$, we say the local state $r_i(t)$ is χ-**coherent with respect to** i, or simply **coherent with respect to** i, iff

$$(\exists r' \in R^{\chi})(\exists t' \in \mathbb{N})\Big(r'_i(t') = r_i(t) \text{ and } i \notin \mathcal{A}\left(Failed\left(r', t'\right)\right)\Big).$$

Definition 2.3.30 (Failure free)**.** For an agent-context χ, a run $r \in R^{\chi}$, and a node $(i,t) \in \mathcal{A} \times \mathbb{N}$, we say the local state $r_i(t)$ is χ-**failure free with respect to** i, or simply **failure free with respect to** i, iff

$$(\exists r' \in R^{\chi})(\exists t' \in \mathbb{N})\Big(r'_i(t') = r_i(t) \text{ and } \mathcal{A}\left(Failed\left(r', t'\right)\right) = \varnothing\Big).$$

2.4 Syntax and Semantics

We define a formal language and its semantics in order to express knowledge of an agent in distributed systems. We will be using the standard adaptation of Kripke models to the run-based environment. Kripke models are based on abstract worlds or states supplied with the indistinguishability relations for the agents. For a collection of runs, it is quite natural to consider states to be various global states achievable during these runs and define the indistinguishability relation for an agent based on its knowledge of the local state: global states are indistinguishable for agent $i \in \mathcal{A}$ if and only if i's local state in these states is the same (i.e., the states are exactly the same from the point of view of agent i).

For multiple reasons, we consider the general set up in the form of agent-context to be common knowledge among agents. In other words, for an agent-context χ, the only possibilities agents consider are global states from various runs from R^{χ}. For instance, a synchronous agent who determined that it had skipped a round should conclude that it is compromised rather than imagining itself in an asynchronous context. By the same token, the same local state should give rise to different epistemic states depending on the type of distributed system. This is the reason why the Consensus problem with Byzantine failures *can* be solved in the synchronous context but *not* in the asynchronous one.

Definition 2.4.1 (Atomic propositions)**.** We consider an infinitely countable set Π of **atomic propositions**.

Definition 2.4.2 (Interpretation function)**.** An **interpretation function** $\pi\colon \mathscr{G} \to \{\bot, \top\}^{\Pi}$ assigns, for a given global state $h \in \mathscr{G}$, a propositional valuation function $\pi(h)\colon \Pi \to \{\bot, \top\}$.

Hence, for a global state $h \in \mathscr{G}$ the truth value $\pi(h)(p)$ of an atomic proposition $p \in \Pi$ is either $\bot$ (*false*) or $\top$ (*true*).

Definition 2.4.3 (Interpreted system)**.** A set $R' \subset R$ of runs and an interpretation function π yield an **interpreted system** $I = (R', \pi)$. For an agent-context $\chi = (\gamma, P)$, an interpreted system (R', π) is called **weakly χ-based** if $R' = R^{w(\chi)}$ and **χ-based** if $R' = R^{\chi}$.

Definition 2.4.4 (Indistinguishability relation)**.** For agent $i \in \mathcal{A} = [\![1; n]\!]$, the **indistinguishability relation** $\sim_i \subset \mathscr{G}^2$ is formally defined as follows:

$$\sim_i := \{(h, h') \mid \pi_{i+1} h = \pi_{i+1} h'\} \tag{2.76}$$

In other words, agent i cannot distinguish between global histories $h = (h_\epsilon, h_1, \ldots, h_n)$ and $h' = (h'_\epsilon, h'_1, \ldots, h'_n)$ iff $h_i = h'_i$, i.e., i sees exactly the same local history at h and h'.

Remark 2.4.5. Generally, for a particular R', the interpretation functions π and the indistinguishability relation $\sim_i$ are also defined for global states only appearing in the runs from $R \setminus R'$. This creates no problems but makes the formalism simpler.

We define a **language** $\mathfrak{L}$ to deal with the expression of knowledge in a *system*. For this we extend the propositional logic with the following operators:

1. three modal operators:
   - K_i for each agent $i \in \mathcal{A}$. For a global state $h = (h_\epsilon, h_1, \ldots, h_n) \in \mathscr{G}$, the formula $K_i\varphi$ can be read as "agent i knows φ (based on its local state h_i)": this means that, in every global state indistinguishable from h for i, the proposition φ holds;
   - E_G for each group $G \subset \mathcal{A}$ of agents. It means that "everyone in the group G of agents knows φ (based on their respective local states)." E_G is naturally defined to be the conjunction of all operators K_i over $i \in G$. We generally assume $G \neq \varnothing$ unless stated otherwise;
   - C_G for each group $G \subset \mathcal{A}$ of agents. It means that "φ is common knowledge among the agents of G," i.e., everyone in G knows that everyone in G knows ... that everyone in G knows φ. We generally assume $G \neq \varnothing$ unless stated otherwise;
2. one temporal operator:
   - $\Box$ is the "always in the future" operator. It expresses statements like "the sender will never forget that he has sent *Hello*."[12]

Definition 2.4.6. For an agent $i \in \mathcal{A}$, a group of agents $\varnothing \neq G \subset \mathcal{A}$ and an atomic proposition $p \in \Pi$, the **language** $\mathfrak{L}$ is generated by the following BNF specification

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_i\varphi \mid C_G\varphi \mid \Box\varphi \mid Y\varphi$$

[12] In temporal logic, this operator is usually denoted G.

We define the remaining Boolean connectives such as $\vee$, $\rightarrow$, and $\leftrightarrow$ in the standard way.[13] Mutual knowledge E_G and iterated mutual knowledge E_G^m are defined by

$$E_G\varphi := \bigwedge_{i\in G} K_i\varphi \qquad E_G^0\varphi := \varphi \qquad E_G^{n+1}\varphi := E_G E_G^n\varphi$$

In addition, each modal operator $\heartsuit$ has its dual $\neg\heartsuit\neg$. The duals of epistemic operators are denoted by putting $\hat{}$ over the operator, e.g., $\hat{K}_i\varphi = \neg K_i\neg\varphi$. The dual of $\Box$ is traditionally denoted by $\Diamond$.[14] Note that $E_G^1\varphi = E_G\varphi$ is syntactically the same formula.

To simplify the definition of truth, we define the following binary relations on the set of global states, using the standard notion of relation composition for binary relations $\star$ and $*$:

$$\star \circ * := \{(x,z) \mid (\exists y)(x \star y \wedge y * z)\}$$

Definition 2.4.7. Other binary relations on $\mathscr{G}$ are defined as follows:

$$\sim_G := \bigcup_{i\in G} \sim_i, \qquad \sim_G^0 := \{(h,h) \mid h \in \mathscr{G}\},$$

$$\sim_G^m := \underbrace{\sim_G \circ \cdots \circ \sim_G}_{m} \text{ (for } m > 1), \qquad \sim_G^C := \bigcup_{m=1}^{\infty} \sim_G^m$$

With the language $\mathfrak{L}$ we can express statements about the knowledge of an agent (or of a group of agents) or about the temporal properties of a formula. The semantics with respect to interpreted systems is as follows:

Definition 2.4.8. For an interpreted system $I = (R', \pi)$ with the set $R' \subset R$ of runs and the interpretation function π, for an agent $i \in \mathcal{A}$, a group of agents $\varnothing \neq G \subset \mathcal{A}$, a run $r \in R'$, and a timestamp $t \in \mathbb{N}$:

$$\begin{aligned}
(I,r,t) &\models p && \text{iff} && \pi(r(t))(p) = \top \\
(I,r,t) &\models \neg\varphi && \text{iff} && (I,r,t) \not\models \varphi \\
(I,r,t) &\models \varphi \wedge \varphi' && \text{iff} && (I,r,t) \models \varphi \text{ and } (I,r,t) \models \varphi' \\
(I,r,t) &\models K_i\varphi && \text{iff} && (\forall r' \in R')(\forall t' \in \mathbb{N})\Big(r'(t') \sim_i r(t) \;\Rightarrow\; (I,r',t') \models \varphi\Big) \\
(I,r,t) &\models C_G\varphi && \text{iff} && (\forall r' \in R')(\forall t' \in \mathbb{N})\Big(r'(t') \sim_G^C r(t) \;\Rightarrow\; (I,r',t') \models \varphi\Big) \\
(I,r,t) &\models \Box\varphi && \text{iff} && (\forall t' \geq t)\,(I,r,t') \models \varphi \\
(I,r,t) &\models Y\varphi && \text{iff} && (t > 0) \text{ and } (I,r,t-1) \models \varphi
\end{aligned}$$

Note that the "yesterday" modality satisfies $(I,r,0) \not\models Y\varphi$ for any φ.

[13] We also use the common ranking of binding strength: $\neg$ and Y are the strongest, then $\vee$ and $\wedge$ which bind equally strong, then $\rightarrow$, and the weakest is $\leftrightarrow$.

[14] In temporal logic, it is usually denoted F.

It immediately follows from the definition, based on the meaning of the secondary connectives, that

$$
\begin{aligned}
(I,r,t) &\models \varphi \vee \varphi' && \text{iff} && (I,r,t) \models \varphi \text{ or } (I,r,t) \models \varphi' \\
(I,r,t) &\models \varphi \rightarrow \varphi' && \text{iff} && (I,r,t) \not\models \varphi \text{ or } (I,r,t) \models \varphi' \\
(I,r,t) &\models E_G\varphi && \text{iff} && (\forall r' \in R')(\forall t' \in \mathbb{N})\Big(r'(t') \sim_G r(t) \;\Rightarrow\; (I,r',t') \models \varphi\Big) \\
& && \text{iff} && (\forall r' \in R')(\forall t' \in \mathbb{N})(\forall i \in G)\Big(r'(t') \sim_i r(t) \;\Rightarrow\; (I,r',t') \models \varphi\Big) \\
(I,r,t) &\models E^0_G\varphi && \text{iff} && (I,r,t) \models \varphi \\
(I,r,t) &\models E^m_G\varphi && \text{iff} && (\forall r' \in R')(\forall t' \in \mathbb{N})\Big(r'(t') \sim^m_G r(t) \;\Rightarrow\; (I,r',t') \models \varphi\Big) \\
& && \text{iff} && (\forall r^0,\ldots,r^m \in R')(\forall t_0,\ldots,t_m \in \mathbb{N})\Big(r^0 = r \wedge t_0 = t \wedge \\
& && && \qquad (\forall k < m)(\exists i_k \in G) r^k(t_k) \sim_{i_k} r^{k+1}(t_{k+1}) \quad\Rightarrow\quad (I,r^m,t_m) \models \varphi\Big) \\
(I,r,t) &\models C_G\varphi && \text{iff} && (\forall m \in \mathbb{N}\setminus\{0\})(I,r,t) \models E^m_G\varphi \\
& && \text{iff} && (\forall m \in \mathbb{N}\setminus\{0\})(\forall r^0,\ldots,r^m \in R')(\forall t_0,\ldots,t_m \in \mathbb{N})\Big(r^0 = r \wedge t_0 = t \wedge \\
& && && \qquad (\forall k < m)(\exists i_k \in G) r^k(t_k) \sim_{i_k} r^{k+1}(t_{k+1}) \quad\Rightarrow\quad (I,r^m,t_m) \models \varphi\Big) \\
(I,r,t) &\models \Diamond\varphi && \text{iff} && (\exists t' \geq t)\,(I,r,t') \models \varphi
\end{aligned}
$$

It is also easy to see that truth is defined with respect to global histories rather than points in a run, i.e., for any formula φ, we have

$$r(t) = r'(t) \quad\Rightarrow\quad (\forall \varphi \in \mathfrak{L})\Big((I,r,t) \models \varphi \quad\Leftrightarrow\quad (I,r',t') \models \varphi\Big)$$

even though $r(t+1)$ may differ from $r'(t+1)$. (Note that $r(t) \neq r'(t')$ for any $t \neq t'$ because the length of the environment's history is a function of time.)

2.5 Atomic Propositions

We have defined the language $\mathfrak{L}$ as the syntax and the associated semantics to tell the truth value of a formula for a given interpreted system I, run $r \in R$ and timestamp $t' \in \mathbb{N}$. We now designate some of the atomic propositions from Π as special and consider their truth values to be fully determined by $r(t')$ rather than arbitrary. In other words, we will restrict interpretations π so as to adhere to the following intended meanings for a given $r(t')$ with $t \leq t'$ and $i \in \mathcal{A}$. We also introduce useful abbreviations for negations of some atomic propositions.

- $\mathit{correct}_{(i,t)}$ states that by timestamp $t \in \mathbb{N}$, i.e., in rounds $0.5, 1.5, \ldots, (t-1).5$, agent $i \in \mathcal{A}$ did not violate its protocol through improper action or improper inaction, i.e., did not exhibit any Byzantine actions or events and was not marked with $\mathit{sleep}\,(i)$ or $\mathit{hibernate}\,(i)$ from timestamp 0 to timestamp t.
- $\mathit{correct}_i$ states that by the time of evaluation ($t' \in \mathbb{N}$, i.e., in rounds $0.5, 1.5, \ldots, (t'-1).5$) agent $i \in \mathcal{A}$ did not violate its protocol through improper action or improper inaction.

- $faulty_{(i,t)} := \neg correct_{(i,t)}$ states that by timestamp $t \in \mathbb{N}$ agent $i \in \mathcal{A}$ violated its protocol through improper action or improper inaction.

- $faulty_i := \neg correct_i$ states that by the time of evaluation agent $i \in \mathcal{A}$ violated its protocol through improper action or improper inaction.

- $fake_{(i,t)}(o)$ states that agent $i \in \mathcal{A}$ thinks that $o \in \overline{Actions} \sqcup \overline{Events}$ occurred in round $(t-1).5$ but thinks so for a wrong reason, i.e., $o \in \sigma\left(\beta_{b_i}^{t-1}(r)\right)$. Note that in the Byzantine setting, if o is an action, it is still possible that agent i did perform o in round $(t-1).5$ but mistook it for another action o'', while at the same time mistaking some third action o' for o, e.g., when $fake(i, O' \mapsto O), fake(i, O \mapsto O'') \in \beta_{b_i}^{t-1}(r)$. It is also possible that action o was performed according to the protocol, e.g., when $fake(i, O' \mapsto O) \in \beta_{b_i}^{t-1}(r)$ and $O \in \beta_i^{t-1}(r)$.

- $\overline{occurred}_{(i,t)}(o)$ states that agent i thinks that $o \in \overline{Actions} \sqcup \overline{Events}$ occurred in round $(t-1).5$ for a right reason, i.e., $o \in label^{-1}\left(\beta_i^{t-1}(r) \sqcup \overline{\beta}_{\epsilon_i}^{t-1}(r)\right)$. Note that in the Byzantine setting, if o is an action, it is possible that agent i has both a right and a wrong reason to believe it performed o in round $(t-1).5$, e.g., when both $O \in \beta_i^{t-1}(r)$ and $fake(i, O' \mapsto O) \in \beta_{b_i}^{t-1}(r)$. In this case, agent i unwittingly piggybacks O' onto the action O. For instance, one might accidentally throw away a postcard with an unwanted catalog, or a ticket-vending machine might accidentally print two tickets instead of one.

- $\overline{occurred}_i(o)$ states that by the time t' of evaluation, agent i correctly registered $o \in \overline{Actions} \sqcup \overline{Events}$ occurring in some previous round, i.e., $(\exists t < t')\, o \in label^{-1}\left(\beta_i^t(r) \sqcup \overline{\beta}_{\epsilon_i}^t(r)\right)$.

- $occurred_i(o)$ states that by the time of evaluation agent i believes that $o \in \overline{Actions} \sqcup \overline{Events}$ occurred.

We now give formal definitions and discuss properties of these atomic propositions.

Definition 2.5.1. A (weakly) χ-based interpreted system $I = (R', \pi)$ and its interpretation π are called **proper** if, for any run $r \in R'$, any agent $i \in \mathcal{A}$, arbitrary two timestamps $t \leq t'$, and any $o \in \overline{Actions} \sqcup \overline{Events}$, the interpretation π satisfies the following properties:

$$\pi\left(r\left(t'\right)\right)\left(correct_{(i,t)}\right) = \top \quad \text{iff} \quad (i,t) \notin Failed\left(r,t'\right) \tag{2.77}$$

$$\pi\left(r\left(t'\right)\right)\left(correct_i\right) = \top \quad \text{iff} \quad (i,t') \notin Failed\left(r,t'\right) \tag{2.78}$$

$$\pi\left(r\left(t'\right)\right)\left(fake_{(i,t)}\left(o\right)\right) = \top \quad \text{iff} \quad t \geq 1 \text{ and } o \in \sigma\left(\beta_{b_i}^{t-1}\left(r\right)\right) \tag{2.79}$$

$$\pi\left(r\left(t'\right)\right)\left(\overline{occurred}_{(i,t)}(o)\right) = \top \quad \text{iff} \quad t \geq 1 \text{ and } o \in label^{-1}\left(\beta_i^{t-1}\left(r\right) \sqcup \overline{\beta}_{\epsilon_i}^{t-1}\left(r\right)\right) \tag{2.80}$$

$$\pi\left(r\left(t'\right)\right)\left(\overline{occurred}_i(o)\right) = \top \quad \text{iff} \quad (\exists t < t')\, o \in label^{-1}\left(\beta_i^t\left(r\right) \sqcup \overline{\beta}_{\epsilon_i}^t\left(r\right)\right) \tag{2.81}$$

$$\pi\left(r\left(t'\right)\right)\left(occurred_i(o)\right) = \top \quad \text{iff} \quad o \in r_i\left(t'\right) \tag{2.82}$$

Proposition 2.5.2. *The following truth values coincide in all weakly (strongly) χ-interpreted systems:*[15]

$$\begin{aligned}
\pi\left(r\left(t'\right)\right)\left(correct_i\right) &= \pi\left(r\left(t'\right)\right)\left(correct_{(i,t')}\right)\\
\left(I,r,t'\right) \models faulty_i &\Leftrightarrow \left(I,r,t'\right) \models faulty_{(i,t')}\\
\left(I,r,t'\right) \models faulty_{(i,t)} &\Leftrightarrow (i,t) \in Failed\left(r,t'\right)\\
\left(I,r,t'\right) \models faulty_i &\Leftrightarrow (i,t') \in Failed\left(r,t'\right)\\
\pi\left(r\left(t'\right)\right)\left(\overline{occurred}_i(o)\right) &= \bigvee_{t=1}^{t'} \pi\left(r\left(t'\right)\right)\left(\overline{occurred}_{(i,t)}(o)\right)\\
\pi\left(r\left(t'\right)\right)\left(occurred_i(o)\right) &= \bigvee_{t=1}^{t'} \left(\pi\left(r\left(t'\right)\right)\left(\overline{occurred}_{(i,t)}(o)\right) \vee \pi\left(r\left(t'\right)\right)\left(fake_{(i,t)}\left(o\right)\right)\right)
\end{aligned}$$

Remark 2.5.3. Although these atomic propositions are objective properties, which are typically imperceptible for agents, some of them are formulated for locally representated actions/events o because they represent objective properties of agents' subjective views.

Note that no conditions are postulated for such atomic propositions if $t > t'$. This is due to the fact that π is defined on finite global histories rather than on infinite runs. The global history $r\left(t'\right)$ at timestamp t' contains no information about later timestamps $t > t'$. Indeed, there generally exist multiple $\tau_{P_\epsilon,P}$-transitional runs extending the global history $r\left(t'\right)$, due to the non-deterministic capabilities of the adversary. Since the run r cannot be singled out based on $r\left(t'\right)$ only, only the features of r already present in $r\left(t'\right)$ can be relied upon.

Note also that, using (2.60)–(2.63), we could have easily given the same definitions in terms of global histories h rather than considering them as prefixes $r\left(t'\right)$ of a transitional run r. The latter is simply what we are interested in.

Remark 2.5.4. Agents can only record their own actions and events: if an agent believes something happened, it could happen to this agent in principle.

$$\begin{aligned}
\pi\left(r\left(t'\right)\right)\left(fake_{(i,t)}\left(o\right)\right) = \top \quad &\text{implies} \quad o \in \overline{Actions}_i \sqcup \overline{Events}_i\\
\pi\left(r\left(t'\right)\right)\left(\overline{occurred}_{(i,t)}(o)\right) = \top \quad &\text{implies} \quad o \in \overline{Actions}_i \sqcup \overline{Events}_i\\
\pi\left(r\left(t'\right)\right)\left(\overline{occurred}_i(o)\right) = \top \quad &\text{implies} \quad o \in \overline{Actions}_i \sqcup \overline{Events}_i\\
\pi\left(r\left(t'\right)\right)\left(occurred_i(o)\right) = \top \quad &\text{implies} \quad o \in \overline{Actions}_i \sqcup \overline{Events}_i
\end{aligned}$$

The omniscient environment does not forget. Note that this is independent of whether agents have perfect recall because $\beta_i\left(h\right)$ is defined in (2.60) based on the environment's history.

Lemma 2.5.5. *Consider an agent-context χ, a proper (weakly) χ-based interpreted system $I = (R', \pi)$, some $o \in \overline{Actions} \sqcup \overline{Events}$, a run $r \in R'$, a node $\theta = (i,t) \in \mathcal{A} \times \mathbb{N}$, a timestamp*

[15] $\top \vee \top = \top \vee \bot = \bot \vee \top = \top$ and $\bot \vee \bot = \bot$.

$t' \geq t$, *and:*

$$\begin{aligned}
(I, r, t') &\models \quad correct_\theta \quad \leftrightarrow \quad \Box correct_\theta \\
(I, r, t') &\models \quad faulty_\theta \quad \leftrightarrow \quad \Box faulty_\theta \\
(I, r, t') &\models \quad fake_\theta(o) \quad \leftrightarrow \quad \Box fake_\theta(o) \\
(I, r, t') &\models \quad \overline{occurred}_\theta(o) \quad \leftrightarrow \quad \Box \overline{occurred}_\theta(o) \\
(I, r, t') &\models \quad faulty_i \quad \leftrightarrow \quad \Box faulty_i \\
(I, r, t') &\models \quad \overline{occurred}_i(o) \quad \leftrightarrow \quad \Box \overline{occurred}_i(o) \\
(I, r, t') &\models \quad occurred_i(o) \quad \leftrightarrow \quad \Box occurred_i(o)
\end{aligned}$$

Proof. The direction from right to left is trivial in all cases because $t' \geq t'$, hence being true at t' is part of being true in all futures of t'.

From left to right, for the first four statements, the truth is based on a particular event/action, correct or Byzantine, occurring in the global run at round $(t-1).5$, at a specific past of t', whereas for the remaining three equivalences something must have happened at an unspecified past of t'. Since all futures of t' lie to the future of this event/action and the round enumeration remains stable, the requisite event remains in the global run. □

Remark 2.5.6. Note that, unlike the atomic propositions from Lemma 2.5.5, atoms $correct_i$ are based on certain kinds of events/actions *not* having occurred yet. Thus, they may not be preserved temporally.

Further, the first four equivalences from Lemma 2.5.5 are not universal validities as they rely on $t' \geq t$. Indeed, for $t > t'$ the truth value of these atoms is not restricted, in particular, it does not depend on the run and can change arbitrarily with time.

Agents cannot both observe an event and be mistaken about observing it. More formally,

Lemma 2.5.7. *Consider a context $\gamma = (P_\epsilon, \mathscr{G}(0), \tau, \Psi)$, a proper weakly (strongly) χ-based interpreted system $I = (R', \pi)$, an agent-context $\chi = (\gamma, P)$, an event $e \in \overline{Events}$, a run $r \in R'$, a node $\theta = (i, t) \in \mathcal{A} \times \mathbb{N}$, and a timestamp $t' \geq t$:*

- $(I, r, t') \models \overline{occurred}_\theta(e) \rightarrow \neg fake_\theta(e)$
- $(I, r, t') \models fake_\theta(e) \rightarrow \neg \overline{occurred}_\theta(e)$

Proof. To prove the first implication, assume $(I, r, t') \models \overline{occurred}_\theta(e)$. By the definition of properness, this means that $e \in label^{-1}\left(\beta_i^{t-1}(r) \sqcup \overline{\beta}_{\epsilon_i}^{t-1}(r)\right)$. Since e is an event, the only option is $e \in label^{-1}\left(\overline{\beta}_{\epsilon_i}^{t-1}(r)\right)$, i.e., there must exist $E \in \overline{\beta}_{\epsilon_i}^{t-1}(r) \subset \beta_\epsilon^{t-1}(r)$ such that $e = label^{-1}(E)$. Our goal is to show that $e \notin \sigma\left(\beta_{b_i}^{t-1}(r)\right)$. The situation splits into two cases:

Case I: $e \in Ext_i$ is an external event. Then $E = external(i, e)$. By the $(t-1)$-coherence of $\beta_\epsilon^{t-1}(r) \supset \beta_{b_i}^{t-1}(r)$ we have $fake(i, external(i, e)) \notin \beta_{b_i}^{t-1}(r)$. Hence, $e \notin \sigma\left(\beta_{b_i}^{t-1}(r)\right)$.

Case II: $e = recv(j, \mu)$ is a message delivery. Then $E = grecv(i, j, \mu, id)$ for some $id \in \mathbb{N}$ (this id cannot be entirely arbitrary but this is irrelevant for the proof). By the $(t-1)$-coherence of $\beta_{\epsilon}^{t-1}(r) \supset \beta_{b_i}^{t-1}(r)$ we have $fake(i, grecv(i, j, \mu, id')) \notin \beta_{b_i}^{t-1}(r)$ for any $id' \in \mathbb{N}$. Hence, $e \notin \sigma\left(\beta_{b_i}^{t-1}(r)\right)$.

We have demonstrated that $(I, r, t') \models \overline{occurred}_\theta(e) \to \neg fake_\theta(e)$. Now the second implication $(I, r, t') \models fake_\theta(e) \to \neg\overline{occurred}_\theta(e)$ follows by contraposition. □

Remark 2.5.8. Needless to say, the absence of a correct (Byzantine) occurrence does not mean that there was a Byzantine (correct) one.

Remark 2.5.9. The same statement does not apply to actions a. For instance, the correct internal action $internal(i, a)$ is generally compatible with a Byzantine action $fake(i, A' \mapsto internal(i, a))$ the agent mistakes for a.

Using these atomic propositions with fixed evaluations, we can define derived concepts with similarly fixed meanings. For instance, the **absolute occurrence** represents information about local actions and events accessible only for the environment.

Definition 2.5.10 (Absolute occurrence)**.** Consider any integer $k \geq 1$ and any $o \in \overline{Actions} \sqcup \overline{Events}$,

$$\overline{occurred}^{(k)}(o) := \bigvee_{\substack{S \subset \mathcal{A} \\ |S| = k}} \bigwedge_{i \in S} \overline{occurred}_i(o) \tag{2.83}$$

$$\overline{occurred}(o) := \overline{occurred}^{(1)}(o) \tag{2.84}$$

We now define several notions related to **relative occurrence** $occurred_{(i,t)}(o)$ that represents agents' information about the same events:

Definition 2.5.11 (Relative occurrence)**.** Consider any agent $i \in \mathcal{A}$, any timestamp $t \in \mathbb{N}$, and any integer $k \geq 1$. For any $o \in \overline{Actions} \sqcup \overline{Events}$,

$$occurred_{(i,t)}(o) := \overline{occurred}_{(i,t)}(o) \vee fake_{(i,t)}(o) \tag{2.85}$$

$$occurred^{(k)}(o) := \bigvee_{\substack{S \subset \mathcal{A} \\ |S| = k}} \bigwedge_{i \in S} occurred_i(o) \tag{2.86}$$

$$occurred(o) := occurred^{(1)}(o) \tag{2.87}$$

Informally speaking,

- $\overline{occurred}(o)$ says that some non-Byzantine version of o happened for at least one agent;
- $\overline{occurred}^{(k)}(o)$ says that some non-Byzantine versions of o happened for at least k distinct agents.
- $occurred_{(i,t)}(o)$ says that some version of event/action o was entered into agent i's history at local timestamp (i, t), i.e., during round $(t-1).5$;

- $\mathit{occurred}(o)$ says that at least one agent believes some version of o happened;
- $\mathit{occurred}^{(k)}(o)$ says that at least k distinct agents believe some versions of o happened.

In all the three last cases agents are not aware of whether o was correct or Byzantine, and whether it really happened or they imagine it did; in the case of k agents, a mixture of correct and Byzantine entries also satisfies the conditions.

Remark 2.5.12. Note that $\mathit{occurred}^{(k)}(o)$ (resp $\overline{\mathit{occurred}}^{(k)}(o)$) requires the existence of *some* k distinct agents. In particular, in order to fulfill $K_i\mathit{occurred}^{(k)}(o)$ (resp. $K_i\overline{\mathit{occurred}}^{(k)}(o)$), it is not necessary that the same k agents observe o in all global states i considers possible. It is sufficient that in each such possible state, there be a group of k agents who have observed o.

The following is a direct corollary of Lemma 2.5.5.

Remark 2.5.13. Consider an agent-context χ, a proper weakly (strongly) χ-based interpreted system $I = (R', \pi)$, some $o \in \overline{\mathit{Actions}} \sqcup \overline{\mathit{Events}}$, a run $r \in R'$, a node $\theta = (i, t) \in \mathcal{A} \times \mathbb{N}$, and a timestamp $t' \geq t$,

$$\begin{aligned}
(I, r, t') \models \quad & \overline{\mathit{occurred}}(o) \quad \leftrightarrow \quad \Box\overline{\mathit{occurred}}(o) \\
(I, r, t') \models \quad & \overline{\mathit{occurred}}^{(k)}(o) \quad \leftrightarrow \quad \Box\overline{\mathit{occurred}}^{(k)}(o) \\
(I, r, t') \models \quad & \mathit{occurred}_{(i,t)}(o) \quad \leftrightarrow \quad \Box\mathit{occurred}_{(i,t)}(o) \\
(I, r, t') \models \quad & \mathit{occurred}(o) \quad \leftrightarrow \quad \Box\mathit{occurred}(o) \\
(I, r, t') \models \quad & \mathit{occurred}^{(k)}(o) \quad \leftrightarrow \quad \Box\mathit{occurred}^{(k)}(o)
\end{aligned}$$

Lemma 2.5.14. *Consider a context $\gamma = (P_\epsilon, \mathscr{G}(0), \tau, \Psi)$, an agent-context $\chi = (\gamma, P)$, a proper weakly (strongly) χ-based interpreted system $I = (R', \pi)$, some $o \in \overline{\mathit{Actions}} \sqcup \overline{\mathit{Events}}$, a run $r \in R'$, a node $(i, t) \in \mathcal{A} \times \mathbb{N}$, and a timestamp $t' \geq t$.*

$$(I, r, t') \models \mathit{occurred}_{(i,t)}(o) \qquad \textit{iff} \qquad t \geq 1 \textit{ and } (\exists\lambda)\big(r_i(t) = \lambda : r_i(t-1) \textit{ and } o \in \lambda\big)$$

Proof. First we prove the direction from left to right. Assume $(I, r, t') \models \mathit{occurred}_{(i,t)}(o)$, which, according to Def. 2.5.11, means

$$(I, r, t') \models \overline{\mathit{occurred}}_{(i,t)}(o) \vee \mathit{fake}_{(i,t)}(o)$$

It is clear that $t \geq 1$.

Case I: $(I, r, t') \models \overline{\mathit{occurred}}_{(i,t)}(o)$. Then $o \in \mathit{label}^{-1}\left(\beta_i^{t-1}(r) \sqcup \overline{\beta}_{\epsilon_i}^{t-1}(r)\right)$. Thus, $o = \mathit{label}^{-1}(O)$ for some $O \in \beta_i^{t-1}(r) \sqcup \overline{\beta}_{\epsilon_i}^{t-1}(r) \subset \left(\beta_i^{t-1}(r) \sqcup \beta_{\epsilon_i}^{t-1}(r)\right) \cap \left(\overline{\mathit{GActions}} \sqcup \overline{\mathit{GEvents}}\right)$. Additionally, by Prop. 2.2.25, if O is an action from $\beta_i^{t-1}(r)$, then by Prop. 2.2.25,

$$\mathit{aware}(i, \beta_\epsilon^{t-1}(r)). \tag{2.88}$$

So by (2.57), definition (2.39) of update_i, and definition (2.32) of σ we conclude that $r_i(t) = \lambda : r_i(t-1)$ and $o = \mathit{label}^{-1}(O) \in \lambda$.

Case II: $(I, r, t') \models \mathit{fake}_{(i,t)}(o)$. Then $o \in \sigma\left(\beta_{b_i}^{t-1}(r)\right)$. It remains to note that $\sigma\left(\beta_{b_i}^{t-1}(r)\right) \neq \varnothing$ implies that

$$\sigma\left(\beta_{\epsilon_i}^{t-1}(r)\right) \neq \varnothing \tag{2.89}$$

and $r_i(t) = \lambda : r_i(t-1)$. Once again, the definition (2.39) of update_i implies $o \in \lambda$.

We proved that in either case $r_i(t) = \lambda\colon r_i(t-1)$ and $o \in \lambda$.

Now we demonstrate the opposite direction from right to left. Assume $r_i(t) = \lambda\colon r_i(t-1)$ and $o \in \lambda$. By definition (2.39) of update_i it means that either (2.88) or (2.89) holds.

Case I: $o \in \mathit{label}^{-1}\left(\beta_i^{t-1}(r) \sqcup \overline{\beta}_{\epsilon_i}^{t-1}(r)\right)$. We have $(I, r, t') \models \overline{\mathit{occurred}}_{(i,t)}(o)$, and, hence, $(I, r, t') \models \mathit{occurred}_{(i,t)}(o)$.

Case II: $o \in \sigma\left(\beta_{b_i}^{t-1}(r)\right)$. We have $(I, r, t') \models \mathit{fake}_{(i,t)}(o)$, and, hence, $(I, r, t') \models \mathit{occurred}_{(i,t)}(o)$

We proved that in either case $(I, r, t') \models \mathit{occurred}_{(i,t)}(o)$. □

Lemma 2.5.15. *Consider a context $\gamma = (P_\epsilon, \mathscr{G}(0), \tau, \Psi)$, an agent-context $\chi = (\gamma, P)$, a proper weakly (strongly) χ-based interpreted system $I = (R', \pi)$, some $o \in \overline{\mathit{Actions}} \sqcup \overline{\mathit{Events}}$, a run $r \in R'$, an agent $i \in \mathcal{A}$, and a timestamp $t \in \mathbb{N}$.*

$$\begin{aligned} (I, r, t) &\models \quad \mathit{occurred}_i(o) \quad \leftrightarrow \quad K_i \mathit{occurred}_i(o) \\ (I, r, t) &\models \quad \neg \mathit{occurred}_i(o) \quad \leftrightarrow \quad K_i \neg \mathit{occurred}_i(o) \end{aligned}$$

Proof. The directions from right to left are trivial because the indistinguishability relation $\sim_i$ is reflexive. We prove the direction from left to right for the case of $(I, r, t) \models \mathit{occurred}_i(o)$ as the other statement is completely analogous. For any $(r', t') \in R' \times \mathbb{N}$ such that $r(t) \sim_i r'(t')$, i.e., $r_i(t) = r_i'(t')$, we have

$$(I, r, t) \models \mathit{occurred}_i(o) \iff o \in r_i(t) \iff o \in r_i'(t') \iff (I, r', t') \models \mathit{occurred}_i(o)$$

□

Formulas $\mathit{occurred}_i(o)$ and $\overline{\mathit{occurred}}_i(o)$ represent events occurring in the system. As shown in Lemma 2.5.15, the former event is detectable by agent i and, hence, can be used by its protocol, whereas the latter may not be detectable by any agents but is fully determined by the global state, i.e., "detectable" by the environment. Following [FHMV99], we define conditions under which formulas can be treated as events:

Definition 2.5.16. A formula φ is called an **i-internal event (within an agent-context χ)** iff

$$r_i(t) = r_i'(t') \quad \Longrightarrow \quad ((I, r, t) \models \varphi \iff (I, r', t') \models \varphi)$$

for all χ-based interpreted systems $I = (R', \pi)$, arbitrary runs $r, r' \in R'$, and arbitrary timestamps $t, t' \in \mathbb{N}$.

A formula φ is called a **state event (within an agent-context χ)** iff

$$r(t) = r'(t) \quad \Longrightarrow \quad ((I, r, t) \models \varphi \iff (I, r', t) \models \varphi)$$

for all χ-based interpreted systems $I = (R', \pi)$, arbitrary runs $r, r' \in R'$, and any timestamp $t \in \mathbb{N}$.

Lemma 2.5.17. *For any i-internal event φ within an agent context χ and any χ-based interpreted system $I = (R', \pi)$, any run $r \in R'$, and any timestamp $t \in \mathbb{N}$.,*

$$(I, r, t) \models \varphi \leftrightarrow K_i \varphi,$$
$$(I, r, t) \models \neg\varphi \leftrightarrow K_i \neg\varphi,$$

2.6 Fully Byzantine Asynchronous Agents

In the previous sections, we defined the general framework using transition templates and introduced the Byzantine transition template. It covers a wide range of settings, from crash failures to fully Byzantine agents and represents asynchronous agents with no additional requirements on communication.

However, no task can be guaranteed if agents are never allowed to act. Thus, it is standard to impose the **Fair Schedule (FS)** admissibility condition, which ensures that each correct agent will eventually be given a possibility to follow its protocol.

Definition 2.6.1 (Fair schedule)**.**

$$FS = \left\{ r \in R \mid (\forall (i,t) \in \mathcal{A} \times \mathbb{N})\,(\exists t' \geq t) \quad \beta_{g_i}^{t'}(r) \neq \varnothing \right\}.$$

Remark 2.6.2. The condition $\beta_{g_i}^{t'}(r) \neq \varnothing$ is equivalent to demanding that eventually $go(i)$, $sleep(i)$, or $hibernate(i)$ be present in $\beta_{\epsilon_i}^{t'}(r)$. In other words, the FS admissibility condition demands that the environment either provide CPU time or *wrongfully* deny CPU time for every processor infinitely many times. This means that correct processes will be treated fairly, i.e., would always be given an opportunity to act, whereas faulty processes can stop their by-the-protocol actions from some point onward. Note, however, that for this to happen, the environment still has to deal with this agent acting infinitely often, via $sleep(i)$ and/or $hibernate(i)$ commands. In other words, an agent malfunctioning only due to wrong actions/events would still be regularly fulfilling its protocol.[16] Avoiding the protocol altogether constitutes a separate type of malfunction.

Thus, allowing all agents to go rogue may also result in the complete crash failure of the whole system, which would preclude any guarantees of fulfilling the goal(s) of the joint protocol. It is, therefore, common to restrict the maximal number f of agents that can become Byzantine.

In preventing the environment from failing too many agents within a round, several options are *a priori* possible. The most general manifestation of this problem is when $f - k$ agents are already faulty and the environment's protocol attempts to fail $k + l + 1$ more agents for some $k, l \geq 0$. In the situation when k is positive, i.e., some but not all agents can still be failed, the choices are

[16] This might appear to be a restriction artificially making agents perform actions. However, an agent that can stay in standby indefinitely would simply have the empty set of actions as an option in its protocol.

1. to fail as many agents as possible, i.e., to fail some k of the proposed $k+l+1$ agents but filter out Byzantine events for the other $l+1$ agents;
2. to fail nobody, i.e., to filter all Byzantine events while keeping all correct events intact;
3. to filter all events, Byzantine or correct alike.

We find that the first option is too arbitrary as it requires to randomly choose which $l+1$ agents from $k+l+1$ are to stay correct. The third option, on the contrary, is too invasive. Given the general postulate that, unlike agents, the environment is not acting according to any plan, it is more natural to consider each event attempted by the environment in isolation, much like the inability to receive an unsent message does not preclude the environment from implementing other events.

Thus, we choose the second option and implement it by adding an additional function $filter_\epsilon^{\leq f}$ for the environment.

Definition 2.6.3 (Filtering for at most f Byzantine agents)**.** For a set $X_\epsilon \subset GEvents$ and agent $i \in \mathcal{A}$, we abbreviate

$$X_{\epsilon_i}^B := X_\epsilon \cap \left(BEvents_i \sqcup \{sleep\,(i), hibernate\,(i)\}\right) \tag{2.90}$$

and define

$$filter_\epsilon^{\leq f}\,(h, X_\epsilon, X_1, \ldots, X_n) := \begin{cases} X_\epsilon & \text{if } \left|\mathcal{A}(Failed\,(h)) \cup \left\{i \mid X_{\epsilon_i}^B \neq \varnothing\right\}\right| \leq f \\ X_\epsilon \setminus \bigcup\limits_{i\in\mathcal{A}} X_{\epsilon_i}^B & \text{otherwise,} \end{cases} \tag{2.91}$$

which removes all Byzantine commands from X_ϵ whenever they would have led to creating more than f faulty agents.

Remark 2.6.4. It might seem that the filters $filter_\epsilon^{\leq f}$ and $filter_\epsilon^B$ are completely independent, i.e., they can be applied in any order. After all, by (2.91) one of them only removes Byzantine events, while by (2.24) the other only removes *grecv* events, which are correct. Unfortunately, this is not entirely accurate. It is possible that a *grecv* command is not filtered by $filter_\epsilon^B$ based on a Byzantine send from the same round. If $filter_\epsilon^{\leq f}$ is applied after that and happens to filter this fake send out, the receipt of the message becomes causally problematic. Thus, the only correct order of applying these two filters is

$$filter_\epsilon^B\left(h, filter_\epsilon^{\leq f}\,(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right).$$

Definition 2.6.5. Given filters $filter_\epsilon^{Z_1}$ and $filter_\epsilon^{Z_2}$ we write

$$filter_\epsilon^{Z_2\circ Z_1}\,(h, X_\epsilon, X_1, \ldots, X_n) := filter_\epsilon^{Z_2}\left(h, filter_\epsilon^{Z_1}\,(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right). \tag{2.92}$$

We also simplify this notation for the combination of the Byzantine and at most f faults filters

$$filter_\epsilon^{B_f}\,(h, X_\epsilon, X_1, \ldots, X_n) := filter_\epsilon^{B\circ\leq f}\,(h, X_\epsilon, X_1, \ldots, X_n). \tag{2.93}$$

Remark 2.6.6. Unlike FS, the upper bound on Byzantine agents cannot be formulated as an admissibility condition if agent-contexts are to be non-excluding. Indeed, a run without such an admissibility condition imposed may incur $> f$ Byzantine agents already in a finite prefix, in which case it would be impossible to extend such a prefix to a run satisfying the upper bound.

This is a typical example of separation between properties determined by a finite prefix of a run (safety properties) on the one hand and properties of the run as a whole (liveness properties) on the other hand. The non-exclusion requirement precludes the former from being imposed via admissibility conditions, which are non-constructive and, hence, should only be used as the last resort.

Definition 2.6.7 (f-Byzantine transition template)**.** For a bound $f \geq 0$, the **f-Byzantine transition template** τ^{B_f} is obtained by replacing $filter_\epsilon^B$ in the definition of τ^B with $filter_\epsilon^{B_f}$.

Definition 2.6.8 (f-Byzantine agent-context)**.** For a bound $f \geq 0$, we call an agent-context

$$\chi = \Big((P_\epsilon, \mathscr{G}(0), \tau^{B_f}, FS), P\Big) \tag{2.94}$$

f-Byzantine when it is based on the transition template τ^{B_f} and has the admissibility condition FS.

Remark 2.6.9. It is easy to see that for $f \geq |\mathcal{A}|$, we have $filter_\epsilon^{B_f} = filter_\epsilon^B$. Indeed, if all agents can become Byzantine simultaneously, the need to restrict their number never materializes. Hence, we generally assume that $f \leq |\mathcal{A}|$. The case of $f = |\mathcal{A}|$ is useful for uniform statements about both $filter_\epsilon^{B_f}$ and $filter_\epsilon^B$.

Lemma 2.6.10. *If the set*

$$\{t \mid (\exists X_\epsilon \in P_\epsilon(t))\, go(i) \in X_\epsilon\} \tag{2.95}$$

is infinite for each $i \in \mathcal{A}$, then the f-Byzantine agent-context (2.94) *is non-excluding.*

Proof. In order to extend a given finite prefix of a weakly χ-consistent run to a consistent one, it is sufficient to make $\beta_{g_i}^t(r)$ non-empty infinitely many times. Since the filter function $filter_\epsilon^{B_f}$ never removes $go(i)$, it is sufficient that $\alpha_{g_i}^t(r)$ contain it infinitely many times, which can be achieved in the rest of the run by the adversary if $(\exists X_\epsilon \in P_\epsilon(t))\, go(i) \in X_\epsilon$ holds for infinitely many t's. □

Remark 2.6.11. Note that, despite its name, a non-excluding f-Byzantine agent-context χ provides neither a guarantee that Byzantine events *will* happen in a particular run nor, indeed, that they *can* happen in a transitional run at all. Thus, our model can represent both correct runs and infallible systems. Indeed, Byzantine events will not occur in a run if the adversary part of the environment never chooses them. But it chooses them out of the possibilities afforded by the environment's protocol P_ϵ. If P_ϵ contains no Byzantine events, the adversary is powerless to effect them.

Lemma 2.6.12. $filter_\epsilon^{B_f} \subset filter_\epsilon^B$.

Proof. If $E \in GEvents$ is any event other than a correct receive, then

$$E \in filter_\epsilon^{B_f}(h, X_\epsilon, X_1, \ldots, X_n) \quad \Rightarrow \quad E \in X_\epsilon \quad \Rightarrow \quad E \in filter_\epsilon^{B}(h, X_\epsilon, X_1, \ldots, X_n).$$

by (2.54) and the fact that $filter_\epsilon^B$ only removes correct receives. If

$$E = grecv(i,j,\mu,id) \in filter_\epsilon^{B_f}(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^{B}\left(h, filter_\epsilon^{\leq f}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right)$$

then $grecv(i,j,\mu,id) \in filter_\epsilon^{\leq f}(h, X_\epsilon, X_1, \ldots, X_n) \subset X_\epsilon$ and for it to remain, one of the following options must be fulfilled

- $gsend(j,i,\mu,id) \in h_\epsilon$ or $fake(j, gsend(j,i,\mu,id) \mapsto A) \in h_\epsilon$ for some $A \in \{\text{📅}\} \sqcup \overline{GActions_j}$.
- $gsend(j,i,\mu,id) \in X_j$ and $go(j) \in filter_\epsilon^{\leq f}(h, X_\epsilon, X_1, \ldots, X_n) \subset X_\epsilon$.
- $fake(j, gsend(j,i,\mu,id) \mapsto A) \in filter_\epsilon^{\leq f}(h, X_\epsilon, X_1, \ldots, X_n) = X_\epsilon$ for some $A \in \{\text{📅}\} \sqcup \overline{GActions_j}$ (the equality in this case follows from the fact that there are Byzantine events left after the filtering).

In each of the options, the same reasoning applies to $filter_\epsilon^B(h, X_\epsilon, X_1, \ldots, X_n)$ equally well because the same h and X_j are used whereas X_ϵ can only become larger. □

Corollary 2.6.13. *All statements from Lemma 2.3.15, Cor. 2.3.16, Cor. 2.3.18, and Lemma* **??** *hold also for contexts* $\gamma = (P_\epsilon, \mathscr{G}(0), \tau^{B_f}, \Psi)$.

So far we have not postulated that Byzantine events must be present in P_ϵ. We will now define several types of protocols that ensure the possibility of particular types of errors, up to the case of fully Byzantine agents, i.e., agents that are in principle capable of any malfunction imaginable.

Definition 2.6.14 (Types of agents)**.** Given an agent-context

$$\left((P_\epsilon, \mathscr{G}(0), \tau, FS), P\right), \tag{2.96}$$

an agent $i \in \mathcal{A}$ in this agent-context is called

- **fallible** if for any $X \in P_\epsilon(t)$,

  $$X \cup \{fail(i)\} \quad \in \quad P_\epsilon(t). \tag{2.97}$$

  In other words, an agent is fallible if it can be branded Byzantine at any moment;
- **infallible** if for any $X \in P_\epsilon(t)$,

  $$X \cap (BEvents_i \sqcup \{sleep(i), hibernate(i)\}) = \varnothing. \tag{2.98}$$

  An infallible agent cannot become Byzantine;

- **degradable** if for any $Y \subset BEvents_i$ and any $X \in P_\epsilon(t)$,

$$X \cup Y \in P_\epsilon(t) \text{ whenever it is } t\text{-coherent;} \tag{2.99}$$
$$(X \setminus SysEvents_i) \cup Y \sqcup \{sleep(i)\} \in P_\epsilon(t) \text{ whenever it is } t\text{-coherent;} \tag{2.100}$$
$$(X \setminus SysEvents_i) \cup Y \sqcup \{hibernate(i)\} \in P_\epsilon(t) \text{ whenever it is } t\text{-coherent.} \tag{2.101}$$

  In other words, an agent is degradable if it can always make more mistakes;

- **correctable** if for any $X \in P_\epsilon(t)$,

$$X \setminus (BEvents_i \sqcup \{sleep(i), hibernate(i)\}) \quad \in \quad P_\epsilon(t). \tag{2.102}$$

  In other words, an agent is correctable if it can always refrain from all mistakes;

- **error-prone** if for any $Y \subset BEvents_i$ and any $X \in P_\epsilon(t)$,

$$(X \setminus (BEvents_i \sqcup \{sleep(i), hibernate(i)\})) \sqcup Y \in P_\epsilon(t) \text{ whenever it is } t\text{-coherent;} \tag{2.103}$$
$$(X \setminus (BEvents_i \sqcup SysEvents_i)) \sqcup Y \sqcup \{sleep(i)\} \in P_\epsilon(t) \text{ whenever it is } t\text{-coherent;} \tag{2.104}$$
$$(X \setminus (BEvents_i \sqcup SysEvents_i)) \sqcup Y \sqcup \{hibernate(i)\} \in P_\epsilon(t) \text{ whenever it is } t\text{-coherent.} \tag{2.105}$$

  In other words, an agent is error-prone if it can commit any combination of Byzantine actions/events in any round;

- **delayable** if for any $X \in P_\epsilon(t)$,

$$X \setminus GEvents_i \quad \in \quad P_\epsilon(t). \tag{2.106}$$

  In other words, an agent is delayable if all its activities can be correctly postponed in any round, forcing its local state to remain unchanged after such a round (note that the absence of $go(i)$ also prevents the agent from acting on its own);

- **gullible** if for any $Y \subset BEvents_i$ and any $X \in P_\epsilon(t)$,

$$(X \setminus GEvents_i) \sqcup Y \in P_\epsilon(t) \text{ whenever it is } t\text{-coherent;} \tag{2.107}$$
$$(X \setminus GEvents_i) \sqcup Y \sqcup \{sleep(i)\} \in P_\epsilon(t) \text{ whenever it is } t\text{-coherent;} \tag{2.108}$$
$$(X \setminus GEvents_i) \sqcup Y \sqcup \{hibernate(i)\} \in P_\epsilon(t) \text{ whenever it is } t\text{-coherent.} \tag{2.109}$$

  In other words, an agent is gullible if all its activities during a round can be replaced with an arbitrary set of Byzantine events;

- **isolatable** if for any $X \in P_\epsilon(t)$,

$$X \setminus \{grecv(i, j, \mu, id) \mid j \in \mathcal{A}, \mu \in Msgs, id \in \mathbb{N}\} \quad \in \quad P_\epsilon(t). \tag{2.110}$$

  In other words, an agent is isolatable if all correct message deliveries to it can be postponed at any round;

- **distractible** if for any $X \in P_\epsilon(t)$,

$$X \setminus \overline{GEvents_i} \quad \in \quad P_\epsilon(t). \tag{2.111}$$

  In other words, an agent is distractible if it can miss all external events, including all incoming messages;

- **impotent** if for any $X \in P_\epsilon(t)$,

$$X \setminus \{go(i)\} \quad \in \quad P_\epsilon(t). \tag{2.112}$$

  In other words, an agent is impotent if it is always possible it does not try to act;

- **fully Byzantine** if it is error-prone and gullible. In other words, whatever combination of correct and faulty events can happen to i, the same correct events are compatible with any other collection of faulty events (error-proneness) and, at the same time, any such collection of faulty events could happen without any correct events whatsoever (gullibility).

In all cases where the new set is not explicitly required to be t-coherent, it can be shown to be t-coherent whenever X is.

Remark 2.6.15. Note that error-proneness, degradability, gullibility, and full Byzanteneity means that the agent can become Byzantine in some runs (for instance, adding $fail(i)$ never violates t-coherency); hence, such agents are not infallible. On the other hand, delayability, isolatability, distractibility, and impotence do not necessarily imply any wrongdoing.

Corollary 2.6.16.

- *Any agent that is degradable, error-prone, or gullible is fallible.*
- *Any agent that is error-prone is degradable and correctable.*
- *An agent that is both error-prone and delayable is also gullible.*
- *An agent that is gullible is delayable.*
- *An agent that is fallible, degradable, error-prone, or gullible cannot be infallible.*
- *A fully Byzantine agent is gullible, error-prone, fallible, degradable, correctable, and delayable.*

Definition 2.6.17. For an upper bound $f \geq 0$, an agent-context (2.94) is called **fully f-Byzantine**, or simply **fully Byzantine**, if all agents are fully Byzantine.

Lemma 2.6.18. *All fully f-Byzantine agent-contexts are non-excluding.*

Proof. It is sufficient to issue $sleep(i)$ in every round. □

2.7 Run Modifications

Definition 2.7.1. An **intervention** for an agent $i \in \mathcal{A}$, or i**-intervention** is a function

$$\rho\colon R \longrightarrow 2^{\overline{GActions_i}} \times 2^{GEvents_i}.$$

The set of all i-interventions is denoted by

$$Intervs\,(i) := \left\{\rho \mid \rho\colon R \longrightarrow 2^{\overline{GActions_i}} \times 2^{GEvents_i}\right\}.$$

One intervention $\rho(r)$ is intended to modify the behavior of one agent in one round of a given run r in a desired way. Whether this modification relies on the agent's original behavior in r or is a complete departure from it, can be encoded in the function ρ. The output

$$(X_i, X_{\epsilon_i}) = \rho(r)$$

is intended to represent a pair of sets $\beta_i^t\,(r') = X_i$ of actions by i and $\beta_{\epsilon_i}^t\,(r') = X_{\epsilon_i}$ of events imposed on i in the same round of a modified run r'. For ease of notation we also define

$$\mathfrak{a}\rho(r) := \pi_1\rho(r);$$
$$\mathfrak{e}\rho(r) := \pi_2\rho(r).$$

In other words, if $\rho(r) = (X_i, X_{\epsilon_i})$, then $\mathfrak{a}\rho(r) = X_i$ and $\mathfrak{e}\rho(r) = X_{\epsilon_i}$.

Definition 2.7.2. A **joint intervention** is a collection of i-interventions for all agents $i \in \mathcal{A}$. We denote the set of all joint interventions

$$Intervs := \prod_{i\in\mathcal{A}} Intervs\,(i). \tag{2.113}$$

Now, let us define an adjustment of a run as a timewise list of joint interventions. Each joint intervention is to be performed at the corresponding timestamp. It is defined as follows

Definition 2.7.3. An **adjustment**

$$[B_t; \ldots; B_0]$$

is a sequence of joint interventions $B_0 \ldots, B_t \in Intervs$ to be performed at successive timestamps from 0 to some $t \in \mathbb{N}$, which is called the **extent** of the adjustment. We denote the set of adjustments

$$Adjusts := \bigcup_{t\in\mathbb{N}} \{[B_t; \ldots; B_0] \mid B_0 \ldots, B_t \in Intervs\}. \tag{2.114}$$

Definition 2.7.4. Let $adj \in Adjusts$ be an adjustment

$$adj = [B_t; \ldots; B_0] \tag{2.115}$$

of extent $t \in \mathbb{N}$ where

$$B_m = (\rho_1^m, \ldots, \rho_n^m) \tag{2.116}$$

for each $0 \leq m \leq t$ (recall that $\mathcal{A} = \{1, \ldots, n\}$), where each ρ_i^m is an i-intervention. Let a run r be a $\tau_{P_\epsilon,P}$-transitional run. We say that a run r' is **obtained from** r **by adjustment** adj, or simply is adj**-adjusted variant of** r iff

$$r'(0) = r(0), \tag{2.117}$$

$$(\forall i \in \mathcal{A})(\forall t' \leq t) \quad \left(\beta_i^{t'}(r'),\, \beta_{\epsilon_i}^{t'}(r')\right) = \rho_i^{t'}(r), \tag{2.118}$$

$$(\forall i \in \mathcal{A})(\forall t' \leq t) \quad r_i'(t'+1) = update_i\left(r_i'(t'), \beta_i^{t'}(r'), \bigcup_{i\in\mathcal{A}} \beta_{\epsilon_i}^{t'}(r')\right), \tag{2.119}$$

$$(\forall t' \leq t) \quad r_\epsilon'(t'+1) = update_\epsilon\left(r_\epsilon'(t'), \quad \left(\bigcup_{i\in\mathcal{A}} \beta_{\epsilon_i}^{t'}(r'), \beta_1^{t'}(r'), \ldots, \beta_n^{t'}(r')\right)\right), \tag{2.120}$$

$$(\forall t' > t) \quad r'(t'+1) \in \tau_{P_\epsilon,P}(r'(t')). \tag{2.121}$$

We denote by $R(\tau_{P_\epsilon,P}, r, adj)$ the set of all adj-adjusted variants of the run r, computed under the transition function $\tau_{P_\epsilon,P}$.

Remark 2.7.5. The first property ensures that both runs start from the same initial state. The second one means that the β-sets of the new run for each agent i, i.e., the actions and events affecting i in rounds 0.5 through t.5, are fully determined by the adjustment. In the absence of global events, this means that everything happening during these rounds is controlled by adj. The third and fourth ones faithfully implement the updating phase of the round (see Figure 2.1) for local and environment states respectively for the extent of the adjustment. Finally, the last property ensures that beyond the adjusted segment, the new run extends in a $\tau_{P_\epsilon,P}$-transitional manner.

Remark 2.7.6. Though r is assumed to be $\tau_{P_\epsilon,P}$-transitional, *a priori* its adjusted variants need not be. Indeed, as noted above, the local and global histories are always updated in a consistent manner, i.e., in accordance with (2.57) and (2.59), but the artificial β-sets imposed by the adjustment adj may not follow the rules of the protocol, adversary, labelling, and filtering phases (see Figure 2.1).

Lemma 2.7.7. *For any run r, any transition function $\tau_{P_\epsilon,P}$, and any adjustment adj*

$$R(\tau_{P_\epsilon,P}, r, adj) \neq \varnothing.$$

Proof. The initial t-prefix of the desired adjusted run, more precisely the behavior up till and including the round t.5, is fully determined by adj and properties (2.117)–(2.120) and exists due to the totality of all the functions involved. The intended behavior starting from the round $(t+1)$.5 is governed by (2.121). Since both P_ϵ and P satisfy the no-apocalypse clause (see Remark 2.2.11), there is always at least one option for continuing the run in every round. Once again, the existence of adjusted runs does not generally imply that they are strongly consistent or even transitional. □

The primary method we use to show that an agent does not/cannot know some fact φ is taking a(n) existing/arbitrary run and adjusting it in a way that is imperceptible for this agent but makes φ false. Note that it is generally not sufficient to intervene with the behavior of only this agent because its local state might be affected by correct messages received from

other agents. Thus, the behavior of other agents generally needs to be modified too. There are several types of interventions useful to achieve the needed adjustments.

The interventions needed for the agent who should not distinguish the original and adjusted runs replace each correct action or event it experienced in the original run with a Byzantine event that looks the same to the agent. Given that the agent's perception can deviate from the real actions/events, there is a range of choices regarding what really happens in the adjusted run. We present two straightforward options representing passive and active interventions $PFake_i$ and $AFake_i$ for agent i respectively.

Definition 2.7.8. For an agent $i \in \mathcal{A}$ and a run $r \in R$, we define i-interventions

$$PFake_i^t, AFake_i^t \quad : \quad R \to 2^{\overline{GActions_i}} \times 2^{GEvents_i}$$

as follows:

$$\begin{aligned} PFake_i^t(r) := \Big(&\varnothing, \\ &\beta_{b_i}^t(r) \quad \cup \quad \left\{fake(i,E) \mid E \in \overline{\beta}_{\epsilon_i}^t(r)\right\} \quad \cup \quad \left\{fake(i, \text{📅} \mapsto A) \mid A \in \beta_i^t(r)\right\} \quad \sqcup \\ &\{sleep(i) \mid aware(i, \beta_{\epsilon_i}^t(r))\} \quad \sqcup \quad \{hibernate(i) \mid unaware(i, \beta_{\epsilon_i}^t(r))\}\Big) \end{aligned} \tag{2.122}$$

$$\begin{aligned} AFake_i^t(r) := \Big(&\varnothing, \\ &\beta_{b_i}^t(r) \quad \cup \quad \left\{fake(i,E) \mid E \in \overline{\beta}_{\epsilon_i}^t(r)\right\} \quad \cup \quad \left\{fake(i, A \mapsto A) \mid A \in \beta_i^t(r)\right\} \quad \sqcup \\ &\{sleep(i) \mid aware(i, \beta_{\epsilon_i}^t(r))\} \quad \sqcup \quad \{hibernate(i) \mid unaware(i, \beta_{\epsilon_i}^t(r))\}\Big) \end{aligned} \tag{2.123}$$

Remark 2.7.9. The only difference between these two i-interventions lies in what the agent actually does while erroneously thinking that it did an action A. In case of the passive version $PFake_i$, agent i does not do anything, whereas in the active version $AFake_i$, agent i does perform action A, albeit in a Byzantine fashion. The two i-interventions coincide on Byzantine events.

Lemma 2.7.10. *Let $\rho \in \{PFake_i^t, AFake_i^t \mid t \in \mathbb{N}\}$ be an intervention and r and r' be arbitrary runs. Then*

1. $\mathfrak{a}\rho(r) = \varnothing$, *i.e., these interventions always produce the empty set of actions.*
2. $go(i) \notin \mathfrak{e}\rho(r)$, *i.e., these interventions never let agent i act.*
3. $|\mathfrak{e}\rho(r) \cap \{sleep(i), hibernate(i)\}| = 1$, *i.e., these interventions always intend to make agent i Byzantine by means of exactly one of commands $sleep(i)$ or $hibernate(i)$.*
4. $\sigma(\mathfrak{a}\rho(r) \cup \mathfrak{e}\rho(r)) = \sigma(\mathfrak{e}\rho(r)) = \sigma\left(\beta_i^t(r) \cup \beta_{\epsilon_i}^t(r)\right)$, *where $\rho \in \{PFake_i^t, AFake_i^t\}$, i.e., events and actions intended to be appended to the local history of agent i as a result of round $t.5$ after the intervention $PFake_i^t$ or $AFake_i^t$ are the same as before the intervention in the same round of the original run r.*[17]

[17] In some cases, the local history remains unaffected by these sets: namely, if there is no events/actions to add and the agent is unaware of the passing round, but the statement is true in this case too.

5. *aware*$(i, \mathfrak{e}\rho(r)) = aware(i, \beta^t_{\epsilon_i}(r))$, *where* $\rho \in \{PFake^t_i, AFake^t_i\}$, *i.e., the awareness of agent* i *of the passing of round* $t.5$ *is not changed by* $PFake^t_i$ *or* $AFake^t_i$.[18]

Proof. The only properties that do not directly follow from the definition are the last three.

Property 3 follows from the fact that *unaware*(i, Z) is defined to be the negation of *aware*(i, Z) (see Def. 2.2.17). Hence, exactly one of them always holds.

Property 4 follows from

$$\sigma(\{fake\,(i, E)\}) = \sigma(\{E\}) \qquad \text{and} \qquad \sigma(\{fake\,(i, \text{📅} \mapsto A)\}) = \sigma(\{fake\,(i, A \mapsto A)\}) = \sigma(\{A\})$$

which are a direct consequence of (2.32). Note that in the latter case $A \in \beta^t_i(r) \subset \overline{GActions_i}$.

For Property 5, note that $|\mathfrak{e}\rho(r) \cap \{go(i), sleep\,(i), hibernate\,(i)\}| = 1$ by Properties 2 and 3. Hence, there are exactly two possibilities: either *unaware*$(i, \mathfrak{e}\rho(r))$ due to the presence of *hibernate* (i) or *awarei*$\mathfrak{e}\rho(r)$ due to the presence of *sleep* (i), equivalently, due to the absence of *hibernate* (i). More precisely,

$$aware(i, \mathfrak{e}\rho(r)) = t \iff hibernate\,(i) \notin \mathfrak{e}\rho(r) \iff sleep\,(i) \in \mathfrak{e}\rho(r) \iff aware(i, \beta^t_{\epsilon_i}(r)) = t.$$

□

Another common construction is freezing an agent, i.e., allowing no actions or events update its local history. Such a behavior is captured by an i-intervention *CFreeze* if the agent is to remain correct or by $BFreeze_i$ if the agent is to become Byzantine. It is defined as follows.

Definition 2.7.11. For a run $r \in R$, we define

$$CFreeze\,(r) := (\varnothing, \varnothing). \tag{2.124}$$

It can serve as an i-intervention for any agent i.

Definition 2.7.12. For an agent $i \in \mathcal{A}$ and a run $r \in R$, we define

$$BFreeze_i\,(r) := (\varnothing, \{fail\,(i)\}). \tag{2.125}$$

Note that interventions *CFreeze* and $BFreeze_i$ are constant, in other words, the modifications they initiate are run-independent.

Sometimes we want an intervention that preserves the exact behavior of an agent $i \in \mathcal{A}$ during round $t.5$.

Definition 2.7.13. For an agent $i \in \mathcal{A}$, a run $r \in R$, and a timestamp $t \in \mathbb{N}$,

$$Copy^t_i\,(r) := (\beta^t_i(r), \beta^t_{\epsilon_i}(r)). \tag{2.126}$$

[18] In some cases, the local history does not depend on such awareness: namely, if there are events/actions to add, but the statement is true in this case too.

Sometimes we want an agent $i \in \mathcal{A}$ during round $t.5$ to concentrate on important messages originating from a specific set X of nodes ignoring the chatter from outside of this set but otherwise to carry on as without the intervention. We will often use a causal cone as X.

Definition 2.7.14. For an agent $i \in \mathcal{A}$, a run $r \in R$, a timestamp $t \in \mathbb{N}$, and a set $X \subset \mathcal{A} \times \mathbb{N}$ of nodes,

$$X\text{-}Focus_i^t(r) := \Big(\beta_i^t(r), \qquad \beta_{\epsilon_i}^t(r) \setminus \{grecv(i,j,\mu,id(j,i,\mu,k,m)) \mid (j,m) \notin X, k \in \mathbb{N}\}\Big). \tag{2.127}$$

Finally, sometimes we do not care about agent i itself but only need it to produce the same communication as in a given round.

Definition 2.7.15. For an agent $i \in \mathcal{A}$, a run $r \in R$, and a timestamp $t \in \mathbb{N}$,

$$FakeEcho_i^t(r) := \Big(\varnothing, \{fail(i)\} \sqcup \{fake(i, gsend(i,j,\mu,id) \mapsto \text{📅}) \mid gsend(i,j,\mu,id) \in \beta_i^t(r) \text{ or } (\exists A \in \overline{GActions_i} \sqcup \{\text{📅}\})\, fake(i, gsend(i,j,\mu,id) \mapsto A) \in \beta_{b_i}^t(r)\}\Big). \tag{2.128}$$

Using some of the assumptions on agents defined at the end of Chapter 2, we can prove the following lemma.

Lemma 2.7.16 (Brain-in-the-Vat Lemma)**.** *Let $\mathcal{A} = [\![1;n]\!]$ be the set of agents with protocols $P = (P_1, \ldots, P_n)$, let P_ϵ be the protocol of the environment, let r be a $\tau^B_{P_\epsilon,P}$-transitional run, let i be an agent, let $t > 0$ be a timestamp, and let $adj = [B_{t-1}; \ldots; B_0]$ be an adjustment of extent $t-1$ satisfying (2.116) for all $0 \le m \le t-1$ with*

$$\rho_i^m = PFake_i^m \qquad \text{and} \qquad \text{for all } j \neq i \quad \rho_j^m \in \{CFreeze, BFreeze_j\}.$$

If the protocol P_ϵ makes

- *agent i gullible,*
- *every agent $j \neq i$ delayable and fallible if $\rho_j^m = BFreeze_j$ for some m,*
- *all remaining agents delayable,*

then each run $r' \in R\left(\tau^B_{P_\epsilon,P}, r, adj\right)$ satisfies the following properties:

1. *r' is $\tau^B_{P_\epsilon,P}$-transitional;*
2. *$(\forall m \le t)\, r'_i(m) = r_i(m)$;*
3. *$(\forall m \le t)\,(\forall j \neq i)\; r'_j(m) = r'_j(0)$.*
4. *$(i,1) \in Bad(r',1)$ and, consequently, $(i,m) \in Failed(r',m')$ for all $m' \ge m > 0$;*

5. $\mathcal{A}\left(Failed\left(r'\left(t\right)\right)\right) = \{i\} \cup \{j \neq i \mid (\exists m \leq t-1)\, \rho_j^m = BFreeze_j\}$.

Proof. We prove all these statements alongside the following properties:

6. $(\forall m < t)\,(\forall j \neq i)\ \beta_{\epsilon_j}^m\left(r'\right) \subset \{fail\left(j\right)\}$.
   More precisely, $\beta_{\epsilon_j}^m\left(r'\right) = \varnothing$ iff $\rho_j^m = CFreeze$ and $\beta_{\epsilon_j}^m\left(r'\right) = \{fail\left(j\right)\}$ iff $\rho_j^m = BFreeze_j$;
7. $(\forall m < t)\, \beta_{\epsilon_i}^m\left(r'\right) \setminus \beta_{f_i}^m\left(r'\right) = \varnothing$;
8. $(\forall m < t)(\forall j \in \mathcal{A})\, \beta_j^m\left(r'\right) = \varnothing$.

Consider an arbitrary $r' \in R\left(\tau_{P_\epsilon,P}^B, r, adj\right)$.

Property 4 follows from Lemma 2.7.10(3).

Property 5 follows from Property 4 and the fact that the only event assigned other agents $j \neq i$ is $fail\left(j\right)$ and it is only assigned by $BFreeze_j$, making all agents with $BFreeze_j$ interventions Byzantine while leaving all agents without $BFreeze_j$ interventions correct.

Property 6 follows from (2.124) and (2.125).

Property 7 follows from (2.122).

Property 8 follows from Lemma 2.7.10(1) for i and from (2.124) and (2.125) for $j \neq i$.

The remaining three properties except for the first depend solely on the first t rounds of r' and the first property starting from round $t.5$ directly follows from (2.121). Thus, it remains to show Properties 1–3 for $m \leq t$ by induction on m.

Base: $m = 0$**.** Properties 1 and 3 are trivial, whereas Property 2 follows from (2.117).

Step from m **to** $m+1$**.** We prove Property 1 based on the gullibility of i and delayability (and fallibility) of all other $j \neq i$. In order to show that $r'\left(m+1\right) \in \tau_{P_\epsilon,P}^B(r'\left(m\right))$, we need to demonstrate that the β-sets prescribed by adj can be obtained in a regular round. Since the adversary's choice of actions $\alpha_j^m\left(r\right)$ is immaterial due to the absence of $go(j)$ by Lemma 2.7.10(2) for i and by (2.124)/(2.125) for other $j \neq i$, we concentrate on showing which α-sets of events the adversary needs to choose. Consider $\alpha_\epsilon^m\left(r\right) \in P_\epsilon\left(m\right)$ from the original run r. It must be m-coherent because r is transitional. By the delayability of all $j \neq i$,

$$\alpha_\epsilon^m\left(r\right) \setminus \bigsqcup_{j \neq i} GEvents_j \quad = \quad \alpha_{\epsilon_i}^m\left(r\right) \quad \in \quad P_\epsilon\left(m\right).$$

Note that for any $Z \subset BEvents_i \sqcup \{sleep\left(i\right), hibernate\left(i\right)\}$,

$$\left(\alpha_{\epsilon_i}^m\left(r\right) \setminus GEvents_i\right) \sqcup Z \quad = \quad \varnothing \sqcup Z \quad = \quad Z$$

because $\alpha_{\epsilon_i}^m\left(r\right) \subset GEvents_i$. Thus, by the gullibility of i,

$$\begin{aligned} Y_0 \quad := \quad & \left(\beta_{\epsilon_i}^m\left(r\right) \cap BEvents_i\right) \quad \cup \\ & \left\{fake(i,E) \mid E \in \beta_{\epsilon_i}^m\left(r\right) \cap \overline{GEvents_i}\right\} \quad \cup \quad \{fake\left(i, \text{📅} \mapsto A\right) \mid A \in \beta_i^m\left(r\right)\} \quad \sqcup \\ & \{sleep\left(i\right) \mid aware(i, \beta_{\epsilon_i}^m\left(r\right))\} \quad \sqcup \quad \{hibernate\left(i\right) \mid unaware(i, \beta_{\epsilon_i}^m\left(r\right))\} \quad \in \quad P_\epsilon\left(m\right) \end{aligned}$$

if it is m-coherent. Note that exactly one of $sleep(i)$ and $hibernate(i)$ is added to Y_0 meaning that condition 2 of m-coherency is fulfilled (see Def. 2.2.1). Conditions 3–5 of m-coherency are trivially fulfilled because Y_0 contains no correct events. Finally, condition 1 is fulfilled because all fake actions in Y_0 either have 🗓 actually performed or originate from $\beta^m_{\epsilon_i}(r) \subset \alpha^m_{\epsilon_i}(r) \subset \alpha^m_{\epsilon}(r)$, the latter being an m-coherent set. Thus, $Y_0 \in P_\epsilon(m)$. Finally, by the fallibility of all agents $j \neq i$ with $\rho^m_j = BFreeze_j$,

$$Y \quad := \quad Y_0 \cup \{fail(j) \mid \rho^m_j = BFreeze_j\} \quad \in \quad P_\epsilon(m).$$

This Y is m-coherent and unaffected by filtering there are no correct events in Y to be filtered out. We conclude that these choices by the adversary result, after the filtering phase, in

$$\begin{aligned} &\beta^m_\epsilon(r) = Y, \\ &\beta^m_j(r) = \varnothing \qquad\qquad \text{for all } j \in \mathcal{A}. \end{aligned}$$

The latter is due to $go(j) \notin Y$ for any $j \in \mathcal{A}$. It remains to note that

$$\begin{aligned} &\beta^m_{\epsilon_i}(r) = \beta^m_\epsilon(r) \cap GEvents_i = Y_0, \\ &\beta^m_{\epsilon_j}(r) = \beta^m_\epsilon(r) \cap GEvents_j = \begin{cases} \varnothing & \text{if } \rho^m_j = CFreeze, \\ \{fail(j)\} & \text{if } \rho^m_j = BFreeze_j \end{cases} \qquad \text{for other } j \neq i. \end{aligned}$$

This completes the induction step for Property 1.

For Property 2, the induction step follows from Lemma 2.7.10(4)–(5). More precisely, given that $\sigma(Y) = \sigma(Y_0)$, we have the following cases:

- if $\sigma\left(\beta^m_{\epsilon_i}(r)\right) \neq \varnothing$, then

  $$r_i(m+1) = \sigma(\beta^m_i(r) \sqcup \beta^m_{\epsilon_i}(r)) : r_i(m) = \sigma(\beta^m_i(r) \sqcup \beta^m_{\epsilon_i}(r)) : r'_i(m)$$

  by the induction hypothesis. It remains to use Lemma 2.7.10(4) to see that it is the same as

  $$\sigma(Y) : r'_i(m) = \sigma(\varnothing \sqcup Y_0) : r'_i(m) = \sigma(\beta^m_i(r') \sqcup \beta^m_{\epsilon_i}(r')) : r'_i(m) = r'_i(m+1) \quad (2.129)$$

  because $\sigma\left(\beta^m_{\epsilon_i}(r')\right) = \sigma(Y_0) = \sigma\left(\beta^m_i(r) \sqcup \beta^m_{\epsilon_i}(r)\right) \supset \sigma\left(\beta^m_{\epsilon_i}(r)\right) \neq \varnothing$.

- if $\sigma\left(\beta^m_{\epsilon_i}(r)\right) = \varnothing$ but $aware(i, \beta^m_\epsilon(r)) = t$, then

  $$r_i(m+1) = \sigma(\beta^m_i(r) \sqcup \beta^m_{\epsilon_i}(r)) : r_i(m) = \sigma(\beta^m_i(r)) : r_i(m) = \sigma(\beta^m_i(r)) : r'_i(m) \quad (2.130)$$

  by the induction hypothesis. By Lemma 2.7.10(5), also

  $$aware(i, \beta^m_\epsilon(r')) = aware(i, \beta^m_{\epsilon_i}(r')) = aware(i, Y) = aware(i, \beta^m_{\epsilon_i}(r)) = aware(i, \beta^m_\epsilon(r)) = t.$$

  Thus, the last expression of (2.130) is equal to the first expression of (2.129) and this case can be concluded by continuing the series of equalities the same way as it was done in (2.129) in the preceding case.

- if $\sigma\left(\beta^m_{\epsilon_i}(r)\right) = \varnothing$ and $unaware(i, \beta^m_\epsilon(r)) = t$, then $unaware(i, \beta^m_\epsilon(r')) = t$ by the same reasoning we just applied to $aware(i, \beta^m_\epsilon(r'))$. In addition, by (2.23), $passive(i, \beta^m_\epsilon(r))$, meaning that $\beta^m_i(r) = \varnothing$ by (2.25). Finally,

$$\sigma\left(\beta^m_{\epsilon_i}(r')\right) = \sigma(Y) = \sigma\left(\beta^m_i(r) \sqcup \beta^m_{\epsilon_i}(r)\right) = \sigma\left(\beta^m_{\epsilon_i}(r)\right) = \varnothing.$$

  Thus, in this case,

$$r'_i(m+1) = r'_i(m) = r_i(m) = r_i(m+1).$$

This completes the proof of the induction step for Property 2.

For Property 3, the induction step is even simpler. Since $\beta^m_j(r') = \varnothing$ and $\beta^m_{\epsilon_j}(r') \subset \{fail(j)\}$ for any $j \neq i$, it follows that $\sigma(\beta^m_{\epsilon_j}(r')) = \varnothing$ and $unaware(j, \beta^m_\epsilon(r')) = unaware(j, \beta^m_{\epsilon_j}(r')) = t$, it follows that

$$r'_j(m+1) = r'_j(m) = r'_j(0)$$

by the induction hypothesis. □

Remark 2.7.17. The previous lemma states that for a designated agent $i \in \mathcal{A}$ at some local state $r_i(t)$ there is always an i-indistinguishable local state $r'_i(t)$ in an alternative transitional run r' such that all other agents are yet to leave their initial local states, with i definitely Byzantine while other agents can be made Byzantine or correct at will. We call this the Brain-in-the-Vat Lemma because agent i attains this indistinguishable local state by imagining that all actions and events from the original run happened to it without any participation of other agents.

Definition 2.7.18. For an adjustment $adj = [B_t; \ldots; B_0]$ of extent t satisfying (2.116), we denote

$$Failed(adj) := \left\{ j \mid (\exists m \leq t)\, \mathfrak{e}\rho^m_j \cap (BEvents_j \sqcup \{sleep(j), hibernate(j)\}) \neq \varnothing \right\}$$

the set of agents who are assigned Byzantine events, including $sleep(i)$ or $hibernate(i)$ instructions by this adjustment.

Corollary 2.7.19. *For the adjustment adj used in Lemma 2.7.16,*

$$Failed(adj) = \{i\} \cup \{j \neq i \mid (\exists m \leq t-1)\, \rho^m_j = BFreeze_j\}.$$

Hence, the number $|Failed(adj)| \geq 1$ of agents necessarily failed by this adjustment is always positive.

Corollary 2.7.20. *Lemma 2.7.16 also holds if r is a $\tau^{B_f}_{P_\epsilon,P}$-transitional run for any*

$$f \geq |Failed(adj)|$$

(in particular, if all $\rho^m_j = CFreeze$ for $j \neq i$, it is sufficient to have $f \geq 1$). Moreover, in this case Property 1 can be replaced by

1′. *r' is $\tau^{B_f}_{P_\epsilon,P}$-transitional.*

Proof. The construction is exactly the same. The additional filtering of too many Byzantine agents introduced in a round will not be used (until timestamp t) because the constructed part of the new run only fails allowable number of agents, including agent i. □

Corollary 2.7.21. *For an agent $i \in \mathcal{A}$, for a set $BD \subset \mathcal{A} \setminus \{i\}$ of agents, for a non-excluding agent-context $\chi = ((P_\epsilon, \mathscr{G}(0), \tau^{B_f}, \Psi), P)$ such that $f \geq 1 + |BD|$ and P_ϵ makes i gullible, all other agents $j \neq i$ delayable, and additionally all agents from BD fallible, for a χ-based interpreted system $I = (R^\chi, \pi)$, for a run $r \in R^\chi$, and for a timestamp $t > 0$, there is a run $r' \in R^\chi$ that satisfies Properties 2–8 from Lemma 2.7.16 and such that $\mathcal{A}(Failed(r'(t))) = \{i\} \cup BD$.*

Proof. Consider an tadjustment $adj = [B_{t-1}; \ldots; B_0]$ with (2.116) such that for all $m < t$,

$$\rho_i^m = PFake_i,$$
$$\rho_j^m = \begin{cases} BFreeze_j & \text{if } j \in BD, \\ CFreeze & \text{if } j \in \mathcal{A} \setminus (\{i\} \sqcup BD). \end{cases}$$

Clearly, $|Failed(adj)| = 1 + |BD|$. By Cor. 2.7.20, there exists a τ^{B_f}-transitional run r'' satisfying all the required conditions. Clearly, $r'' \in R^{w\chi}$ because, by (2.117) we have $r''(0) = r(0) \in \mathscr{G}(0)$. It remains to note that the the initial prefix of r'' up to timestamp t can be extended to a run $r' \in R^\chi$ because χ is non-excluding. Since all required properties depend only on this initial prefix, they are also satisfied for r'. □

Corollary 2.7.22. *For an agent $i \in \mathcal{A}$, a set $BD \subset \mathcal{A} \setminus \{i\}$ of agents, a fully f-Byzantine agent-context $\chi = ((P_\epsilon, \mathscr{G}(0), \tau^{B_f}, FS), P)$ with $f \geq 1 + |BD|$, a χ-based interpreted system $I = (R^\chi, \pi)$, a run $r \in R^\chi$, and a timestamp $t > 0$, there is a run $r' \in R^\chi$ that satisfies Properties 2–8 from Lemma 2.7.16.*

Proof. f-fully Byzantine agent-contexts are non-excluding by Lemma 2.6.18. Since all agents in such agent-contexts are fully Byzantine, they are gullible by definition. In addition to i being gullible, it follows from Cor. 2.6.16 that all other $j \neq i$ are delayable. Finally, since fully Byzantine agents are error-prone by definition, it follows from the same Cor. 2.6.16 that all $j \in BD$ are fallible. □

Corollary 2.7.23. *For the χ-based interpreted system $I = (R^\chi, \pi)$ and run r' from Cor. 2.7.21 or Cor. 2.7.22 with $BD = \varnothing$ and, accordingly, with $f \geq 1$, for any timestamp $t \in \mathbb{N}$,*

$$(\forall o \in \overline{Actions} \sqcup \overline{Events})\,(\forall t' \leq t)\,(I, r', t') \not\models \overline{occurred}(o). \tag{2.131}$$

Proof. Recall that by (2.84) and (2.83)

$$\overline{occurred}(o) = \bigvee_{j \in \mathcal{A}} \overline{occurred}_j(o).$$

Thus, we need to show that

$$(I, r', t') \not\models \overline{occurred}_j(o)$$

for each $j \in \mathcal{A}$, each $t' \leq t$, and each $o \in \overline{Actions} \sqcup \overline{Events}$. By (2.81), we need to show that

$$(\forall t'' < t')\, o \notin label^{-1}\left(\beta_j^{t''}(r') \sqcup \overline{\beta}_{\epsilon_j}^{t''}(r')\right).$$

Since all such $t'' < t$, it is sufficient to show

$$(\forall t'' < t)\, o \notin label^{-1}\left(\beta_j^{t''}(r') \sqcup \overline{\beta}_{\epsilon_j}^{t''}(r')\right).$$

For $t = 0$, this is vacuously true. For $t > 0$, we can use Cor. 2.7.21 or Cor. 2.7.22 respectively. Then this statement follows from the fact that for all agents $j \in \mathcal{A}$,

$$\beta_j^{t''}(r') = \overline{\beta}_{\epsilon_j}^{t''}(r') = \varnothing.$$

Regarding $\beta_j^{t''}(r')$, this follows from Lemma 2.7.16(8). Regarding $\overline{\beta}_{\epsilon_j}^{t''}(r')$ for $j \neq i$, it follows from Lemma 2.7.16(6). Finally, regarding $\overline{\beta}_{\epsilon_i}^{t''}(r')$ for agent i it follows from Lemma 2.7.16(7) and $\overline{\beta}_{\epsilon_i}^{t''}(r') \subset \beta_{\epsilon_i}^{t''}(r') \setminus \beta_{f_i}^{t''}(r')$. □

2.8 Introspection

By introspection we understand the ability of agents to reason about their own state and their own knowledge. The agent is primarily interested in its own correctness and the reliability of data it receives.

2.8.1 Local Introspection

We will now use the brain-in-the-vat construction to show that fully Byzantine agents can never be sure that a particular action/event definitively took place or, barring the initial state, that they are correct.

Lemma 2.8.1. *For an agent $i \in \mathcal{A}$, for a non-excluding agent-context $\chi = ((P_\epsilon, \mathscr{G}(0), \tau^{B_f}, \Psi), P)$ such that $f \geq 1$ and P_ϵ makes i gullible and all other agents $j \neq i$ delayable, for a χ-based interpreted system $I = (R^\chi, \pi)$, for a run $r \in R^\chi$, for a timestamp $t \in \mathbb{N}$, and for an action or event $o \in \overline{Actions} \sqcup \overline{Events}$,*

$$(I, r, t) \not\models K_i \overline{occurred}(o).$$

In particular, this statement holds for fully f-Byzantine agent-contexts.

Proof. For $t = 0$, the statement is obvious. For $t > 0$, the statement follows directly from (2.131) of Cor. 2.7.23 and Lemma 2.7.16(2). □

Lemma 2.8.2. *For an agent $i \in \mathcal{A}$, for a non-excluding agent-context $\chi = ((P_\epsilon, \mathscr{G}(0), \tau^{B_f}, \Psi), P)$ such that $f \geq 1$ and P_ϵ makes i gullible and all other agents $j \neq i$ delayable, for a χ-based interpreted system $I = (R^\chi, \pi)$, for a run $r \in R^\chi$, and for a timestamp $t > 0$,*

$$(I, r, t) \not\models K_i correct_i.$$

In particular, this statement holds for fully f-Byzantine agent-contexts.

Proof. Consider the run r' constructed in Cor. 2.7.21 (respectively, Cor. 2.7.22) for $BD = \varnothing$. Recall that by (2.78)

$$(I, r', t) \models correct_i \qquad \Longleftrightarrow \qquad (i, t) \notin Failed(r', t).$$

Thus, the statement follows directly from Properties 2 and 4 of Lemma 2.7.16. □

Remark 2.8.3. It is clear that within any agent-context based on a transition template τ^{B_0}, i.e., for $f = 0$, all agents always know that they are correct and they can learn about real actions/events from observation because no Byzantine events can ever happen in such runs. Conversely, such agents can never learn that they are faulty because it will never be true.

Unlike the knowledge of own correctness, it is in principle possible for a fully Byzantine agent to learn of its own defectiveness (provided, as just noted, that mistakes are, in fact, allowed).

Lemma 2.8.4. *For some agent $i \in \mathcal{A}$, an agent-context $\chi = ((P_\epsilon, \mathscr{G}(0), \tau^{B_f}, \Psi), P)$ such that $f \geq 1$, it is possible that for some (weakly) χ-based interpreted system I, some (weakly) χ-consistent run r, and some timestamp $t > 0$,*

$$(I, r, t) \models K_i faulty_i.$$

Proof. This happens whenever there is a mismatch between actions recorded in the agent's local history and actions prescribed by the agent's protocol for the preceding local state. □

2.8.2 Global Introspection

Similarly to the virtual impossibility for the agent to ascertain its own correctness, it is similarly almost impossible for an agent to learn the Byzantine status of another agent.

Lemma 2.8.5. *For some agents $i \neq j$, for a non-excluding agent-context $\chi = ((P_\epsilon, \mathscr{G}(0), \tau^{B_f}, \Psi), P)$ such that $f \geq 1$ and P_ϵ makes agent i gullible and all other agents delayable, for a χ-based interpreted system $I = (R^\chi, \pi)$, for a run $r \in R^\chi$, and for a timestamp $t \in \mathbb{N}$,*

$$(I, r, t) \not\models K_i faulty_j.$$

In particular, this statement holds for fully f-Byzantine agent-contexts.

Proof. For $t = 0$, the statement is obvious because no agent can be faulty in the initial state. For $t > 0$, the statement follows directly from Properties 2 and 6 of Lemma 2.7.16 applied to $\rho_j^m = CFreeze$ for all $j \neq i$. □

Lemma 2.8.6. *For agents $i \neq j$, for a non-excluding agent-context $\chi = ((P_\epsilon, \mathscr{G}(0), \tau^{B_f}, \Psi), P)$ such that $f \geq 2$ and P_ϵ makes agent i gullible, agent j delayable and fallible, and all other agents delayable, for a χ-based interpreted system $I = (R^\chi, \pi)$, for a run $r \in R^\chi$, and for a timestamp $t > 0$,*

$$(I, r, t) \not\models K_i correct_j.$$

In particular, this statement holds for fully f-Byzantine agent-contexts.

Proof. The statement follows directly from Properties 2 and 5 of Lemma 2.7.16 applied to

$$\rho_k^m = \begin{cases} CFreeze & \text{if } k \notin \{i, j\} \\ BFreeze_j & \text{if } k = j. \end{cases}$$

It is also sufficient to set $\rho_j^0 = BFreeze_j$ and all other $\rho_k^m = CFreeze$ for $k \neq i$. □

In this proof, $BD = \{j\}$ and $Failed\,(adj) = \{i, j\}$, which is why it was necessary to allow at least two agents to become Byzantine.

Given that typical Byzantine agents can never be sure that they are correct, or that another agent is correct (faulty), or that a particular action/event happened, their behavior cannot rely on knowledge but should be governed by a weaker epistemic state. We define the following operators:

$$B_i\varphi := K_i(correct_i \to \varphi) \tag{2.132}$$
$$H_i\varphi := correct_i \to B_i\varphi = correct_i \to K_i(correct_i \to \varphi). \tag{2.133}$$

Thus, a belief in φ means that φ follows from the local state of the agent provided the agent is correct, whereas hope that φ states that this belief need only hold if the agent is correct.

Lemma 2.8.7 (Properties of belief and hope)**.** *For any formula φ, any agent i, the following formulas are propositional tautologies and, hence, are valid in every interpreted system.*

- $\models B_i\varphi \to H_i\varphi$
- $\models correct_i \to (B_i\varphi \leftrightarrow H_i\varphi)$
- $\models faulty_i \to H_i\varphi$

Proof. The first two statements are obvious. The last statement follows from the definition of $faulty_i$ as $\neg correct_i$. □

As can be seen from the preceding lemma, belief is stronger than hope in general, but equivalent to it for correct agents. Hope provides no information for faulty agents, whereas belief can, which can be used for designing algorithms for malfunctioning agents. In fact, given that

$$\models (correct_i \to faulty_i) \leftrightarrow faulty_i$$

is also a propositional tautology, it is the case that

$$\models B_i faulty_i \leftrightarrow K_i faulty_i$$

due to the normality of the K_i modality. In other words, in order to react to its own faults, the agent should know of them, which can be formulated in terms of the belief modality. On the other hand, $\not\models H_i faulty_i \to K_i faulty_i$ because a faulty agent always hopes to be faulty but may not know it is.

On the other hand, the hope modality will be technically convenient in future proofs by affording more elegant formulations for iterated modalities.

CHAPTER 3

The Extension Framework

This chapter provides the core contribution of this thesis, a generic *extension framework*, which allows to specify and safely combine extensions. Such framework extensions are restrictions of the general framework, which are formally defined as follows:

Definition 3.0.1 (Extension)**.** For a set of pairs of environment and joint protocols $PP^\alpha \subseteq \mathscr{C}_\epsilon \times \mathscr{C}$, a transition template τ^α, an admissibility condition Ψ^α, and a set of sets of global initial states IS^α, such that $PP^\alpha \neq \varnothing$, $IS^\alpha \neq \varnothing$, $\Psi^\alpha \neq \varnothing$ we define

$$\mathscr{E}^\alpha := (PP^\alpha, IS^\alpha, \tau^\alpha, \Psi^\alpha). \tag{3.1}$$

Further we say an agent-context $\chi = \big((P_\epsilon, \mathscr{G}(0), \tau, \Psi), P\big)$ is part of $\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^\alpha, \Psi^\alpha)$, denoted $\chi \in \mathscr{E}^\alpha$ iff

1. $(P_\epsilon, P) \in PP^\alpha$
2. $\mathscr{G}(0) \in IS^\alpha$
3. $\tau = \tau^\alpha$
4. $\Psi = \Psi^\alpha$
5. $R^\chi \neq \varnothing$.

Finally we call $\mathscr{E}^\alpha$ a *framework extension* or just an *extension* iff there exists an agent context χ such that $\chi \in \mathscr{E}^\alpha$.

Now, we extend the notion of non-excluding agent-contexts from Definition 2.3.28 to extensions as follows:

Definition 3.0.2. For an extension $\mathscr{E}$, we say that $\mathscr{E}$ is non-excluding iff $\forall \chi \in \mathscr{E}$, χ is non-excluding.

3.1 Filter Combination

In order to merge two extensions into a third one, we first have to define the intersection of its constituent parts. For two sets of sets of initial states, admissibility conditions (which are sets of runs) as for sets of pairs of environment and joint protocols we can simply use conventional set intersection. However for transition templates the notion of intersection still has to be defined.

Definition 3.1.1. For any agent $i \in \mathcal{A}$, global history $h \in \mathscr{G}$, sets $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ we define the *neutral* event and action filters (the weakest filters) as

$$filter_\epsilon^N(h, X_\epsilon, X_1, \ldots, X_n) := X_\epsilon \tag{3.2}$$

and

$$filter_i^N(X_1, \ldots, X_n, X_\epsilon) := X_i. \tag{3.3}$$

The transition template, where only the neutral filters are used we denote by τ^N.

The only part that distinguishes different transition templates are the filter functions. Hence we define different combinations of filter functions as follows (note that (3.4) we actually repeat (2.92)):

Definition 3.1.2. Given two event filter functions $filter_\epsilon^\alpha$ of τ^α and $filter_\epsilon^\beta$ of τ^β for some $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ we define

- *filter composition* as

$$\begin{aligned} &filter_\epsilon^{\beta\circ\alpha}(h, X_\epsilon, X_1, \ldots, X_n) := \\ &filter_\epsilon^\beta(h, filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n) \end{aligned} \tag{3.4}$$

- *filter intersection* as

$$\begin{aligned} &filter_\epsilon^{\alpha+\beta}(h, X_\epsilon, X_1, \ldots, X_n) := \\ &filter_\epsilon^{\beta\circ\alpha}(h, X_\epsilon, X_1, \ldots, X_n) \cap filter_\epsilon^{\alpha\circ\beta}(h, X_\epsilon, X_1, \ldots, X_n) \end{aligned} \tag{3.5}$$

- *k-filter intersection* for $k \in \mathbb{N} \setminus \{0\}$ as

$$\begin{aligned} &filter_\epsilon^{k\cdot(\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) := \\ &filter_\epsilon^{\alpha+\beta}\left(h, \underbrace{\left(\ldots\left(filter_\epsilon^{\alpha+\beta}(h, X_\epsilon, X_1, \ldots, X_n)\right)\ldots\right)}_{k-1 \text{ times}}, X_1, \ldots, X_n\right) \end{aligned} \tag{3.6}$$

- 0-*filter intersection* as

$$filter_\epsilon^{0\cdot(\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) := filter_\epsilon^N(h, X_\epsilon, X_1, \ldots, X_n) \tag{3.7}$$

- *fixpoint filter intersection* as

$$filter_\epsilon^{\alpha*\beta}(h, X_\epsilon, X_1, \ldots, X_n) := \lim_{k\to\infty} filter_\epsilon^{k\cdot(\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n). \tag{3.8}$$

We do the same for action filters.

Definition 3.1.3. Given two action filter functions (for $i \in \mathcal{A}$) $filter_i^\alpha$ of τ^α and $filter_i^\beta$ of τ^β for sets $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ we define

- *filter composition* as

$$\begin{aligned} &filter_i^{\beta\circ\alpha}(X_1, \ldots, X_n, X_\epsilon) := \\ &filter_i^\beta(X_1, \ldots, X_{i-1}, filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon,), X_{i+1}, \ldots, X_n, X_\epsilon) \end{aligned} \tag{3.9}$$

- *filter intersection*

$$\begin{aligned} &filter_i^{\alpha+\beta}(X_1, \ldots, X_n, X_\epsilon) := \\ &filter_i^{\beta\circ\alpha}(X_1, \ldots, X_n, X_\epsilon) \cap filter_i^{\alpha\circ\beta}(X_1, \ldots, X_n, X_\epsilon) \end{aligned} \tag{3.10}$$

- *k-filter intersection* for $k \in \mathbb{N} \setminus \{0\}$ as

$$\begin{aligned} &filter_i^{k\cdot(\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon) := \\ &filter_i^{\alpha+\beta}\left(X_1, \ldots, X_{i-1}, \underbrace{\left(\ldots\left(filter_i^{\alpha+\beta}(X_1, \ldots, X_n, X_\epsilon)\right)\ldots\right)}_{k-1 \text{ times}}, X_{i+1}, \ldots, X_n, X_\epsilon\right) \end{aligned} \tag{3.11}$$

- 0-*filter intersection* as

$$filter_i^{0\cdot(\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon) := filter_i^N(X_1, \ldots, X_n, X_\epsilon) \tag{3.12}$$

- *fixpoint filter intersection* as

$$filter_i^{\alpha*\beta}(X_1, \ldots, X_n, X_\epsilon) := \lim_{k\to\infty} filter_i^{k\cdot(\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon) \tag{3.13}$$

The reason for not just using the intersection of two filters in Definitions 3.1.2, 3.1.3 without filter composition, is that this would neglect any interaction between the filters with respect to events happening in the current round. The combination of the at-most-f-Byzantine-agents-filter (2.91) together with the Byzantine event filter (2.24) is a good example why filter composition (and possibly filter intersection) is necessary.

In the worst case there could be a circular dependency between the filter functions, which can then only be resolved by a fixpoint computation (3.8), (3.13).

Definition 3.1.4 (Repeated Composition)**.** To simplify notation and stay consistent with (3.6), (3.7) and (3.11), (3.12) for some $k \in \mathbb{N} \setminus \{0\}$ we define for arbitrary filters $filter_\epsilon^\alpha$, $filter_i^\alpha$ for $i \in \mathcal{A}$

$$\begin{aligned} &filter_\epsilon^{k\cdot\alpha}(h, X_\epsilon, X_1, \ldots, X_n) := \\ &filter_\epsilon^\alpha\left(\underbrace{h, \ldots\left(filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n)\right)\ldots}_{k-1 \text{ times}}, X_1, \ldots, X_n\right), \end{aligned} \tag{3.14}$$

$$filter_\epsilon^{0\cdot\alpha}(h, X_\epsilon, X_1, \ldots, X_n) := filter_\epsilon^N(h, X_\epsilon, X_1, \ldots, X_n) \tag{3.15}$$

and similarly

$$filter_i^{k\cdot\alpha}(X_1, \ldots, X_n, X_\epsilon) := filter_i^\alpha\left(X_1, \ldots, X_{i-1}, \underbrace{\ldots\left(filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon)\right)\ldots}_{k-1 \text{ times}}, X_{i+1}, \ldots, X_n, X_\epsilon\right), \tag{3.16}$$

$$filter_i^{0\cdot\alpha}(X_1, \ldots, X_n, X_\epsilon) := filter_i^N(X_1, \ldots, X_n, X_\epsilon). \tag{3.17}$$

Definition 3.1.5. Considering Definitions 3.1.2 and 3.1.3, we define

- $\tau^{\alpha\circ\beta}$ as the resulting transition template with the combined filter functions from (3.4) and (3.9).
- $\tau^{\alpha+\beta}$ as the resulting transition template with the combined filter functions from (3.5) and (3.10).
- $\tau^{k\cdot(\alpha+\beta)}$ as the resulting transition template with the combined filter functions from (3.6) or (3.7) and (3.11) or (3.12) (depending on k).
- $\tau^{\alpha*\beta}$ as the resulting transition template with the combined filter functions from (3.8) and (3.13).

Definition 3.1.6. We define PD_ϵ^{t-coh} as *the (downward closed) domain of all t-coherent events.*

$$PD_\epsilon^{t-coh} := \{X_\epsilon \in 2^{GEvents} \mid X_\epsilon \text{ is } t\text{-coherent for some } t \in \mathbb{N}\} \tag{3.18}$$

3.1.1 Basic Filter Property

To make reasoning about filters more concise, we formalize the following property, which we expect to hold true for all filter functions. Hence, whenever we refer to a function as filter and write $filter_\epsilon$ or $filter_i$ (for some agent $i \in \mathcal{A}$), we assume them to satisfy the following:

Definition 3.1.7 (Basic Filter Property)**.** We call a function $filter_\epsilon^\alpha$ (in accordance with Definition 2.2.19) an *event filter function* iff for all $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$

$$filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) \subseteq X_\epsilon. \tag{3.19}$$

Similarly we call a function $filter_i^\alpha$ (in accordance with Definition 2.2.19) for some $i \in \mathcal{A}$ an *action filter function* iff for all $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$

$$filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon) \subseteq X_i. \tag{3.20}$$

Corollary 3.1.8. *For two arbitrary filter functions $filter_\epsilon^\gamma$ and $filter_i^\gamma$ for some $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ and some $k \in \mathbb{N}$ it holds that*

$$\begin{aligned} filter_\epsilon^{(k+1)\cdot\gamma}(h, X_\epsilon, X_1, \ldots, X_n) &\subseteq filter_\epsilon^{k\cdot\gamma}(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{(k+1)\cdot\gamma}(X_1, \ldots, X_n, X_\epsilon) &\subseteq filter_i^{k\cdot\gamma}(X_1, \ldots, X_n, X_\epsilon). \end{aligned} \tag{3.21}$$

Proof. The statement immediately follows from Definitions 3.1.4 and 3.1.7. □

Lemma 3.1.9. *For two arbitrary filter functions* $filter_\epsilon^\gamma$ *and* $filter_i^\gamma$ *for some* $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ *and some* $k, k' \in \mathbb{N}$, *where* $k \geq k'$, *it holds that*

$$\begin{aligned} filter_\epsilon^{k\cdot\gamma}(h, X_\epsilon, X_1, \ldots, X_n) &\subseteq filter_\epsilon^{k'\cdot\gamma}(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{k\cdot\gamma}(X_1, \ldots, X_n, X_\epsilon) &\subseteq filter_i^{k'\cdot\gamma}(X_1, \ldots, X_n, X_\epsilon). \end{aligned} \tag{3.22}$$

Proof. By induction over $k - k'$.
Induction Hypothesis: For $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ and some $k, k' \in \mathbb{N}$, where $k \geq k'$, it holds that

$$\begin{aligned} filter_\epsilon^{k\cdot\gamma}(h, X_\epsilon, X_1, \ldots, X_n) &\subseteq filter_\epsilon^{k'\cdot\gamma}(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{k\cdot\gamma}(X_1, \ldots, X_n, X_\epsilon) &\subseteq filter_i^{k'\cdot\gamma}(X_1, \ldots, X_n, X_\epsilon). \end{aligned} \tag{3.23}$$

Base Case: If $k - k' = 0$, the statement trivially holds.
Induction Step: Suppose the induction hypothesis (3.23) holds for some $k \geq k'$, applying Corollary 3.1.8 gives us

$$\begin{aligned} &filter_\epsilon^{(k+1)\cdot\gamma}(h, X_\epsilon, X_1, \ldots, X_n) \subseteq filter_\epsilon^{k\cdot\gamma}(h, X_\epsilon, X_1, \ldots, X_n) \subseteq \text{(by the induction hypothesis (3.23))} \\ &filter_\epsilon^{k'\cdot\gamma}(h, X_\epsilon, X_1, \ldots, X_n) \\ &\text{or similarly for action filters} \\ &filter_i^{(k+1)\cdot\gamma}(X_1, \ldots, X_n, X_\epsilon) \subseteq filter_i^{k\cdot\gamma}(X_1, \ldots, X_n, X_\epsilon) \subseteq \text{(by the induction hypothesis (3.23))} \\ &filter_i^{k'\cdot\gamma}(X_1, \ldots, X_n, X_\epsilon), \end{aligned} \tag{3.24}$$

from which the statement follows by transitivity of the subset relation $\subseteq$. □

Lemma 3.1.10. $filter_\epsilon^{\leq f}$ *satisfies the basic filter property (Definition 3.1.7), thus is an event filter function.*

Proof. Follows directly from (2.91) of its definition. □

3.1.2 Monotonicity

Definition 3.1.11. We say an event filter $filter_\epsilon^\alpha$ is *monotonic* for a downward closed domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$ iff for every $h \in \mathscr{G}$, sets $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$ it holds that

$$\begin{aligned} &(\forall \widetilde{X_\epsilon} \subseteq X_\epsilon)(\forall \widetilde{X_1} \subseteq X_1) \ldots (\forall \widetilde{X_n} \subseteq X_n) \\ &filter_\epsilon^\alpha\left(h, \widetilde{X_\epsilon}, \widetilde{X_1}, \ldots, \widetilde{X_n}\right) \subseteq filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n). \end{aligned} \tag{3.25}$$

Definition 3.1.12. We say an event filter $filter_\epsilon^\alpha$ is *simply monotonic* for a downward closed domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$ iff for some $h \in \mathscr{G}$, sets $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$ it holds that

$$(\forall \widetilde{X_\epsilon} \subseteq X_\epsilon)\ filter_\epsilon^\alpha\left(h, \widetilde{X_\epsilon}, X_1, \ldots, X_n\right) \subseteq filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n). \tag{3.26}$$

Definition 3.1.13. Similarly we say an action filter $filter_i^\alpha$ for $i \in \mathcal{A}$ is *monotonic* for a downward closed domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$ iff for any sets $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$ it holds that

$$\begin{aligned}(\forall \widetilde{X_\epsilon} \subseteq X_\epsilon)(\forall \widetilde{X_1} \subseteq X_1) \dots (\forall \widetilde{X_n} \subseteq X_n) \\ filter_i^\alpha \left(\widetilde{X_1}, \dots, \widetilde{X_n}, \widetilde{X_\epsilon}\right) \subseteq filter_i^\alpha \left(X_1, \dots, X_n, X_\epsilon\right).\end{aligned} \tag{3.27}$$

Definition 3.1.14. We say an action filter $filter_i^\alpha$ for $i \in \mathcal{A}$ is *simply monotonic* for a downward closed domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$ iff for any sets $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$ it holds that

$$(\forall \widetilde{X_i} \subseteq X_i)\ filter_i^\alpha \left(X_1, \dots, X_{i-1}, \widetilde{X_i}, X_{i+1}, \dots, X_n, X_\epsilon\right) \subseteq filter_i^\alpha \left(X_1, \dots, X_n, X_\epsilon\right). \tag{3.28}$$

Lemma 3.1.15. *If an event (or action) filter $filter_\epsilon^\alpha$ ($filter_i^\alpha$ for $i \in \mathcal{A}$) is monotonic for some downward closed domain, then $filter_\epsilon^\alpha$ ($filter_i^\alpha$) is simply monotonic in this domain as well.*

Proof. The statement directly follows from the definitions of (simple) monotonicity (3.25), (3.26), (3.27) and (3.28). □

Lemma 3.1.16. *For every global history $h \in \mathcal{G}$ the neutral filter functions $filter_\epsilon^N$ and $filter_i^N$ for any $i \in \mathcal{A}$ are monotonic in the domain $2^{GEvents}$, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$.*

Proof. By Definition 3.1.1 of the neutral filter functions it holds that for all $i \in \mathcal{A}$, $h \in \mathcal{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions_1}$, ..., $X_n \subseteq \overline{GActions_n}$, $X'_\epsilon \subseteq X_\epsilon$, $X'_1 \subseteq X_1$, ..., $X'_n \subseteq X_n$

$$filter_\epsilon^N (h, X'_\epsilon, X'_1, \dots, X'_n) = X'_\epsilon \subseteq X_\epsilon = filter_\epsilon^N (h, X_\epsilon, X_1, \dots, X_n) \tag{3.29}$$

and

$$filter_i^N (X'_1, \dots, X'_n, X'_\epsilon) = X'_i \subseteq X_i = filter_i^N (X_1, \dots, X_n, X_\epsilon). \tag{3.30}$$

Hence the statement follows. □

Lemma 3.1.17. *For any global history $h \in \mathcal{G}$ and sets $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \subseteq \overline{GActions_1}$, ..., $X_n \subseteq \overline{GActions_n}$ the filter functions of the Byzantine transition template τ^B are monotonic.*

Proof. We conduct the following proofs by induction. First we examine the Byzantine event filter function. Suppose $O_1, \dots, O_K$ is an arbitrary ordering of the events and actions in $X_\epsilon \sqcup X_1 \sqcup \dots \sqcup X_n$, where $K = |X_\epsilon \sqcup X_1 \sqcup \dots \sqcup X_n|$, let $X_\epsilon^0 = X_\epsilon$, $X_1^0 = X_1$, ..., $X_n^0 = X_n$ and define for $0 \leq k \leq K-1$ and $i \in \mathcal{A}$

$$X_\epsilon^{k+1} = \begin{cases} X_\epsilon^k \setminus \{O_{k+1}\} & \text{if } O_{k+1} \in X_\epsilon \\ X_\epsilon^k & \text{otherwise} \end{cases} \tag{3.31}$$

$$X_i^{k+1} = \begin{cases} X_i^k \setminus \{O_{k+1}\} & \text{if } O_{k+1} \in X_i \\ X_i^k & \text{otherwise} \end{cases} \tag{3.32}$$

and denote

$$filter_\epsilon^B\left(h, X_\epsilon^k, X_1^k, \ldots, X_n^k\right) = Y_\epsilon^k \tag{3.33}$$

and

$$filter_\epsilon^B\left(h, X_\epsilon, X_1, \ldots, X_n\right) = Y_\epsilon. \tag{3.34}$$

Induction Hypothesis: For some $0 \leq k < K$

$$filter_\epsilon^B\left(h, X_\epsilon^k, X_1^k, \ldots, X_n^k\right) = Y_\epsilon^k \subseteq Y_\epsilon. \tag{3.35}$$

Base Case: For $k = 0$

$$filter_\epsilon^B\left(h, X_\epsilon^0, X_1^0, \ldots, X_n^0\right) = filter_\epsilon^B\left(h, X_\epsilon, X_1, \ldots, X_n\right) = Y_\epsilon. \tag{3.36}$$

Therefore $Y_\epsilon^0 \subseteq Y_\epsilon$ trivially holds.
Induction Step for $k \to k+1$:
If $O_{k+1} = E \in X_\epsilon$ is ...

- some $grecv(i, j, \mu, id)$ or $fake\left(i, grecv(i, j, \mu, id)\right)$ or $external\left(i, e\right)$ or $fake\left(i, external\left(i, e\right)\right)$ or $fake\left(i, internal\left(i, a\right) \mapsto A\right)$ or $sleep(i)$ or $hibernate(i)$ (since t-coherence of X_ϵ was assumed, by Lemma 2.2.3 X_ϵ^k is also t-coherent and there can only ever be one of the three system events present for some agent $i \in \mathcal{A}$; in particular this means after removing either $sleep(i)$ or $hibernate(i)$ there can be no $go(i)$ left) for some $i, j \in \mathcal{A}$, $e \in Ext_i$, $a \in Int_i$, $\mu \in Msgs$, $id \in \mathbb{N}$, $A \in \{🗓\} \sqcup \overline{GActions_i}$ by definition of the Byzantine event filter function (2.24) the result of the filtering is

  $$filter_\epsilon^B\left(h, X_\epsilon^{k+1}, X_1^{k+1}, \ldots, X_n^{k+1}\right) = filter_\epsilon^B\left(h, X_\epsilon^k \setminus \{E\}, X_1^k, \ldots, X_n^k\right) = Y_\epsilon^k \setminus \{E\}. \tag{3.37}$$

  By assumption of the induction hypothesis (3.35) $Y_\epsilon^k \setminus \{E\} \subseteq Y_\epsilon^k \subseteq Y_\epsilon$ holds.

- $fake\left(i, gsend(i, j, \mu, id) \mapsto A\right)$ for some $i, j \in \mathcal{A}$, $\mu \in Msgs$, $id \in \mathbb{N}$, $A \in \{🗓\} \sqcup \overline{GActions_i}$ by definition of the Byzantine event filter function (2.24) the result of the filtering is either

  $$filter_\epsilon^B\left(h, X_\epsilon^{k+1}, X_1^{k+1}, \ldots, X_n^{k+1}\right) = filter_\epsilon^B\left(h, X_\epsilon^k \setminus \{E\}, X_1^k, \ldots, X_n^k\right) = Y_\epsilon^k \setminus \{E\} \tag{3.38}$$

  if $gsend(i, j, \mu, id) \in X_i^k$ and $active(i, X_\epsilon^k) = t$, or

  $$\begin{aligned} &filter_\epsilon^B\left(h, X_\epsilon^{k+1}, X_1^{k+1}, \ldots, X_n^{k+1}\right) = \\ &filter_\epsilon^B\left(h, X_\epsilon^k \setminus \{E\}, X_1^k, \ldots, X_n^k\right) = Y_\epsilon^k \setminus \{E, grecv(j, i, \mu, id)\} \end{aligned} \tag{3.39}$$

  otherwise.

  In either case by assumption of the induction hypothesis (3.35) $Y_\epsilon^k \setminus \{E, grecv(j, i, \mu, id)\} \subseteq Y_\epsilon^k \subseteq Y_\epsilon$ and $Y_\epsilon^k \setminus \{E\} \subseteq Y_\epsilon^k \subseteq Y_\epsilon$ holds.

- $go(i)$ for $i \in \mathcal{A}$ by definition of the Byzantine event filter (2.24) we get

$$filter_\epsilon^B\left(h, X_\epsilon^{k+1}, X_1^{k+1}, \ldots, X_n^{k+1}\right) = filter_\epsilon^B\left(h, X_\epsilon^k \setminus \{E\}, X_1^k, \ldots, X_n^k\right) = Y_\epsilon^k \setminus \Big(\{E\} \sqcup \{grecv(j,i,\mu,id) \mid gsend(i,j,\mu,id) \in X_i^k \wedge (\forall A \in \{\text{📅}\} \sqcup \overline{GActions_i})\ fake\,(i, gsend(i,j,\mu,id) \mapsto A) \notin X_\epsilon^k\}\Big). \tag{3.40}$$

  By assumption of the induction hypothesis (3.35) the result of the filter is again a subset of $Y_\epsilon^k \subseteq Y_\epsilon$.

If $O_{k+1} = A \in X_i$ there are only two possibilities:

- $A = internal\,(i,a)$ for some $a \in \overline{Actions_i}$
  In this case by definition of the Byzantine event filter (2.24) it follows that

$$filter_\epsilon^B\left(h, X_\epsilon^{k+1}, X_1^{k+1}, \ldots, X_n^{k+1}\right) = filter_\epsilon^B\left(h, X_\epsilon^k, X_1^k, \ldots, X_{i-1}^k, X_i^k \setminus \{A\}, X_{i+1}^k, \ldots, X_n^k\right) = Y_\epsilon^k, \tag{3.41}$$

  from which the statement trivially follows by the induction hypothesis (3.35) $Y_\epsilon^k \subseteq Y_\epsilon$.

- $A = gsend(i,j,\mu,id)$ for some $j \in \mathcal{A}$, $\mu \in Msgs$ and $id \in \mathbb{N}$
  In this case, again by definition of the Byzantine event filter (2.24) it holds that

$$filter_\epsilon^B\left(h, X_\epsilon^{k+1}, X_1^{k+1}, \ldots, X_n^{k+1}\right) = filter_\epsilon^B\left(h, X_\epsilon^k, X_1^k, \ldots, X_{i-1}^k, X_i^k \setminus \{A\}, X_{i+1}^k, \ldots, X_n^k\right) = Y_\epsilon^k \setminus \{grecv(j,i,\mu,id) \mid (\forall A \in \{\text{📅}\} \sqcup \overline{GActions_i})\ fake\,(i, gsend(i,j,\mu,id) \mapsto A) \notin X_\epsilon^k\}. \tag{3.42}$$

  By assumption of the induction hypothesis (3.35), the filtering result is obviously a subset of $Y_\epsilon^k \subseteq Y_\epsilon$.

This completes the induction step. Since the ordering $O_1, \ldots, O_K$ was arbitrary, we conclude that the Byzantine event filter is indeed monotonic (in the domain $PD_\epsilon^{t-coh}, 2^{\overline{GActions_1}}, \ldots, 2^{\overline{GActions_n}}$).

Next we examine the Byzantine action filters (for $i \in \mathcal{A}$). Using the same arbitrary ordering $O_1, \ldots, O_K$ as before and

$$filter_i^B\,(X_1, \ldots, X_n, X_\epsilon) = Y_i, \tag{3.43}$$

we get the following induction hypothesis for every $i \in \mathcal{A}$:
Induction Hypothesis: For $0 \leq k < K$

$$filter_i^B\left(X_1^k, \ldots, X_n^k, X_\epsilon^k\right) = Y_i^k \subseteq Y_i. \tag{3.44}$$

Base Case: For $k = 0$

$$filter_i^B\left(X_1^0, \ldots, X_n^0, X_\epsilon^0\right) = Y_i^0 = filter_i^B\,(X_1, \ldots, X_n, X_\epsilon) = Y_i. \tag{3.45}$$

Induction Step for $k \to k+1$:
If $O_{k+1} = A \in X_i$ there are two possibilities:

- $j = i$
  By definition of the Byzantine action filter function (2.25) the following holds

$$\begin{aligned} &filter_i^B\left(X_1^{k+1}, \ldots, X_n^{k+1}, X_\epsilon^{k+1}\right) = \\ &filter_i^B\left(X_1^k, \ldots, X_{i-1}^k, X_i^k \setminus \{A\}, X_{i+1}^k, \ldots, X_n^k, X_\epsilon^k\right) = Y_i^k \setminus \{A\} \end{aligned} \tag{3.46}$$

  and the induction hypothesis (3.44) is again satisfied for $k+1$, as $Y_i^k \setminus \{A\} \subseteq Y_i^k \subseteq Y_i$.

- $j \neq i$
  Again by (2.25) it holds that

$$\begin{aligned} &filter_i^B\left(X_1^{k+1}, \ldots, X_n^{k+1}, X_\epsilon^{k+1}\right) = \\ &filter_i^B\left(X_1^k, \ldots, X_{j-1}^k, X_j^k \setminus \{A\}, X_{j+1}^k, \ldots, X_n^k, X_\epsilon^k\right) = Y_i^k. \end{aligned} \tag{3.47}$$

  In this case the induction hypothesis (3.44) for $k+1$ trivially holds.

Suppose $O_{k+1} = E \in X_\epsilon$. By (2.25) there are two possibilities:

- $E = go(i)$
  It follows that

$$filter_i^B\left(X_1^{k+1}, \ldots, X_n^{k+1}, X_\epsilon^{k+1}\right) = filter_i^B\left(X_1^k, \ldots, X_n^k, X_\epsilon^k \setminus \{E\}\right) = \varnothing. \tag{3.48}$$

  Since $\varnothing \subseteq Y_i$ the induction hypothesis (3.44) for $k+1$ is again satisfied.

- $E \neq go(i)$
  Since t-coherence of X_ϵ was assumed and t-coherent sets are downward closed, it cannot happen that after the removal of $sleep\,(i)$ or $hibernate\,(i)$ a $go(i)$ event remains in X'_ϵ. Therefore in this case it holds that

$$filter_i^B\left(X_1^{k+1}, \ldots, X_n^{k+1}, X_\epsilon^{k+1}\right) = filter_i^B\left(X_1^k, \ldots, X_n^k, X_\epsilon^k \setminus \{E\}\right) = Y_i^k \subseteq Y_i, \tag{3.49}$$

  which trivially satisfies the induction hypothesis (3.44).

This completes the induction step for every $i \in \mathcal{A}$. Since the ordering $O_1, \ldots, O_K$ is arbitrary, we can safely conclude that the Byzantine action filter is monotonic as well (in the domain PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}, \ldots, 2^{\overline{GActions_n}}$). □

Lemma 3.1.18. *The filter function $filter_\epsilon^{\leq 0}$ is monotonic for the domain PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$.*

Proof. By induction:
For any $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \in 2^{\overline{GActions_1}}$, ..., $X_n \in 2^{\overline{GActions_n}}$ similar as in the proof for Lemma 3.1.17 suppose $O_1, \ldots, O_K$ is an arbitrary ordering of the events and actions in

$X_\epsilon \sqcup X_1 \sqcup \ldots \sqcup X_n$, where $K = |X_\epsilon \sqcup X_1 \sqcup \ldots \sqcup X_n|$, let $X_\epsilon^0 = X_\epsilon,\ X_1^0 = X_1,\ \ldots,\ X_n^0 = X_n$ and define for $0 \leq k \leq K-1$ and $i \in \mathcal{A}$

$$X_\epsilon^{k+1} = \begin{cases} X_\epsilon^k \setminus \{O_{k+1}\} & \text{if } O_{k+1} \in X_\epsilon \\ X_\epsilon^k & \text{otherwise} \end{cases} \tag{3.50}$$

$$X_i^{k+1} = \begin{cases} X_i^k \setminus \{O_{k+1}\} & \text{if } O_{k+1} \in X_i \\ X_i^k & \text{otherwise} \end{cases} \tag{3.51}$$

and denote

$$filter_\epsilon^{\leq 0}\left(h, X_\epsilon^k,\ X_1^k,\ \ldots,\ X_n^k\right) = Y_\epsilon^k \tag{3.52}$$

and

$$filter_\epsilon^{\leq 0}\left(h, X_\epsilon,\ X_1,\ \ldots,\ X_n\right) = Y_\epsilon. \tag{3.53}$$

Induction Hypothesis: For some $0 \leq k < K$

$$filter_\epsilon^{\leq 0}\left(h, X_\epsilon^k, X_1^k, \ldots, X_n^k\right) = Y_\epsilon^k \subseteq Y_\epsilon. \tag{3.54}$$

Base Case: For $k = 0$

$$filter_\epsilon^{\leq 0}\left(h, X_\epsilon^0, X_1^0, \ldots, X_n^0\right) = filter_\epsilon^{\leq 0}\left(h, X_\epsilon, X_1, \ldots, X_n\right) = Y_\epsilon. \tag{3.55}$$

Thus $Y_\epsilon^0 \subseteq Y_\epsilon$ is trivially satisfied.

Induction Step for $k \to k+1$: If $O_{k+1} = E \in X_\epsilon$

$$\begin{aligned}
&filter_\epsilon^{\leq 0}\left(h, X_\epsilon^{k+1}, X_1^{k+1}, \ldots, X_n^{k+1}\right) = \\
&filter_\epsilon^{\leq 0}\left(h, X_\epsilon^k \setminus \{E\}, X_1^k, \ldots, X_n^k\right) = \\
&\left(X_\epsilon^k \setminus \{E\}\right) \setminus \left(\left(X_\epsilon^k \setminus \{E\}\right) \cap \left(BEvents \sqcup \bigcup_{i \in \mathcal{A}} \{sleep\,(i), hibernate\,(i)\}\right)\right) = \\
&\left(X_\epsilon^k \setminus \{E\}\right) \setminus \left(X_\epsilon^k \cap \left(BEvents \sqcup \bigcup_{i \in \mathcal{A}} \{sleep\,(i), hibernate\,(i)\}\right)\right) \subseteq \\
&X_\epsilon^k \setminus \left(X_\epsilon^k \cap \left(BEvents \sqcup \bigcup_{i \in \mathcal{A}} \{sleep\,(i), hibernate\,(i)\}\right)\right) = Y_\epsilon^k.
\end{aligned} \tag{3.56}$$

Therefore by assumption of the induction hypothesis (3.54) we conclude $Y_\epsilon^{k+1} \subseteq Y_\epsilon^k \subseteq Y_\epsilon$.

If $O_{k+1} = A \in X_i$ for $i \in \mathcal{A}$ by (2.91) it follows that

$$\begin{aligned}
&filter_\epsilon^{\leq 0}\left(h, X_\epsilon^{k+1}, X_1^{k+1}, \ldots, X_n^{k+1}\right) = filter_\epsilon^{\leq 0}\left(h, X_\epsilon^k, X_1^k, \ldots, X_{i-1}^k, X_i^k \setminus \{A\}, X_{i+1}^k, \ldots, X_n^k\right) = \\
&filter_\epsilon^{\leq 0}\left(h, X_\epsilon^k, X_1^k, \ldots, X_n^k\right) = X_\epsilon^k \setminus \left(X_\epsilon^k \cap \left(BEvents \sqcup \bigcup_{i \in \mathcal{A}} \{sleep\,(i), hibernate\,(i)\}\right)\right).
\end{aligned} \tag{3.57}$$

Thus from (3.57) we conclude that removing any action A from the set X_i^k does not affect $filter_\epsilon^{\leq 0}$, hence by assumption of the induction hypothesis (3.54) it follows that $Y_\epsilon^{k+1} = Y_\epsilon^k \subseteq Y_\epsilon$.

This completes the induction step. Since the ordering $O_1, \ldots, O_K$ is arbitrary, we can safely conclude that $filter_\epsilon^{\leq 0}$ is monotonic (in the domain PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$). □

3.1.3 Idempotence

Definition 3.1.19. An event filter function $filter_\epsilon^\alpha$ is called *idempotent* for some downward closed domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$ iff for all histories $h \in \mathscr{G}$, all sets $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$ it holds that

$$filter_\epsilon^\alpha\left(h, X_\epsilon, X_1, \ldots, X_n\right) = filter_\epsilon^\alpha\left(h, filter_\epsilon^\alpha\left(h, X_\epsilon, X_1, \ldots, X_n\right), X_1, \ldots, X_n\right). \quad (3.58)$$

Definition 3.1.20. Similarly, an action filter function $filter_i^\alpha$ for some $i \in \mathcal{A}$ is called *idempotent* for some downward closed domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$ iff for all sets $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$ it holds that

$$filter_i^\alpha\left(X_1, \ldots, X_n, X_\epsilon\right) = filter_i^\alpha\left(X_1, \ldots, X_{i-1}, filter_i^\alpha\left(X_1, \ldots, X_n, X_\epsilon\right), X_{i+1}, \ldots, X_n, X_\epsilon\right). \quad (3.59)$$

Lemma 3.1.21. *For every global history $h \in \mathscr{G}$ the neutral filter functions $filter_\epsilon^N$ and $filter_i^N$ for any $i \in \mathcal{A}$ are idempotent in the domain $2^{GEvents}$, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$.*

Proof. By Definition 3.1.1 of the neutral filters for any $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$, we get

$$filter_\epsilon^N\left(h, filter_\epsilon^N\left(h, X_\epsilon, X_1, \ldots, X_n\right), X_1, \ldots, X_n\right) = filter_\epsilon^N\left(h, X_\epsilon, X_1, \ldots, X_n\right) = X_\epsilon \quad (3.60)$$

and

$$\begin{aligned} &filter_i^N\left(X_1, \ldots, X_{i-1}, filter_i^N\left(X_1, \ldots, X_n, X_\epsilon\right), X_{i+1}, \ldots, X_n, X_\epsilon\right) = \\ &filter_i^N\left(X_1, \ldots, X_n, X_\epsilon\right) = X_i, \end{aligned} \quad (3.61)$$

from which the statement follows. □

Lemma 3.1.22. *The filter functions of the Byzantine transition template τ^B are idempotent for every global history in the domain PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$.*

Proof. By definition of the Byzantine event filter function (2.24) $filter_\epsilon^B$ only removes receive events. The condition for removing these however does not depend on the fact whether receive events are present in its arguments. Suppose by contradiction that

$$filter_\epsilon^B\left(h, X_\epsilon, X_1, \ldots, X_n\right) = Y_\epsilon$$

and it holds that

$$filter_\epsilon^B\left(h, Y_\epsilon, X_1, \ldots, X_n\right) = Y'_\epsilon \subset Y_\epsilon.$$

This implies that there exist $j, i \in \mathcal{A}$, $\mu \in Msgs$ and $id \in \mathbb{N}$ such that

$$grecv(j, i, \mu, id) \in Y_\epsilon \text{ and} \quad (3.62)$$

$$gsend(i, j, \mu, id) \notin h_\epsilon \text{ and} \quad (3.63)$$

$$(\forall A \in \{\text{🗓}\} \sqcup \overline{GActions_i})\ fake\left(i, gsend(i, j, \mu, id) \mapsto A\right) \notin h_\epsilon \text{ and} \quad (3.64)$$

$$(gsend(i, j, \mu, id) \notin X_i \vee passive(i, Y_\epsilon)) \text{ and} \quad (3.65)$$

$$(\forall A \in \{\text{🗓}\} \sqcup \overline{GActions_i})\ fake\left(i, gsend(i, j, \mu, id) \mapsto A\right) \notin Y_\epsilon. \quad (3.66)$$

However since $Y_\epsilon \subseteq X_\epsilon$ and X_ϵ and Y_ϵ only differ with respect to correct receive events (therefore in particular $passive(i, X_\epsilon) = passive(i, Y_\epsilon)$), (3.62), (3.63, (3.64, (3.65), (3.66) would hold for X_ϵ as well (i.e. if X_ϵ was substituted for Y_ϵ in (3.62), (3.65), (3.66)). Hence any such $grecv(j, i, \mu, id)$ would have already been removed by definition of the Byzantine event filter (2.24). As a result $grecv(j, i, \mu, id) \notin Y_\epsilon$ contradicting (3.62).
Regarding the action filter function $filter_i^B$ for some $i \in \mathcal{A}$, it either returns X_i or $\varnothing$ depending on the presence of $go(i)$ in X_ϵ (by Definition 2.2.17).

- $go(i) \in X_\epsilon$:
  In this first case, the filter simply returns the input set (since t-coherence of X_ϵ was assumed, it is guaranteed that no other system events $sleep\,(i)$ or $hibernate\,(i)$ are present in X_ϵ):
  $$filter_i^B\,(X_1, \ldots, X_n, X_\epsilon) = X_i.$$
  Hence it trivially follows that
  $$filter_i^B\left(X_1, \ldots, X_{i-1}, filter_i^B\,(X_1, \ldots, X_n, X_\epsilon), X_{i+1} \ldots, X_n, X_\epsilon\right) =$$
  $$filter_i^B\,(X_1, \ldots, X_n, X_\epsilon) = X_i.$$

- $go(i) \notin X_\epsilon$:
  In this second case, we get
  $$filter_i^B\,(X_1, \ldots, X_n, X_\epsilon) = \varnothing.$$
  Since $filter_i^B$ satisfies the basic filter property we further get
  $$filter_i^B\left(X_1, \ldots, X_{i-1}, filter_i^B\,(X_1, \ldots, X_n, X_\epsilon), X_{i+1} \ldots, X_n, X_\epsilon\right) =$$
  $$filter_i^B\,(X_1, \ldots, X_{i-1}, \varnothing, X_{i+1}, \ldots, X_n, X_\epsilon) = \varnothing.$$

□

Lemma 3.1.23. *For any global history $h \in \mathscr{G}$, natural number $f \in \mathbb{N}$, the filter $filter_\epsilon^{\leq f}$ (2.91) is idempotent in the domain PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$.*

Proof. By definition of $filter_\epsilon^{\leq f}$ for some $f \in \mathbb{N}$, $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ either

1. $filter_\epsilon^{\leq f}\,(h, X_\epsilon, X_1, \ldots, X_n) = X_\epsilon$
   Since the output of the filter is equal to its input, it is trivially idempotent.
2. $filter_\epsilon^{\leq f}\,(h, X_\epsilon, X_1, \ldots, X_n) = X_\epsilon \setminus \bigcup_{i \in \mathcal{A}} X_{\epsilon_i}^B$
   for $X_{\epsilon_i}^B$ from (2.90). When applying $filter_\epsilon^{\leq f}$ again, we get either
   a) $filter_\epsilon^{\leq f}\left(h, X_\epsilon \setminus \bigcup_{i \in \mathcal{A}} X_{\epsilon_i}^B, X_1, \ldots, X_n\right) = X_\epsilon \setminus \bigcup_{i \in \mathcal{A}} X_{\epsilon_i}^B$
   Since the output is equal to the input, the filter is idempotent in this case.

b) $filter_{\epsilon}^{\leq f}\left(h, X_\epsilon \setminus \bigcup_{i\in\mathcal{A}} X_{\epsilon_i}^B, X_1, \ldots, X_n\right) = (X_\epsilon \setminus \bigcup_{i\in\mathcal{A}} X_{\epsilon_i}^B) \setminus \bigcup_{i\in\mathcal{A}} (X_\epsilon \setminus \bigcup_{j\in\mathcal{A}} X_{\epsilon_j}^B)_{\epsilon_i}^B$
From (2.90) it follows that

$$\bigcup_{i\in\mathcal{A}} (X_\epsilon \setminus \bigcup_{j\in\mathcal{A}} X_{\epsilon_j}^B)_{\epsilon_i}^B = \varnothing. \tag{3.67}$$

Therefore

$$(X_\epsilon \setminus \bigcup_{i\in\mathcal{A}} X_{\epsilon_i}^B) \setminus \bigcup_{i\in\mathcal{A}} (X_\epsilon \setminus \bigcup_{j\in\mathcal{A}} X_{\epsilon_j}^B)_{\epsilon_i}^B = X_\epsilon \setminus \bigcup_{i\in\mathcal{A}} X_{\epsilon_i}^B \tag{3.68}$$

holds. Since the output is equal to the input of the filter in this case also, the statement of the Lemma follows.

□

3.1.4 Filter Composition

To reiterate, given two event filter functions $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$ for some $h \in \mathcal{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ we defined *filter composition* (3.4) as

$$\begin{aligned} &filter_\epsilon^{\beta\circ\alpha}(h, X_\epsilon, X_1, \ldots, X_n) := \\ &filter_\epsilon^{\beta}(h, filter_\epsilon^{\alpha}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n) \end{aligned} \tag{3.69}$$

and for two action filter functions $filter_i^\alpha$ and $filter_i^\beta$ (for $i \in \mathcal{A}$) (3.9) as

$$\begin{aligned} &filter_i^{\beta\circ\alpha}(X_1, \ldots, X_n, X_\epsilon) := \\ &filter_i^{\beta}(X_1, \ldots, X_{i-1}, filter_i^{\alpha}(X_1, \ldots, X_n, X_\epsilon,), X_{i+1}, \ldots, X_n, X_\epsilon). \end{aligned} \tag{3.70}$$

Lemma 3.1.24. *$filter_\epsilon^N$ and $filter_i^N$ (for some $i \in \mathcal{A}$) are the neutral elements w.r.t. filter composition (3.4), (3.9).*

Proof. This directly follows from the definition of the neutral filters (3.2), (3.3). For some arbitrary filters $filter_\epsilon^\alpha$, $filter_i^\alpha$, $i \in \mathcal{A}$, $h \in \mathcal{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ it holds by (3.4), (3.9) and (3.1.1) that

$$\begin{aligned} filter_\epsilon^{\alpha\circ N}(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^{\alpha}(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^{N\circ\alpha}(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{\alpha\circ N}(X_1, \ldots, X_n, X_\epsilon) = filter_i^{\alpha}(X_1, \ldots, X_n, X_\epsilon) = filter_i^{N\circ\alpha}(X_1, \ldots, X_n, X_\epsilon). \end{aligned} \tag{3.71}$$

□

Lemma 3.1.25. *Filter composition preserves the basic filter property, meaning for any $i \in \mathcal{A}$, $h \in \mathcal{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ and arbitrary filters $filter_\epsilon^\alpha$, $filter_\epsilon^\beta$, $filter_i^\alpha$, $filter_i^\beta$ it holds that*

$$\begin{aligned} filter_\epsilon^{\alpha\circ\beta}(h, X_\epsilon, X_1, \ldots, X_n) \subseteq X_\epsilon \\ filter_i^{\alpha\circ\beta}(X_1, \ldots, X_n, X_\epsilon) \subseteq X_i. \end{aligned} \tag{3.72}$$

Thus $filter_\epsilon^{\alpha\circ\beta}$ and $filter_i^{\alpha\circ\beta}$ are again filter functions.

Proof. By definition of event filter composition (3.4):

$$filter_{\epsilon}^{\alpha\circ\beta}(h, X_\epsilon, X_1, \ldots, X_n) = filter_{\epsilon}^{\alpha}\left(h, filter_{\epsilon}^{\beta}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right). \quad (3.73)$$

Since $filter_{\epsilon}^{\alpha}$ and $filter_{\epsilon}^{\beta}$ are both filter functions and therefore satisfy the basic filter property, it holds that

$$filter_{\epsilon}^{\alpha}\left(h, filter_{\epsilon}^{\beta}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right) \subseteq filter_{\epsilon}^{\beta}(h, X_\epsilon, X_1, \ldots, X_n) \subseteq X_\epsilon, \quad (3.74)$$

resulting (by transitivity of $\subseteq$) in

$$filter_{\epsilon}^{\alpha\circ\beta}(h, X_\epsilon, X_1, \ldots, X_n) \subseteq X_\epsilon. \quad (3.75)$$

Similarly by definition of action filter composition (3.9):

$$filter_{i}^{\alpha\circ\beta}(X_1, \ldots, X_n, X_\epsilon) = filter_{i}^{\alpha}\left(X_1, \ldots, X_{i-1}, filter_{i}^{\beta}(X_1, \ldots, X_n, X_\epsilon), X_{i+1}, \ldots, X_n, X_\epsilon\right). \quad (3.76)$$

Since $filter_{i}^{\alpha}$ and $filter_{i}^{\beta}$ are both filter functions and thus satisfy the basic filter property, it holds that

$$\begin{aligned} &filter_{i}^{\alpha}\left(X_1, \ldots, X_{i-1}, filter_{i}^{\beta}(X_1, \ldots, X_n, X_\epsilon), X_{i+1}, \ldots, X_n, X_\epsilon\right) \subseteq \\ &filter_{i}^{\beta}(X_1, \ldots, X_n, X_\epsilon) \subseteq X_i, \end{aligned} \quad (3.77)$$

resulting (again by transitivity of $\subseteq$) in

$$filter_{i}^{\alpha\circ\beta}(X_1, \ldots, X_n, X_\epsilon) \subseteq X_i. \quad (3.78)$$

□

Lemma 3.1.26. *Filter composition (3.4), (3.9) preserves monotonicity: Given any two filter functions $filter_{\epsilon}^{\alpha}$ and $filter_{\epsilon}^{\beta}$ (respectively $filter_{i}^{\alpha}$ and $filter_{i}^{\beta}$ for some $i \in \mathcal{A}$), which are monotonic for a given downward closed domain ($PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$), the combined filter $filter_{\epsilon}^{\alpha\circ\beta}$ (respectively $filter_{i}^{\alpha\circ\beta}$) is also monotonic for this domain.*

Proof. Since given two arbitrary monotonic event filter functions $filter_{\epsilon}^{\alpha}$, $filter_{\epsilon}^{\beta}$ for any $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$, $X'_\epsilon \subseteq X_\epsilon$, $X'_1 \subseteq X_1$, ..., $X'_n \subseteq X_n$, it holds that

$$\begin{aligned} filter_{\epsilon}^{\alpha}(h, X'_\epsilon, X'_1, \ldots, X'_n) &\subseteq filter_{\epsilon}^{\alpha}(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_{\epsilon}^{\beta}(h, X'_\epsilon, X'_1, \ldots, X'_n) &\subseteq filter_{\epsilon}^{\beta}(h, X_\epsilon, X_1, \ldots, X_n), \end{aligned} \quad (3.79)$$

it immediately follows that

$$\begin{aligned} &filter_{\epsilon}^{\alpha}\left(h, filter_{\epsilon}^{\beta}(h, X'_\epsilon, X'_1, \ldots, X'_n), X'_1, \ldots, X'_n\right) \subseteq \\ &filter_{\epsilon}^{\alpha}\left(h, filter_{\epsilon}^{\beta}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right) \end{aligned} \quad (3.80)$$

and

$$\begin{aligned} &filter_{\epsilon}^{\beta}(h, filter_{\epsilon}^{\alpha}(h, X'_\epsilon, X'_1, \ldots, X'_n), X'_1, \ldots, X'_n) \subseteq \\ &filter_{\epsilon}^{\beta}(h, filter_{\epsilon}^{\alpha}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n). \end{aligned} \quad (3.81)$$

Similarly, for monotonic action filter functions $filter_i^\alpha$, $filter_i^\beta$, $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$, $X'_\epsilon \subseteq X_\epsilon$, $X'_1 \subseteq X_1$, ..., $X'_n \subseteq X_n$, since

$$\begin{aligned} filter_i^\alpha\left(X'_1,\ldots,X'_n,X'_\epsilon\right) &\subseteq filter_i^\alpha\left(X_1,\ldots,X_n,X_\epsilon\right) \\ filter_i^\beta\left(X'_1,\ldots,X'_n,X'_\epsilon\right) &\subseteq filter_i^\beta\left(X_1,\ldots,X_n,X_\epsilon\right) \end{aligned} \tag{3.82}$$

it immediately follows that

$$\begin{aligned} &filter_i^\alpha\left(X'_1,\ldots,X'_{i-1},filter_i^\beta\left(X'_1,\ldots,X'_n,X'_\epsilon\right),X'_{i+1},\ldots,X'_n,X'_\epsilon\right) \subseteq \\ &filter_i^\alpha\left(X_1,\ldots,X_{i-1},filter_i^\beta\left(X_1,\ldots,X_n,X_\epsilon\right),X_{i+1},\ldots,X_n,X_\epsilon\right) \end{aligned} \tag{3.83}$$

and

$$\begin{aligned} &filter_i^\beta\left(X'_1,\ldots,X'_{i-1},filter_i^\alpha\left(X'_1,\ldots,X'_n,X'_\epsilon\right),X'_{i+1},\ldots,X'_n,X'_\epsilon\right) \subseteq \\ &filter_i^\beta\left(X_1,\ldots,X_{i-1},filter_i^\alpha\left(X_1,\ldots,X_n,X_\epsilon\right),X_{i+1},\ldots,X_n,X_\epsilon\right). \end{aligned} \tag{3.84}$$

□

Lemma 3.1.27. *Filter composition (3.4), (3.9) preserves simple monotonicity: Given any two filter functions $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$ and $filter_i^\beta$ for some $i \in \mathcal{A}$), which are simply monotonic for a given downward closed domain ($PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$), the combined filter $filter_\epsilon^{\alpha\circ\beta}$ (respectively $filter_i^{\alpha\circ\beta}$) is also simply monotonic for this domain.*

Proof. This proof is analogous to that of Lemma 3.1.26. Since, given two arbitrary simply monotonic event filter functions $filter_\epsilon^\alpha$, $filter_\epsilon^\beta$ for any $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$, $X'_\epsilon \subseteq X_\epsilon$, it holds that

$$\begin{aligned} filter_\epsilon^\alpha\left(h,X'_\epsilon,X_1,\ldots,X_n\right) &\subseteq filter_\epsilon^\alpha\left(h,X_\epsilon,X_1,\ldots,X_n\right) \\ filter_\epsilon^\beta\left(h,X'_\epsilon,X_1,\ldots,X_n\right) &\subseteq filter_\epsilon^\beta\left(h,X_\epsilon,X_1,\ldots,X_n\right), \end{aligned} \tag{3.85}$$

it immediately follows that

$$\begin{aligned} &filter_\epsilon^\alpha\left(h,filter_\epsilon^\beta\left(h,X'_\epsilon,X_1,\ldots,X_n\right),X_1,\ldots,X_n\right) \subseteq \\ &filter_\epsilon^\alpha\left(h,filter_\epsilon^\beta\left(h,X_\epsilon,X_1,\ldots,X_n\right),X_1,\ldots,X_n\right) \end{aligned} \tag{3.86}$$

and

$$\begin{aligned} &filter_\epsilon^\beta\left(h,filter_\epsilon^\alpha\left(h,X'_\epsilon,X_1,\ldots,X_n\right),X_1,\ldots,X_n\right) \subseteq \\ &filter_\epsilon^\beta\left(h,filter_\epsilon^\alpha\left(h,X_\epsilon,X_1,\ldots,X_n\right),X_1,\ldots,X_n\right). \end{aligned} \tag{3.87}$$

Similarly, for simply monotonic action filter functions $filter_i^\alpha$, $filter_i^\beta$, $i \in \mathcal{A}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$, $X'_i \subseteq X_i$, since

$$\begin{aligned} filter_i^\alpha\left(X_1,\ldots,X_{i-1},X'_i,X_{i+1},\ldots,X_n,X_\epsilon\right) &\subseteq filter_i^\alpha\left(X_1,\ldots,X_n,X_\epsilon\right) \\ filter_i^\beta\left(X_1,\ldots,X_{i-1},X'_i,X_{i+1},\ldots,X_n,X_\epsilon\right) &\subseteq filter_i^\beta\left(X_1,\ldots,X_n,X_\epsilon\right) \end{aligned} \tag{3.88}$$

it immediately follows that

$$filter_i^{\alpha}\left(X_1,\ldots,X_{i-1},filter_i^{\beta}\left(X_1,\ldots,X_{i-1},X_i',X_{i+1},\ldots,X_n,X_\epsilon\right),X_{i+1},\ldots,X_n,X_\epsilon\right)\subseteq$$
$$filter_i^{\alpha}\left(X_1,\ldots,X_{i-1},filter_i^{\beta}\left(X_1,\ldots,X_n,X_\epsilon\right),X_{i+1},\ldots,X_n,X_\epsilon\right) \tag{3.89}$$

and

$$filter_i^{\beta}\left(X_1,\ldots,X_{i-1},filter_i^{\alpha}\left(X_1,\ldots,X_{i-1},X_i',X_{i+1},\ldots,X_n,X_\epsilon\right),X_{i+1},\ldots,X_n,X_\epsilon\right)\subseteq$$
$$filter_i^{\beta}\left(X_1,\ldots,X_{i-1},filter_i^{\alpha}\left(X_1,\ldots,X_n,X_\epsilon\right),X_{i+1},\ldots,X_n,X_\epsilon\right). \tag{3.90}$$

□

In the following we define two sets of event filters, which we will use as counterexamples.

Definition 3.1.28. For some $k\in\mathbb{N}\setminus\{0\}$ and $\{E_a^1,E_a^2,\ldots,E_a^k,E_b^1,E_b^2,\ldots,E_b^k\}\subseteq GEvents$, where all elements in $\{E_a^1,E_a^2,\ldots,E_a^k,E_b^1,E_b^2,\ldots,E_b^k\}$ are pairwise distinct, let us define

$$filter_\epsilon^{\alpha^k}(h,X_\epsilon,X_1,\ldots,X_n):=X_\epsilon\setminus\{E_a^l\in GEvents\mid 1\leq l\leq k\ \wedge\ E_b^l\notin X_\epsilon\} \tag{3.91}$$

$$filter_\epsilon^{\beta^k}(h,X_\epsilon,X_1,\ldots,X_n):=X_\epsilon\setminus\{E_b^{(l\bmod k)+1}\in GEvents\mid 1\leq l\leq k\wedge E_a^l\notin X_\epsilon\}. \tag{3.92}$$

Lemma 3.1.29. *The filter functions $filter_\epsilon^{\alpha^k}$ (3.91) and $filter_\epsilon^{\beta^k}$ (3.92) are monotonic and idempotent for the domain $2^{GEvents}$, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$.*

Proof. From (3.91) and (3.92) it trivially follows that $filter_\epsilon^{\alpha^k}$ and $filter_\epsilon^{\beta^k}$ satisfy the basic filter property. The fact that the filter functions (3.91) and (3.92) are idempotent is also quite obvious, as they do not remove any elements on which their removal depends on. Monotonicity on the other hand is less apparent. Since both $filter_\epsilon^{\alpha^k}$ and $filter_\epsilon^{\beta^k}$ do not depend on any actions, we only need to focus on the set of events in our proof.

Suppose $O_1,\ldots,O_K$ is an arbitrary ordering of the events X_ϵ, where $K=|X_\epsilon|$, let $X_\epsilon^0=X_\epsilon$ and define for $0\leq l\leq K-1$

$$X_\epsilon^{l+1}=X_\epsilon^k\setminus\{O_{l+1}\} \tag{3.93}$$

and $X_i^0=X_i$, X_i^l arbitrary for $i\in\mathcal{A}$.

We start with the proof for $filter_\epsilon^{\alpha^k}$ by induction on l.
Induction Hypothesis:

$$filter_\epsilon^{\alpha^k}\left(h,X_\epsilon^l,X_1^l,\ldots,X_n^l\right)\subseteq filter_\epsilon^{\alpha^k}(h,X_\epsilon,X_1,\ldots,X_n)=Y_\epsilon \tag{3.94}$$

Base Case: For $l=0$ trivially

$$filter_\epsilon^{\alpha^k}\left(h,X_\epsilon^0,X_1^0,\ldots,X_n^0\right)=filter_\epsilon^{\alpha^k}(h,X_\epsilon,X_1,\ldots,X_n)=Y_\epsilon \tag{3.95}$$

Induction Step: Suppose the induction hypothesis (3.94) holds for some $l\geq 0$. We distinguish between the following cases for O_{l+1}:

- If $O_{l+1} = E_b^{k'}$, for $0 \leq k' \leq k$

$$filter_\epsilon^{\alpha^k}\left(h, X_\epsilon^{l+1}, X_1^{l+1}, \ldots, X_n^{l+1}\right) = filter_\epsilon^{\alpha^k}\left(h, X_\epsilon^l \setminus \{E_b^{k'}\}, X_1^l, \ldots, X_n^l\right) = filter_\epsilon^{\alpha^k}\left(h, X_\epsilon^l, X_1^l, \ldots, X_n^l\right) \setminus \{E_a^{k'}, E_b^{k'}\} \subseteq Y_\epsilon \tag{3.96}$$

  and the induction hypothesis remains satisfied for $l+1$.

- Otherwise if $O_{l+1} \neq E_b^{k'}$, for $0 \leq k' \leq k$

$$filter_\epsilon^{\alpha^k}\left(h, X_\epsilon^{l+1}, X_1^{l+1}, \ldots, X_n^{l+1}\right) = filter_\epsilon^{\alpha^k}\left(h, X_\epsilon^l \setminus \{O_{l+1}\}, X_1^l, \ldots, X_n^l\right) = filter_\epsilon^{\alpha^k}\left(h, X_\epsilon^l, X_1^l, \ldots, X_n^l\right) \setminus \{O_{l+1}\} \subseteq Y_\epsilon \tag{3.97}$$

  and the induction hypothesis again remains satisfied for $l+1$.

This concludes the induction step. Since $O_1, \ldots, O_K$ is an arbitrary sequence of events in X_ϵ, we conclude that $filter_\epsilon^{\alpha^k}$ is indeed monotonic.

Next we examine $filter_\epsilon^{\beta^k}$. We use the same arbitrary sequence $O_1, \ldots, O_K$, (3.92), $X_i^0 = X_i$ and X_i^l arbitrary for $i \in \mathcal{A}$.
Induction Hypothesis:

$$filter_\epsilon^{\beta^k}\left(h, X_\epsilon^l, X_1^l, \ldots, X_n^l\right) \subseteq filter_\epsilon^{\beta^k}\left(h, X_\epsilon, X_1, \ldots, X_n\right) = Y_\epsilon \tag{3.98}$$

Base Case: For $l = 0$ trivially

$$filter_\epsilon^{\beta^k}\left(h, X_\epsilon^0, X_1^0, \ldots, X_n^0\right) = filter_\epsilon^{\beta^k}\left(h, X_\epsilon, X_1, \ldots, X_n\right) = Y_\epsilon \tag{3.99}$$

Induction Step: Suppose the induction hypothesis (3.98) holds for some $l \geq 0$. We distinguish between the following cases for O_{l+1}:

- If $O_{l+1} = E_a^{k'}$, for $0 \leq k' \leq k$

$$filter_\epsilon^{\beta^k}\left(h, X_\epsilon^{l+1}, X_1^{l+1}, \ldots, X_n^{l+1}\right) = filter_\epsilon^{\beta^k}\left(h, X_\epsilon^l \setminus \{E_a^{k'}\}, X_1^l, \ldots, X_n^l\right) = filter_\epsilon^{\beta^k}\left(h, X_\epsilon^l, X_1^l, \ldots, X_n^l\right) \setminus \{E_b^{(k' \bmod k)+1}, E_a^{k'}\} \subseteq Y_\epsilon \tag{3.100}$$

  and the induction hypothesis remains satisfied for $l+1$.

- Otherwise if $O_{l+1} \neq E_a^{k'}$, for $0 \leq k' \leq k$

$$filter_\epsilon^{\beta^k}\left(h, X_\epsilon^{l+1}, X_1^{l+1}, \ldots, X_n^{l+1}\right) = filter_\epsilon^{\beta^k}\left(h, X_\epsilon^l \setminus \{O_{l+1}\}, X_1^l, \ldots, X_n^l\right) = filter_\epsilon^{\beta^k}\left(h, X_\epsilon^l, X_1^l, \ldots, X_n^l\right) \setminus \{O_{l+1}\} \subseteq Y_\epsilon \tag{3.101}$$

  and the induction hypothesis again remains satisfied for $l+1$.

This concludes the induction step. Since $O_1, \ldots, O_K$ is an arbitrary sequence of events in X_ϵ, we conclude that the event filter $filter_\epsilon^{\beta^k}$ is monotonic as well. □

Lemma 3.1.30. *Filter composition generally does not preserve idempotence - even for monotonic filters: Given any two filter functions* $filter_\epsilon^\alpha$ *and* $filter_\epsilon^\beta$ *(respectively* $filter_i^\alpha$ *and* $filter_i^\beta$ *for some* $i \in \mathcal{A}$*), which are idempotent and monotonic for some given downward closed domain (*$PD_\epsilon \subseteq 2^{GEvents}$*,* $PD_1 \subseteq 2^{\overline{GActions}_1}$*, ...,* $PD_n \subseteq 2^{\overline{GActions}_n}$*), the combined filter* $filter_\epsilon^{\alpha\circ\beta}$ *(respectively* $filter_i^{\alpha\circ\beta}$*) is not necessarily idempotent for this domain.*

Proof. Assuming $\{E_a^1, E_a^2, E_b^1, E_b^2\} \subseteq GEvents$ for pairwise distinct elements $\{E_a^1, E_a^2, E_b^1, E_b^2\}$, we provide the following counterexample. Using the monotonic and idempotent (by Lemma 3.1.29) filter functions from Definition 3.1.28 for $k = 2$, any $h \in \mathscr{G}$ and sets $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$, the combined filter (3.4) gives the following result:

$$filter_\epsilon^{\beta^2\circ\alpha^2}\left(h, \{E_a^1, E_a^2, E_b^1\}, X_1, \ldots, X_n\right) = \{E_a^1\}. \tag{3.102}$$

A second application on this resulting set however gives

$$filter_\epsilon^{\beta^2\circ\alpha^2}\left(h, \{E_a^1\}, X_1, \ldots, X_n\right) = \varnothing. \tag{3.103}$$

Thus we conclude that filter composition (3.4) generally does not preserve idempotence, even if the filter functions are both monotonic as well (a similar counterexample can be constructed for action filter composition (3.9)). □

Lemma 3.1.31. *Filter composition is associative.*

Proof. For three event filter functions $filter_\epsilon^\alpha$, $filter_\epsilon^\beta$, $filter_\epsilon^\gamma$, $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ by definition of event filter composition (3.4)

$$\begin{aligned}
&filter_\epsilon^{(\alpha\circ\beta)\circ\gamma}(h, X_\epsilon, X_1, \ldots, X_n) = \\
&filter_\epsilon^{\alpha}\left(h, filter_\epsilon^{\beta}(h, filter_\epsilon^{\gamma}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n), X_1, \ldots, X_n\right) = \\
&filter_\epsilon^{\alpha\circ(\beta\circ\gamma)}(h, X_\epsilon, X_1, \ldots, X_n).
\end{aligned} \tag{3.104}$$

Thus we conclude

$$filter_\epsilon^{(\alpha\circ\beta)\circ\gamma}(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^{\alpha\circ(\beta\circ\gamma)}(h, X_\epsilon, X_1, \ldots, X_n). \tag{3.105}$$

Similarly for three action filter functions $filter_i^\alpha$, $filter_i^\beta$, $filter_i^\gamma$, $i \in \mathcal{A}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ by definition of action filter composition (3.9)

$$\begin{aligned}
&filter_i^{(\alpha\circ\beta)\circ\gamma}(X_1, \ldots, X_n, X_\epsilon) = \\
&filter_i^{\alpha}\left(X_1, \ldots, X_{i-1}, filter_i^{\beta}(X_1, \ldots, X_{i-1}, filter_i^{\gamma}(X_1, \ldots, X_n, X_\epsilon), X_{i+1}, \ldots, X_n, X_\epsilon),\right. \\
&\left. X_{i+1}, \ldots, X_n, X_\epsilon\right) = filter_i^{\alpha\circ(\beta\circ\gamma)}(X_1, \ldots, X_n, X_\epsilon).
\end{aligned} \tag{3.106}$$

Thus we conclude also for action filters

$$filter_i^{(\alpha\circ\beta)\circ\gamma}(X_1, \ldots, X_n, X_\epsilon) = filter_i^{\alpha\circ(\beta\circ\gamma)}(X_1, \ldots, X_n, X_\epsilon). \tag{3.107}$$

□

Lemma 3.1.32. *Filter composition is generally not commutative, even for monotonic and idempotent filters.*

Proof. By counterexample. Recall that $filter_\epsilon^B$ and $filter_\epsilon^{\leq 0}$ are both monotonic and idempotent for the domain PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ by Lemmas 3.1.17, 3.1.18, 3.1.22, 3.1.23.
For $i, j \in \mathcal{A}$, $\mu \in Msgs$, $id \in \mathbb{N}$, $A' \in \{🗓\} \sqcup \overline{GActions_i}$, $h \in \mathscr{G}$, where

- $gsend(i, j, \mu, id) \notin h$,
- for all $A \in \{🗓\} \sqcup \overline{GActions_i}$ $fake\,(i, gsend(i, j, \mu, id) \mapsto A) \notin h$,
- $X_1 \subseteq \overline{GActions_1}$, ..., $X_n \subseteq \overline{GActions_n}$, where $gsend(i, j, \mu, id) \notin X_i$ and
- $X_\epsilon = \{fake\,(i, gsend(i, j, \mu, id) \mapsto A'), grecv(j, i, \mu, id)\}$

we find

$$\begin{gathered} filter_\epsilon^{\leq 0 \circ B}\,(h, X_\epsilon, X_1, \ldots, X_n) = \{grecv(j, i, \mu, id)\} \\ \text{but} \\ filter_\epsilon^{B \circ \leq 0}\,(h, X_\epsilon, X_1, \ldots, X_n) = \varnothing. \end{gathered} \tag{3.108}$$

□

Lemma 3.1.33. *For some arbitrary filters $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$) and $filter_\epsilon^\beta$ (respectively $filter_i^\beta$ for $i \in \mathcal{A}$), if for some downward closed domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$ $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$) is simply monotonic or $filter_\epsilon^\beta$ (respectively $filter_i^\beta$) is stricter than $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$), then for some $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$*

$$\begin{aligned} filter_\epsilon^{\alpha \circ \beta}\,(h, X_\epsilon, X_1, \ldots, X_n) &\subseteq filter_\epsilon^\alpha\,(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{\alpha \circ \beta}\,(X_1, \ldots, X_n, X_\epsilon) &\subseteq filter_i^\alpha\,(X_1, \ldots, X_n, X_\epsilon) \end{aligned} \tag{3.109}$$

holds.

Proof. There are two cases.

1. Suppose that $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$) is simply monotonic for the downward closed domain PD_ϵ, PD_1, ..., PD_n. This implies that for all $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$, $X'_\epsilon \subseteq X_\epsilon$

   $$filter_\epsilon^\alpha\,(h, X'_\epsilon, X_1, \ldots, X_n) \subseteq filter_\epsilon^\alpha\,(h, X_\epsilon, X_1, \ldots, X_n)\,. \tag{3.110}$$

   Since by the basic filter property for all $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions_1}$, ..., $X_n \subseteq \overline{GActions_n}$

   $$filter_\epsilon^\beta\,(h, X_\epsilon, X_1, \ldots, X_n) \subseteq X_\epsilon, \tag{3.111}$$

it particularly holds that

$$filter_{\epsilon}^{\alpha\circ\beta}(h, X_{\epsilon}, X_1, \ldots, X_n) = filter_{\epsilon}^{\alpha}\left(h, filter_{\epsilon}^{\beta}(h, X_{\epsilon}, X_1, \ldots, X_n), X_1, \ldots, X_n\right) \subseteq filter_{\epsilon}^{\alpha}(h, X_{\epsilon}, X_1, \ldots, X_n). \tag{3.112}$$

For action filters our assumption implies that for all $X_{\epsilon} \in PD_{\epsilon}$, $X_1 \in PD_1$, ..., $X_n \in PD_n$, $X_i' \subseteq X_i$

$$filter_i^{\alpha}(X_1, \ldots, X_{i-1}, X_i', X_{i+1}, \ldots X_n, X_{\epsilon}) \subseteq filter_i^{\alpha}(X_1, \ldots, X_n, X_{\epsilon}). \tag{3.113}$$

Since by the basic filter property for all $i \in \mathcal{A}$, $X_{\epsilon} \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$

$$filter_i^{\beta}(X_1, \ldots, X_n, X_{\epsilon}) \subseteq X_i, \tag{3.114}$$

it also holds that

$$\begin{aligned} &filter_i^{\alpha\circ\beta}(X_1, \ldots, X_n, X_{\epsilon}) = \\ &filter_i^{\alpha}\left(X_1, \ldots, X_{i-1}, filter_i^{\beta}(X_1, \ldots, X_n, X_{\epsilon}), X_{i+1}, \ldots, X_n, X_{\epsilon}\right) \subseteq \\ &filter_i^{\alpha}(X_1, \ldots, X_n, X_{\epsilon}). \end{aligned} \tag{3.115}$$

the statement is also satisfied for action filters.

2. Suppose that $filter_{\epsilon}^{\beta}$ (respectively $filter_i^{\beta}$) is stricter than $filter_{\epsilon}^{\alpha}$ (respectively $filter_i^{\alpha}$). This implies that for all $h \in \mathscr{G}$, $X_{\epsilon} \in PD_{\epsilon}$, $X_1 \in PD_1$, ..., $X_n \in PD_n$

$$filter_{\epsilon}^{\beta}(h, X_{\epsilon}, X_1, \ldots, X_n) \subseteq filter_{\epsilon}^{\alpha}(h, X_{\epsilon}, X_1, \ldots, X_n), \tag{3.116}$$

thus by the basic filter property of $filter_{\epsilon}^{\alpha}$

$$\begin{aligned} &filter_{\epsilon}^{\alpha\circ\beta}(h, X_{\epsilon}, X_1, \ldots, X_n) = \\ &filter_{\epsilon}^{\alpha}\left(h, filter_{\epsilon}^{\beta}(h, X_{\epsilon}, X_1, \ldots, X_n), X_1, \ldots, X_n\right) \subseteq \\ &filter_{\epsilon}^{\beta}(h, X_{\epsilon}, X_1, \ldots, X_n) \subseteq filter_{\epsilon}^{\alpha}(h, X_{\epsilon}, X_1, \ldots, X_n) \end{aligned} \tag{3.117}$$

Similarly, for action filters for any $i \in \mathcal{A}$, $X_{\epsilon} \in PD_{\epsilon}$, $X_1 \in PD_1$, ..., $X_n \in PD_n$ we get that

$$filter_i^{\beta}(X_1, \ldots, X_n, X_{\epsilon}) \subseteq filter_i^{\alpha}(X_1, \ldots, X_n, X_{\epsilon}) \tag{3.118}$$

thus by the basic filter property of $filter_i^{\alpha}$

$$\begin{aligned} &filter_i^{\alpha\circ\beta}(X_1, \ldots, X_n, X_{\epsilon}) = \\ &filter_i^{\alpha}\left(X_1, \ldots, X_{i-1}, filter_i^{\beta}(X_1, \ldots, X_n, X_{\epsilon}), X_{i+1}, \ldots, X_n, X_{\epsilon}\right) \subseteq \\ &filter_i^{\beta}(X_1, \ldots, X_n, X_{\epsilon}) \subseteq filter_i^{\alpha}(X_1, \ldots, X_n, X_{\epsilon}) \end{aligned} \tag{3.119}$$

and we are done.

□

Since Definition 2.3.14 regarding strictness of filters only considers the whole domain of actions and events, we provide a separate definition of strictness that is limited to special domains.

Definition 3.1.34. We say that $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$ for some $i \in \mathcal{A}$) is *stricter than* $filter_\epsilon^\beta$ (respectively $filter_i^\beta$) *for some domain* $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$ iff for any $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$

$$\begin{aligned} filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) &\subseteq filter_\epsilon^\beta(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon) &\subseteq filter_i^\beta(X_1, \ldots, X_n, X_\epsilon). \end{aligned}$$

Similarly we say that $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$ for some $i \in \mathcal{A}$) is *equal to* $filter_\epsilon^\beta$ (respectively $filter_i^\beta$) *for some domain* $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$ iff for any $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$

$$\begin{aligned} filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) &= filter_\epsilon^\beta(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon) &= filter_i^\beta(X_1, \ldots, X_n, X_\epsilon). \end{aligned}$$

Lemma 3.1.35. *For two filter functions* $filter_\epsilon^\alpha$ *(respectively* $filter_i^\alpha$*) and* $filter_\epsilon^\beta$ *(respectively* $filter_i^\beta$*), where for some downward closed domain* $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$, $filter_\epsilon^\alpha$ *(respectively* $filter_i^\alpha$*) is stricter than* $filter_\epsilon^\beta$ *(respectively* $filter_i^\beta$*) and idempotent, the combined filter* $filter_\epsilon^{\beta\circ\alpha}$ *(respectively* $filter_i^{\beta\circ\alpha}$*) simplifies to*

$$\begin{aligned} filter_\epsilon^{\beta\circ\alpha}(h, X_\epsilon, X_1, \ldots, X_n) &= filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{\beta\circ\alpha}(X_1, \ldots, X_n, X_\epsilon) &= filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon) \end{aligned} \tag{3.120}$$

for this domain.

Proof. If $filter_\epsilon^\beta$ were to remove something after $filter_\epsilon^\alpha$ had been applied, it would contradict our assumption that $filter_\epsilon^\alpha \subseteq filter_\epsilon^\beta$, as $filter_\epsilon^\alpha$ is assumed to be idempotent. Therefore by definition of event filter composition (3.4) and the basic filter property of $filter_\epsilon^\beta$ we get for some $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$

$$\begin{aligned} &filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^\alpha(h, filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n) \subseteq \\ &filter_\epsilon^\beta(h, filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n) \subseteq filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) \end{aligned}$$

resulting in

$$\begin{aligned} &filter_\epsilon^\beta(h, filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n) = \\ &filter_\epsilon^{\beta\circ\alpha}(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n). \end{aligned} \tag{3.121}$$

The reasoning for action filters is completely analogous. □

Lemma 3.1.36. *For two filter functions* $filter_\epsilon^\alpha$ *(respectively* $filter_i^\alpha$ *for* $i \in \mathcal{A}$*) and* $filter_\epsilon^\beta$ *(respectively* $filter_i^\beta$*), where for some downward closed domain* $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq$

$2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$, $filter_\epsilon^\alpha$ *(respectively* $filter_i^\alpha$*) is stricter than* $filter_\epsilon^\beta$ *(respectively* $filter_i^\beta$*), idempotent and simply monotonic, the combined filter* $filter_\epsilon^{\alpha\circ\beta}$ *(*$filter_i^{\alpha\circ\beta}$*) simplifies to*

$$\begin{aligned} filter_\epsilon^{\alpha\circ\beta}(h, X_\epsilon, X_1, \ldots, X_n) &= filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{\alpha\circ\beta}(X_1, \ldots, X_n, X_\epsilon) &= filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon) \end{aligned} \tag{3.122}$$

for this domain.

Proof. For some $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$, by Lemma 3.1.33 it follows that

$$filter_\epsilon^\alpha\left(h, filter_\epsilon^\beta(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right) \subseteq filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n). \tag{3.123}$$

Also by simple monotonicity of $filter_\epsilon^\alpha$ and our assumption that it is stricter than $filter_\epsilon^\beta$, we get that

$$\begin{aligned} &filter_\epsilon^\alpha(h, filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n) \subseteq \\ &filter_\epsilon^\alpha\left(h, filter_\epsilon^\beta(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right). \end{aligned} \tag{3.124}$$

From (3.124), by idempotence of $filter_\epsilon^\alpha$, we get

$$filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) \subseteq filter_\epsilon^\alpha\left(h, filter_\epsilon^\beta(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right). \tag{3.125}$$

By (3.125) and (3.123) it follows that

$$filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^\alpha\left(h, filter_\epsilon^\beta(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right). \tag{3.126}$$

Thus, the combined filter simplifies to

$$filter_\epsilon^{\alpha\circ\beta}(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n). \tag{3.127}$$

Similarly, for action filters, by simple monotonicity of $filter_i^\alpha$ using Lemma 3.1.33, it follows that

$$filter_i^\alpha\left(X_1, \ldots, X_{i-1}, filter_i^\beta(X_1, \ldots, X_n, X_\epsilon), X_{i+1}, \ldots, X_n, X_\epsilon\right) \subseteq filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon). \tag{3.128}$$

Also by simple monotonicity of $filter_i^\alpha$ and our assumption that it is stricter than $filter_i^\beta$, we get that

$$\begin{aligned} &filter_i^\alpha(X_1, \ldots, X_{i-1}, filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon), X_{i+1}, \ldots, X_n, X_\epsilon) \subseteq \\ &filter_i^\alpha\left(X_1, \ldots, X_{i-1}, filter_i^\beta(X_1, \ldots, X_n, X_\epsilon), X_{i+1}, \ldots, X_n, X_\epsilon\right). \end{aligned} \tag{3.129}$$

From (3.129), by idempotence of $filter_i^\alpha$, we get

$$filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon) \subseteq filter_i^\alpha\left(X_1, \ldots, X_{i-1}, filter_i^\beta(X_1, \ldots, X_n, X_\epsilon), X_{i+1}, \ldots, X_n, X_\epsilon\right). \tag{3.130}$$

By (3.130) and (3.128), it follows that

$$filter_i^{\alpha}\left(X_1, \ldots, X_n, X_\epsilon\right) = filter_i^{\alpha}\left(X_1, \ldots, X_{i-1}, filter_i^{\beta}\left(X_1, \ldots, X_n, X_\epsilon\right), X_{i+1}, \ldots, X_n, X_\epsilon\right). \tag{3.131}$$

Thus the combined filter again simplifies to

$$filter_i^{\alpha\circ\beta}\left(X_1, \ldots, X_n, X_\epsilon\right) = filter_i^{\alpha}\left(X_1, \ldots, X_n, X_\epsilon\right) \tag{3.132}$$

and we are done. □

What follows is a special case for Lemmas 3.1.35 and 3.1.36.

Corollary 3.1.37. *For two filter functions $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$ for $i \in \mathcal{A}$) and $filter_\epsilon^\beta$ (respectively $filter_i^\beta$), where for a downward closed domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$, $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$) is equal to $filter_\epsilon^\beta$ (respectively $filter_i^\beta$) and idempotent, the combined filter $filter_\epsilon^{\alpha\circ\beta}$ (respectively $filter_i^{\alpha\circ\beta}$) simplifies to*

$$\begin{aligned} filter_\epsilon^{\alpha\circ\beta}\left(h, X_\epsilon, X_1, \ldots, X_n\right) &= filter_\epsilon^{\alpha}\left(h, X_\epsilon, X_1, \ldots, X_n\right) = filter_\epsilon^{\beta}\left(h, X_\epsilon, X_1, \ldots, X_n\right) \\ filter_i^{\alpha\circ\beta}\left(X_1, \ldots, X_n, X_\epsilon\right) &= filter_i^{\alpha}\left(X_1, \ldots, X_n, X_\epsilon\right) = filter_i^{\beta}\left(X_1, \ldots, X_n, X_\epsilon\right) \end{aligned} \tag{3.133}$$

for this domain.

3.1.5 Filter Intersection

To reiterate, for two event filter functions $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$ for some $h \in \mathcal{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions_1}$, ..., $X_n \subseteq \overline{GActions_n}$, we defined *filter intersection* (3.5) as

$$\begin{aligned} &filter_\epsilon^{\alpha+\beta}\left(h, X_\epsilon, X_1, \ldots, X_n\right) := \\ &filter_\epsilon^{\beta\circ\alpha}\left(h, X_\epsilon, X_1, \ldots, X_n\right) \cap filter_\epsilon^{\alpha\circ\beta}\left(h, X_\epsilon, X_1, \ldots, X_n\right) \end{aligned} \tag{3.134}$$

and for two action filter functions $filter_i^\alpha$ and $filter_i^\beta$ (for $i \in \mathcal{A}$) (3.10) as

$$\begin{aligned} &filter_i^{\alpha+\beta}\left(X_1, \ldots, X_n, X_\epsilon\right) := \\ &filter_i^{\beta\circ\alpha}\left(X_1, \ldots, X_n, X_\epsilon\right) \cap filter_i^{\alpha\circ\beta}\left(X_1, \ldots, X_n, X_\epsilon\right). \end{aligned} \tag{3.135}$$

Lemma 3.1.38. *$filter_\epsilon^N$ and $filter_i^N$ (for some $i \in \mathcal{A}$) are the neutral elements w.r.t. filter intersection (3.5), (3.10).*

Proof. This directly follows from the definition of the neutral filters (3.2), (3.3). For some arbitrary filters $filter_\epsilon^\alpha$, $filter_i^\alpha$, $i \in \mathcal{A}$, $h \in \mathcal{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions_1}$, ..., $X_n \subseteq \overline{GActions_n}$, from Lemma 3.1.24 (particularly (3.71)) and (3.5), (3.10), we immediately get that

$$\begin{aligned} filter_\epsilon^{\alpha+N}\left(h, X_\epsilon, X_1, \ldots, X_n\right) = filter_\epsilon^{\alpha}\left(h, X_\epsilon, X_1, \ldots, X_n\right) \cap filter_\epsilon^{\alpha}\left(h, X_\epsilon, X_1, \ldots, X_n\right) = \\ filter_\epsilon^{\alpha}\left(h, X_\epsilon, X_1, \ldots, X_n\right) = filter_\epsilon^{N+\alpha}\left(h, X_\epsilon, X_1, \ldots, X_n\right) \\ \text{and} \\ filter_i^{\alpha+N}\left(X_1, \ldots, X_n, X_\epsilon\right) = filter_i^{\alpha}\left(X_1, \ldots, X_n, X_\epsilon\right) \cap filter_i^{\alpha}\left(X_1, \ldots, X_n, X_\epsilon\right) = \\ filter_i^{\alpha}\left(X_1, \ldots, X_n, X_\epsilon\right) = filter_i^{N+\alpha}\left(X_1, \ldots, X_n, X_\epsilon\right). \end{aligned} \tag{3.136}$$

□

Lemma 3.1.39. *Filter intersection (3.5), (3.10) preserves the basic filter property: For any $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ and arbitrary filters $filter_\epsilon^\alpha$, $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$, $filter_i^\beta$), it holds that*

$$\begin{aligned} filter_\epsilon^{\alpha+\beta}(h, X_\epsilon, X_1, \ldots, X_n) &\subseteq X_\epsilon \\ filter_i^{\alpha+\beta}(X_1, \ldots, X_n, X_\epsilon) &\subseteq X_i. \end{aligned} \tag{3.137}$$

Thus $filter_\epsilon^{\alpha+\beta}$ and $filter_i^{\alpha+\beta}$ are again filter functions.

Proof. Since filter intersection is defined as

$$\begin{gathered} filter_\epsilon^{\alpha+\beta}(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^{\alpha\circ\beta}(h, X_\epsilon, X_1, \ldots, X_n) \cap filter_\epsilon^{\beta\circ\alpha}(h, X_\epsilon, X_1, \ldots, X_n) \\ \text{and} \\ filter_i^{\alpha+\beta}(X_1, \ldots, X_n, X_\epsilon) = filter_i^{\alpha\circ\beta}(X_1, \ldots, X_n, X_\epsilon) \cap filter_i^{\beta\circ\alpha}(X_1, \ldots, X_n, X_\epsilon) \end{gathered} \tag{3.138}$$

by Lemma 3.1.25 and semantics of set intersection, the statement follows. □

Lemma 3.1.40. *Filter intersection (3.5), (3.10) preserves monotonicity (for a given downward closed domain): Given any two filter functions $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$ and $filter_i^\beta$ for some $i \in \mathcal{A}$), which are monotonic for a given downward closed domain ($PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$), the combined filter $filter_\epsilon^{\alpha+\beta}$ (respectively $filter_i^{\alpha+\beta}$) is also monotonic for this domain.*

Proof. By our assumption regarding monotonicity of the filters $filter_\epsilon^\alpha$, $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$, $filter_i^\beta$) for some downward closed domain by Lemma 3.1.26, we get that for $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$, $X'_\epsilon \subseteq X_\epsilon$, $X'_1 \subseteq X_1$, ..., $X'_n \subseteq X_n$

$$\begin{aligned} filter_\epsilon^{\alpha\circ\beta}(h, X'_\epsilon, X'_1, \ldots, X'_n) &\subseteq filter_\epsilon^{\alpha\circ\beta}(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_\epsilon^{\beta\circ\alpha}(h, X'_\epsilon, X'_1, \ldots, X'_n) &\subseteq filter_\epsilon^{\beta\circ\alpha}(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{\alpha\circ\beta}(X'_1, \ldots, X'_n, X'_\epsilon) &\subseteq filter_i^{\alpha\circ\beta}(X_1, \ldots, X_n, X_\epsilon) \\ filter_i^{\beta\circ\alpha}(X'_1, \ldots, X'_n, X'_\epsilon) &\subseteq filter_i^{\beta\circ\alpha}(X_1, \ldots, X_n, X_\epsilon). \end{aligned}$$

Therefore, by definition of filter intersection (3.5), (3.10) and semantics of set intersection,

$$\begin{aligned} &filter_\epsilon^{\beta\circ\alpha}(h, X'_\epsilon, X'_1, \ldots, X'_n) \cap filter_\epsilon^{\alpha\circ\beta}(h, X'_\epsilon, X'_1, \ldots, X'_n) \subseteq \\ &filter_\epsilon^{\beta\circ\alpha}(h, X_\epsilon, X_1, \ldots, X_n) \cap filter_\epsilon^{\alpha\circ\beta}(h, X_\epsilon, X_1, \ldots, X_n) \end{aligned}$$

and

$$\begin{aligned} &filter_i^{\beta\circ\alpha}(X'_1, \ldots, X'_n, X'_\epsilon) \cap filter_i^{\alpha\circ\beta}(X'_1, \ldots, X'_n, X'_\epsilon) \subseteq \\ &filter_i^{\beta\circ\alpha}(X_1, \ldots, X_n, X_\epsilon) \cap filter_i^{\alpha\circ\beta}(X_1, \ldots, X_n, X_\epsilon), \end{aligned}$$

from which the statement follows. □

Lemma 3.1.41. *Filter intersection (3.5), (3.10) preserves simple monotonicity (for given downward closed domain): Given any two filter functions* $filter_\epsilon^\alpha$ *and* $filter_\epsilon^\beta$ *(respectively* $filter_i^\alpha$ *and* $filter_i^\beta$ *for some* $i \in \mathcal{A}$*), which are simply monotonic for a given (downward closed) domain (*$PD_\epsilon \subseteq 2^{GEvents}$*,* $PD_1 \subseteq 2^{\overline{GActions}_1}$*, ...,* $PD_n \subseteq 2^{\overline{GActions}_n}$*), the combined filter* $filter_\epsilon^{\alpha+\beta}$ *(respectively* $filter_i^{\alpha+\beta}$*) is also simply monotonic for this domain.*

Proof. Analogous to the proof for Lemma 3.1.40. □

Lemma 3.1.42. *Filter intersection generally does not preserve idempotence (for some downward closed domain), even if the filter functions are monotonic: Given any two filter functions* $filter_\epsilon^\alpha$ *and* $filter_\epsilon^\beta$ *(respectively* $filter_i^\alpha$ *and* $filter_i^\beta$ *for some* $i \in \mathcal{A}$*), which are idempotent and monotonic for some given downward closed domain (*$PD_\epsilon \subseteq 2^{GEvents}$*,* $PD_1 \subseteq 2^{\overline{GActions}_1}$*, ...,* $PD_n \subseteq 2^{\overline{GActions}_n}$*), the combined filter* $filter_\epsilon^{\alpha+\beta}$ *(respectively* $filter_i^{\alpha+\beta}$*) is not necessarily idempotent for this domain.*

Proof. We provide the following counterexample for event filters, assuming $\{E_a^1, E_a^2, E_b^1, E_b^2\} \in PD_\epsilon$ and pairwise distinct elements in $\{E_a^1, E_a^2, E_b^1, E_b^2\}$. Using the monotonic and idempotent (by Lemma 3.1.29) filter functions from Definition 3.1.28 for any $h \in \mathscr{G}$ and sets $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$, the combined filter (3.5) gives the following result

$$filter_\epsilon^{\alpha^2+\beta^2}\left(h, \{E_a^1, E_a^2, E_b^1\}, X_1, \ldots, X_n\right) = \{E_a^1\}. \tag{3.139}$$

A second application on this resulting set however gives

$$filter_\epsilon^{\alpha^2+\beta^2}\left(h, \{E_a^1\}, X_1, \ldots, X_n\right) = \varnothing. \tag{3.140}$$

Thus we conclude that event filter intersection (3.5) generally does not preserve idempotence, even if the filter functions are both monotonic as well. A similar counterexample can be constructed for action filter intersection (3.10). □

Definition 3.1.43. We define the following event filter functions for $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$, where $E_a, E_a^c, E_a^b, E_a^{cb}, E_a^{bc}, E_b, E_b^c, E_b^a, E_b^{ca}, E_b^{ac}, E_c, E_c^a, E_c^b, E_c^{ab}, E_c^{ba} \in GEvents$

$$filter_\epsilon^{\overline{\alpha}}(h, X_\epsilon, X_1, \ldots, X_n) := \begin{cases} X_\epsilon \setminus \{E_a\} & E_b \in X_\epsilon \wedge E_c \in X_\epsilon \wedge E_b^c \in X_\epsilon \wedge E_c^b \in X_\epsilon \\ X_\epsilon \setminus \{E_a, E_a^b\} & E_b \notin X_\epsilon \wedge E_c \in X_\epsilon \wedge E_b^c \in X_\epsilon \wedge E_c^b \in X_\epsilon \\ X_\epsilon \setminus \{E_a, E_a^c\} & E_b \in X_\epsilon \wedge E_c \notin X_\epsilon \wedge E_b^c \in X_\epsilon \wedge E_c^b \in X_\epsilon \\ X_\epsilon \setminus \{E_a, E_a^c, E_a^b, E_a^{bc}\} & E_b \notin X_\epsilon \wedge E_c \notin X_\epsilon \wedge E_b^c \in X_\epsilon \wedge E_c^b \notin X_\epsilon \\ X_\epsilon \setminus \{E_a, E_a^c, E_a^b, E_a^{cb}\} & E_b \notin X_\epsilon \wedge E_c \notin X_\epsilon \wedge E_b^c \notin X_\epsilon \wedge E_c^b \in X_\epsilon \\ X_\epsilon \setminus \{E_a, E_a^c, E_a^b, E_a^{cb}, E_a^{bc}\} & E_b \notin X_\epsilon \wedge E_c \notin X_\epsilon \wedge E_b^c \notin X_\epsilon \wedge E_c^b \notin X_\epsilon \wedge E_a \in \\ X_\epsilon & E_b \notin X_\epsilon \wedge E_c \notin X_\epsilon \wedge E_b^c \notin X_\epsilon \wedge E_c^b \notin X_\epsilon \wedge E_a \notin \\ \varnothing & \text{otherwise,} \end{cases} \tag{3.141}$$

$$filter_{\epsilon}^{\overline{\beta}}(h, X_\epsilon, X_1, \ldots, X_n) := \begin{cases} X_\epsilon \setminus \{E_b\} & E_a \in X_\epsilon \wedge E_c \in X_\epsilon \wedge E_a^c \in X_\epsilon \wedge E_c^a \in X_\epsilon \\ X_\epsilon \setminus \{E_b, E_b^a\} & E_a \notin X_\epsilon \wedge E_c \in X_\epsilon \wedge E_a^c \in X_\epsilon \wedge E_c^a \in X_\epsilon \\ X_\epsilon \setminus \{E_b, E_b^c\} & E_a \in X_\epsilon \wedge E_c \notin X_\epsilon \wedge E_a^c \in X_\epsilon \wedge E_c^a \in X_\epsilon \\ X_\epsilon \setminus \{E_b, E_b^c, E_b^a, E_b^{ac}\} & E_a \notin X_\epsilon \wedge E_c \notin X_\epsilon \wedge E_a^c \in X_\epsilon \wedge E_c^a \notin X_\epsilon \\ X_\epsilon \setminus \{E_b, E_b^c, E_b^a, E_b^{ca}\} & E_a \notin X_\epsilon \wedge E_c \notin X_\epsilon \wedge E_a^c \notin X_\epsilon \wedge E_c^a \in X_\epsilon \\ X_\epsilon \setminus \{E_b, E_b^c, E_b^a, E_b^{ca}, E_b^{ac}\} & E_a \notin X_\epsilon \wedge E_c \notin X_\epsilon \wedge E_a^c \notin X_\epsilon \wedge E_c^a \notin X_\epsilon \wedge E_b \in \\ X_\epsilon & E_a \notin X_\epsilon \wedge E_c \notin X_\epsilon \wedge E_a^c \notin X_\epsilon \wedge E_c^a \notin X_\epsilon \wedge E_b \notin \\ \varnothing & \text{otherwise,} \end{cases} \tag{3.142}$$

$$filter_{\epsilon}^{\overline{\gamma}}(h, X_\epsilon, X_1, \ldots, X_n) := \begin{cases} X_\epsilon \setminus \{E_c\} & E_b \in X_\epsilon \wedge E_a \in X_\epsilon \wedge E_b^a \in X_\epsilon \wedge E_a^b \in X_\epsilon \\ X_\epsilon \setminus \{E_c, E_c^b\} & E_b \notin X_\epsilon \wedge E_a \in X_\epsilon \wedge E_b^a \in X_\epsilon \wedge E_a^b \in X_\epsilon \\ X_\epsilon \setminus \{E_c, E_c^a\} & E_b \in X_\epsilon \wedge E_a \notin X_\epsilon \wedge E_b^a \in X_\epsilon \wedge E_a^b \in X_\epsilon \\ X_\epsilon \setminus \{E_c, E_c^a, E_c^b, E_c^{ba}\} & E_b \notin X_\epsilon \wedge E_a \notin X_\epsilon \wedge E_b^a \in X_\epsilon \wedge E_a^b \notin X_\epsilon \\ X_\epsilon \setminus \{E_c, E_c^a, E_c^b, E_c^{ab}\} & E_b \notin X_\epsilon \wedge E_a \notin X_\epsilon \wedge E_b^a \notin X_\epsilon \wedge E_a^b \in X_\epsilon \\ X_\epsilon \setminus \{E_c, E_c^a, E_c^b, E_c^{ab}, E_c^{ba}\} & E_b \notin X_\epsilon \wedge E_a \notin X_\epsilon \wedge E_b^a \notin X_\epsilon \wedge E_a^b \notin X_\epsilon \wedge E_c \in \\ X_\epsilon & E_b \notin X_\epsilon \wedge E_a \notin X_\epsilon \wedge E_b^a \notin X_\epsilon \wedge E_a^b \notin X_\epsilon \wedge E_c \notin \\ \varnothing & \text{otherwise.} \end{cases} \tag{3.143}$$

Lemma 3.1.44. *The functions $filter_{\epsilon}^{\overline{\alpha}}$, $filter_{\epsilon}^{\overline{\beta}}$, $filter_{\epsilon}^{\overline{\gamma}}$ from Definition 3.1.43 are event filters and idempotent for the domain $2^{GEvents}$, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ under the assumption that all elements in $\{E_a, E_a^c, E_a^b, E_a^{cb}, E_a^{bc}, E_b, E_b^c, E_b^a, E_b^{ca}, E_b^{ac}, E_c, E_c^a, E_c^b, E_c^{ab}, E_c^{ba}\}$ are pairwise distinct.*

Proof. The three filters' adherence to the basic filter property follows from their Definition 3.1.43.

Regarding idempotence the only non-trivial case for $filter_{\epsilon}^{\overline{\alpha}}$ occurs if X_ϵ is such that $E_b \notin X_\epsilon \wedge E_c \notin X_\epsilon \wedge E_b^c \notin X_\epsilon \wedge E_c^b \notin X_\epsilon \wedge E_a \in X_\epsilon$. For any $h \in \mathscr{G}$ and $X_1 \subseteq \overline{GActions_1}$, ..., $X_n \subseteq \overline{GActions_n}$

$$\begin{aligned} &filter_{\epsilon}^{\overline{\alpha}}\left(h, filter_{\epsilon}^{\overline{\alpha}}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right) = \\ &filter_{\epsilon}^{\overline{\alpha}}\left(h, X_\epsilon \setminus \{E_a, E_a^c, E_a^b, E_a^{cb}, E_a^{bc}\}, X_1, \ldots, X_n\right) = X_\epsilon \setminus \{E_a, E_a^c, E_a^b, E_a^{cb}, E_a^{bc}\}. \end{aligned}$$

The reasoning for $filter_{\epsilon}^{\overline{\beta}}$ and $filter_{\epsilon}^{\overline{\gamma}}$ are analogous. □

Lemma 3.1.45. *Filter intersection is generally not associative, even for idempotent filter functions.*

Proof. We construct the following counterexample using the three event filter functions (3.141), (3.142), (3.143). Their idempotence follows from Lemma 3.1.44. For

$$X_\epsilon = \{E_a, E_a^c, E_a^b, E_a^{cb}, E_a^{bc}, E_b, E_b^c, E_b^a, E_b^{ca}, E_b^{ac}, E_c, E_c^a, E_c^b, E_c^{ab}, E_c^{ba}\} \tag{3.144}$$

(where we assume that all elements in X_ϵ are pairwise distinct) and any history $h \in \mathscr{G}$ and sets $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$, after tedious but simple computations using

$$\begin{aligned}\overline{\alpha} + (\overline{\beta} + \overline{\gamma}) &= \overline{\alpha} \circ \overline{\beta} \circ \overline{\gamma} \cap \overline{\alpha} \circ \overline{\gamma} \circ \overline{\beta} \cap \overline{\beta} \circ \overline{\gamma} \circ \overline{\alpha} \cap \overline{\gamma} \circ \overline{\beta} \circ \overline{\alpha} \\ (\overline{\alpha} + \overline{\beta}) + \overline{\gamma} &= \overline{\alpha} \circ \overline{\beta} \circ \overline{\gamma} \cap \overline{\beta} \circ \overline{\alpha} \circ \overline{\gamma} \cap \overline{\gamma} \circ \overline{\alpha} \circ \overline{\beta} \cap \overline{\gamma} \circ \overline{\beta} \circ \overline{\alpha}\end{aligned}$$

we get

$$filter_\epsilon^{\overline{\alpha}+(\overline{\beta}+\overline{\gamma})}(h, X_\epsilon, X_1, \ldots, X_n) = \{E_b^{ca}, E_c^{ba}\} \tag{3.145}$$

and

$$filter_\epsilon^{(\overline{\alpha}+\overline{\beta})+\overline{\gamma}}(h, X_\epsilon, X_1, \ldots, X_n) = \{E_a^{bc}, E_b^{ac}\}. \tag{3.146}$$

Thus we conclude that filter intersection is not associative even for idempotent filters. □

Lemma 3.1.46. *Filter intersection is commutative.*

Proof. From definition of filter intersection (3.5), (3.10), the lemma follows by commutativity of set intersection. □

Lemma 3.1.47. *For two filter functions $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$ for $i \in \mathcal{A}$) and $filter_\epsilon^\beta$ (respectively $filter_i^\beta$), where for some downward closed domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$, $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$) is stricter than $filter_\epsilon^\beta$ (respectively $filter_i^\beta$), idempotent and simply monotonic, the combined filter $filter_\epsilon^{\alpha+\beta}$ (respectively $filter_i^{\alpha+\beta}$) simplifies to*

$$\begin{aligned}filter_\epsilon^{\alpha+\beta}(h, X_\epsilon, X_1, \ldots, X_n) &= filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{\alpha+\beta}(X_1, \ldots, X_n, X_\epsilon) &= filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon)\end{aligned} \tag{3.147}$$

for this domain.

Proof. Follows directly from Lemmas 3.1.35 and 3.1.36. □

Corollary 3.1.48. *For two filter functions $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$ for $i \in \mathcal{A}$) and $filter_\epsilon^\beta$ (respectively $filter_i^\beta$), where for some downward closed domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$, $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$) is equal to $filter_\epsilon^\beta$ (respectively $filter_i^\beta$) and idempotent, the combined filter $filter_\epsilon^{\alpha+\beta}$ (respectively $filter_i^{\alpha+\beta}$) simplifies to*

$$\begin{aligned}filter_\epsilon^{\alpha+\beta}(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^\beta(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{\alpha+\beta}(X_1, \ldots, X_n, X_\epsilon) = filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon) = filter_i^\beta(X_1, \ldots, X_n, X_\epsilon)\end{aligned} \tag{3.148}$$

in this domain.

Proof. By definition of filter intersection (3.5), (3.10) the statement immediately follows from Corollary 3.1.37 and semantics of set intersection. □

3.1.6 k-Filter Intersection

To reiterate, given two event filter functions $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$ for some $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$, we defined *k-filter intersection* for $k \in \mathbb{N} \setminus \{0\}$ (3.6) as

$$filter_\epsilon^{k\cdot(\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) := filter_\epsilon^{\alpha+\beta}\left(h, \underbrace{\left(\ldots\left(filter_\epsilon^{\alpha+\beta}(h, X_\epsilon, X_1, \ldots, X_n)\right)\ldots\right)}_{k-1 \text{ times}}, X_1, \ldots, X_n\right) \tag{3.149}$$

and 0-*filter intersection* (3.7) as

$$filter_\epsilon^{0\cdot(\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) := filter_\epsilon^{N}(h, X_\epsilon, X_1, \ldots, X_n). \tag{3.150}$$

For two action filter functions $filter_i^\alpha$ and $filter_i^\beta$ (for $i \in \mathcal{A}$) we defined *k-filter intersection* (3.11) as

$$filter_i^{k\cdot(\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon) := filter_i^{\alpha+\beta}\left(X_1, \ldots, X_{i-1}, \underbrace{\left(\ldots\left(filter_i^{\alpha+\beta}(X_1, \ldots, X_n, X_\epsilon)\right)\ldots\right)}_{k-1 \text{ times}}, X_{i+1}, \ldots, X_n, X_\epsilon\right) \tag{3.151}$$

and 0-*filter intersection* (3.12) as

$$filter_i^{0\cdot(\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon) := filter_i^{N}(X_1, \ldots, X_n, X_\epsilon). \tag{3.152}$$

Lemma 3.1.49. *$filter_\epsilon^N$ (respectively $filter_i^N$ for some $i \in \mathcal{A}$) is the neutral element w.r.t. k-filter intersection (for $k \geq 2$) for some $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$) in the domain ($PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$) if and only if $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$) is idempotent in this domain.*

Proof. Direction "⇒":
Suppose for some arbitrary filter $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$ for $i \in \mathcal{A}$) and $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$

$$\begin{aligned} filter_\epsilon^{k\cdot(\alpha+N)}(h, X_\epsilon, X_1, \ldots, X_n) &= filter_\epsilon^{\alpha}(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{k\cdot(\alpha+N)}(X_1, \ldots, X_n, X_\epsilon) &= filter_i^{\alpha}(X_1, \ldots, X_n, X_\epsilon). \end{aligned} \tag{3.153}$$

Since k-filter intersection (3.6), (3.11) for $k \geq 2$ is defined as compositions of filter intersection, by Definition 3.1.7 (basic filter property) $filter_\epsilon^\alpha$ and $filter_i^\alpha$ have to be idempotent in the domain PD_ϵ, PD_1, ..., PD_n, as otherwise

$$\begin{aligned} &filter_\epsilon^{k\cdot(\alpha+N)}(h, X_\epsilon, X_1, \ldots, X_n) = \text{ (by Lemma 3.1.38)} \\ &filter_\epsilon^{k\cdot\alpha}(h, X_\epsilon, X_1, \ldots, X_n) \subset filter_\epsilon^{\alpha}(h, X_\epsilon, X_1, \ldots, X_n) \end{aligned} \tag{3.154}$$

and similarly for action filters

$$\begin{aligned} &filter_i^{k\cdot(\alpha+N)}\left(X_1,\ldots,X_n,X_\epsilon\right) = \text{ (by Lemma 3.1.38)} \\ &filter_i^{k\cdot\alpha}\left(X_1,\ldots,X_n,X_\epsilon\right) \subset filter_i^{\alpha}\left(X_1,\ldots,X_n,X_\epsilon\right). \end{aligned} \tag{3.155}$$

would hold, which however violates our assumption (3.153).

Direction "⇐":
Suppose the filters $filter_\epsilon^\alpha$, $filter_i^\alpha$ for $i \in \mathcal{A}$ are idempotent in the domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$. By Lemma 3.1.38 and (3.6), (3.11), k-filter intersection boils down to nesting $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$) $k-1$ times. However, since $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$) were assumed to be idempotent, it follows that for any $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$

$$(\forall l \in \mathbb{N} \setminus \{0,1\})\ filter_\epsilon^{\alpha+N}\left(h, \underbrace{\left(\ldots\left(filter_\epsilon^{\alpha+N}\left(h, X_\epsilon, X_1, \ldots, X_n\right)\right)\ldots\right)}_{l-1 \text{ times}}, X_1, \ldots, X_n\right) =$$
$$filter_\epsilon^{\alpha}\left(h, X_\epsilon, X_1, \ldots, X_n\right)$$

and

$$(\forall l \in \mathbb{N} \setminus \{0,1\})\ filter_i^{\alpha+N}\left(X_1,\ldots,X_{i-1}, \underbrace{\left(\ldots\left(filter_i^{\alpha+N}\left(X_1,\ldots,X_n,X_\epsilon\right)\right)\ldots\right)}_{l-1 \text{ times}}, X_{i+1}, \ldots, X_n, X_\epsilon\right) =$$
$$filter_i^{\alpha}\left(X_1,\ldots,X_n,X_\epsilon\right).$$

□

Lemma 3.1.50. *k-filter intersection (3.6), (3.11), (3.7), (3.12) preserves the basic filter property: For any $k \in \mathbb{N}$, $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ and arbitrary filters $filter_\epsilon^\alpha$, $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$, $filter_i^\beta$) it holds that*

$$\begin{aligned} filter_\epsilon^{k\cdot(\alpha+\beta)}\left(h, X_\epsilon, X_1, \ldots, X_n\right) &\subseteq X_\epsilon \\ filter_i^{k\cdot(\alpha+\beta)}\left(X_1,\ldots,X_n,X_\epsilon\right) &\subseteq X_i. \end{aligned} \tag{3.156}$$

Thus, $filter_\epsilon^{k\cdot(\alpha+\beta)}$ and $filter_i^{k\cdot(\alpha+\beta)}$ are again filter functions.

Proof. For $k \geq 1$,

$$filter_\epsilon^{k\cdot(\alpha+\beta)}\left(h, X_\epsilon, X_1, \ldots, X_n\right) =$$
$$filter_\epsilon^{\alpha+\beta}\left(h, \underbrace{\left(\ldots\left(filter_\epsilon^{\alpha+\beta}\left(h, X_\epsilon, X_1, \ldots, X_n\right)\right)\ldots\right)}_{k-1 \text{ times}}, X_1, \ldots, X_n\right)$$

and

$$filter_i^{k\cdot(\alpha+\beta)}(X_1,\ldots,X_n,X_\epsilon) = filter_i^{\alpha+\beta}\left(X_1,\ldots,X_{i-1},\underbrace{\left(\ldots\left(filter_i^{\alpha+\beta}(X_1,\ldots,X_n,X_\epsilon)\right)\ldots\right)}_{k-1 \text{ times}},X_{i+1},\ldots,X_n,X_\epsilon\right)$$

A trivial induction using Lemma 3.1.39 and 3.1.9 in the induction step establishes (3.156). For $k=0$, it follows trivially from

$$filter_\epsilon^{0\cdot(\alpha+\beta)}(h,X_\epsilon,X_1,\ldots,X_n) = filter_\epsilon^{N}(h,X_\epsilon,X_1,\ldots,X_n) = X_\epsilon$$
$$filter_i^{0\cdot(\alpha+\beta)}(X_1,\ldots,X_n,X_\epsilon) = filter_i^{N}(X_1,\ldots,X_n,X_\epsilon) = X_i.$$

□

Lemma 3.1.51. *k-filter intersection (for some $k\in\mathbb{N}$) (3.6), (3.11), (3.7), (3.12) preserves monotonicity (for a given downward closed domain): Given any two filter functions $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$ and $filter_i^\beta$ for some $i\in\mathcal{A}$), which are monotonic for a given (downward closed) domain ($PD_\epsilon\subseteq 2^{GEvents}$, $PD_1\subseteq 2^{\overline{GActions_1}}$, ..., $PD_n\subseteq 2^{\overline{GActions_n}}$), the combined filter $filter_\epsilon^{k\cdot(\alpha+\beta)}$ (respectively $filter_i^{k\cdot(\alpha+\beta)}$) is also monotonic for this domain.*

Proof. We start by examining event filters and use induction on k. **Induction Hypothesis:** For any $h\in\mathscr{G}$, $X_\epsilon\in PD_\epsilon$, $X_1\in PD_1$, ..., $X_n\in PD_n$ and $X'_\epsilon\subseteq X_\epsilon$, $X'_1\subseteq X_1$, ..., $X'_n\subseteq X_n$ and $k\geq 1$

$$filter_\epsilon^{k\cdot(\alpha+\beta)}(h,X'_\epsilon,X'_1,\ldots,X'_n)\subseteq filter_\epsilon^{k\cdot(\alpha+\beta)}(h,X_\epsilon,X_1,\ldots,X_n) \tag{3.157}$$

Base Case: If $k=0$, then monotonicity follows from Lemma 3.1.16 as for any $h\in\mathscr{G}$, $X_\epsilon\in PD_\epsilon$, $X_1\in PD_1$, ..., $X_n\in PD_n$ and $X'_\epsilon\subseteq X_\epsilon$, $X'_1\subseteq X_1$, ..., $X'_n\subseteq X_n$

$$filter_\epsilon^{N}(h,X_\epsilon,X_1,\ldots,X_n) = filter_\epsilon^{0\cdot(\alpha+\beta)}(h,X_\epsilon,X_1,\ldots,X_n). \tag{3.158}$$

Induction Step: Suppose the induction hypothesis (3.157) holds for k, then for any $h\in\mathscr{G}$, $X_\epsilon\in PD_\epsilon$, $X_1\in PD_1$, ..., $X_n\in PD_n$ and $X'_\epsilon\subseteq X_\epsilon$, $X'_1\subseteq X_1$, ..., $X'_n\subseteq X_n$ and $k\geq 1$ by monotonicity of $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$ and Lemma 3.1.40

$$\begin{aligned} &filter_\epsilon^{\alpha+\beta}\left(h,filter_\epsilon^{k\cdot(\alpha+\beta)}(h,X'_\epsilon,X'_1,\ldots,X'_n),X'_1,\ldots,X'_n\right)\subseteq\\ &filter_\epsilon^{\alpha+\beta}\left(h,filter_\epsilon^{k\cdot(\alpha+\beta)}(h,X_\epsilon,X_1,\ldots,X_n),X_1,\ldots,X_n\right). \end{aligned} \tag{3.159}$$

Since

$$filter_\epsilon^{(k+1)\cdot(\alpha+\beta)}(h,X'_\epsilon,X'_1,\ldots,X'_n) = filter_\epsilon^{\alpha+\beta}\left(h,filter_\epsilon^{k\cdot(\alpha+\beta)}(h,X'_\epsilon,X'_1,\ldots,X'_n),X'_1,\ldots,X'_n\right)$$

and

$$filter_\epsilon^{(k+1)\cdot(\alpha+\beta)}(h,X_\epsilon,X_1,\ldots,X_n) = filter_\epsilon^{\alpha+\beta}\left(h,filter_\epsilon^{k\cdot(\alpha+\beta)}(h,X_\epsilon,X_1,\ldots,X_n),X_1,\ldots,X_n\right), \tag{3.160}$$

this completes the induction step. The reasoning for action filters is completely analogous. □

Corollary 3.1.52. *k-filter intersection (for some $k \in \mathbb{N}$) (3.6), (3.11), (3.7), (3.12) preserves simple monotonicity (for a given downward closed domain): Given any two filter functions $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$ and $filter_i^\beta$ for some $i \in \mathcal{A}$), which are simply monotonic for a given downward closed domain ($PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$), the combined filter $filter_\epsilon^{k\cdot(\alpha+\beta)}$ (respectively $filter_i^{k\cdot(\alpha+\beta)}$) is also simply monotonic for this domain.*

Proof. Analogous to the proof of Lemma 3.1.51. □

Lemma 3.1.53. *For some $k \geq 1$, any $h \in \mathscr{G}$, $X_\epsilon = \{E_a^1, E_a^2, \ldots, E_a^k, E_b^2, \ldots, E_b^k\}$ (assuming $\{E_a^1, E_a^2, \ldots, E_a^k, E_b^1, E_b^2, \ldots, E_b^k\} \subseteq GEvents$ with distinct elements in $\{E_a^1, E_a^2, \ldots, E_a^k, E_b^1, E_b^2, \ldots, E_b^k\}$), arbitrary sets $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$, and $0 \leq \widetilde{k} < k$, the result of the combined filter $filter_\epsilon^{\widetilde{k}\cdot(\alpha^k+\beta^k)}$ is*

$$filter_\epsilon^{\widetilde{k}\cdot(\alpha^k+\beta^k)}(h, X_\epsilon, X_1, \ldots, X_n) = \{E_a^{\widetilde{k}+1}, E_a^{\widetilde{k}+2}, \ldots, E_a^k, E_b^{\widetilde{k}+2}, E_b^{\widetilde{k}+3}, \ldots, E_b^k\}. \tag{3.161}$$

Proof. For $\widetilde{k} = 0$ we get

$$filter_\epsilon^{0\cdot(\alpha^k+\beta^k)}(h, X_\epsilon, X_1, \ldots, X_n) = \{E_a^1, E_a^2, \ldots, E_a^k, E_b^2, E_b^3, \ldots, E_b^k\} \tag{3.162}$$

For $\widetilde{k} \geq 1$ we use induction.
Base Case: By using Definition 3.1.28 we obtain

$$filter_\epsilon^{1\cdot(\alpha^k+\beta^k)}(h, X_\epsilon, X_1, \ldots, X_n) = \{E_a^2, E_a^3, \ldots, E_a^k, E_b^3, E_b^4, \ldots, E_b^k\}. \tag{3.163}$$

Induction Step: Suppose the induction hypothesis (3.161) holds for some $\widetilde{k} < k-1$. By definition of $\widetilde{k}$-filter intersection (3.6), the $\widetilde{k}+1$-intersected filter results in

$$\begin{gathered}
filter_\epsilon^{(\widetilde{k}+1)\cdot(\alpha^k+\beta^k)}(h, X_\epsilon, X_1, \ldots, X_n) = \\
filter_\epsilon^{\alpha^k+\beta^k}\left(h, filter_\epsilon^{\widetilde{k}\cdot(\alpha^k+\beta^k)}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right) = \\
\text{(by the induction hypothesis (3.161))} \\
filter_\epsilon^{\alpha^k+\beta^k}\left(h, \{E_a^{\widetilde{k}+1}, \ldots, E_a^k, E_b^{\widetilde{k}+2}, \ldots, E_b^k\}, X_1, \ldots, X_n\right) = \text{ (by (3.91), (3.92))} \\
filter_\epsilon^{\alpha^k}\left(h, filter_\epsilon^{\beta^k}\left(h, \{E_a^{\widetilde{k}+1}, \ldots, E_a^k, E_b^{\widetilde{k}+2}, \ldots, E_b^k\}, X_1, \ldots, X_n\right), X_1, \ldots, X_n\right) \cap \\
filter_\epsilon^{\beta^k}\left(h, filter_\epsilon^{\alpha^k}\left(h, \{E_a^{\widetilde{k}+1}, \ldots, E_a^k, E_b^{\widetilde{k}+2}, \ldots, E_b^k\}, X_1, \ldots, X_n\right), X_1, \ldots, X_n\right) = \\
filter_\epsilon^{\alpha^k}\left(h, \{E_a^{\widetilde{k}+1}, \ldots, E_a^k, E_b^{\widetilde{k}+2}, \ldots, E_b^k\}, X_1, \ldots, X_n\right) \cap \\
filter_\epsilon^{\beta^k}\left(h, \{E_a^{\widetilde{k}+2}, \ldots, E_a^k, E_b^{\widetilde{k}+2}, \ldots, E_b^k\}, X_1, \ldots, X_n\right) = \\
\{E_a^{\widetilde{k}+2}, \ldots, E_a^k, E_b^{\widetilde{k}+2}, \ldots, E_b^k\} \cap \{E_a^{\widetilde{k}+2}, \ldots, E_a^k, E_b^{\widetilde{k}+3}, \ldots, E_b^k\} = \\
\{E_a^{\widetilde{k}+2}, \ldots, E_a^k, E_b^{\widetilde{k}+3}, \ldots, E_b^k\} = \{E_a^{(\widetilde{k}+1)+1}, \ldots, E_a^k, E_b^{(\widetilde{k}+1)+2}, \ldots, E_b^k\}.
\end{gathered} \tag{3.164}$$

This completes the induction step. □

Lemma 3.1.54. *k-filter intersection for some $k \geq 1$ does not necessarily preserve idempotence (for some downward closed domain), even if the filter functions are monotonic: There does not exists a number $k \in \mathbb{N} \setminus \{0\}$ s.t. for any two filter functions $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$ and $filter_i^\beta$ for some $i \in \mathcal{A}$), which are idempotent and monotonic for some given downward closed domain ($PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$), the combined filter $filter_\epsilon^{k\cdot(\alpha+\beta)}$ (respectively $filter_i^{k\cdot(\alpha+\beta)}$) is also idempotent for this domain.*

Proof. We use the filter functions $filter_\epsilon^{\alpha^{k'}}$ (3.91) and $filter_\epsilon^{\beta^{k'}}$ (3.92) with $k' = k+1$ as our counterexample. By Lemma 3.1.29, both are monotonic and idempotent. Suppose by contradiction that there exists some number $k \geq 1$ s.t. the k-filter intersection of any two filters results in an idempotent filter. When applying the k-combined filter $filter_\epsilon^{k\cdot(\alpha^{k'}+\beta^{k'})}$ to the set $X_\epsilon = \{E_a^1, E_a^2, \ldots, E_a^{k'}, E_b^2, \ldots, E_b^{k'}\}$ (assuming $\{E_a^1, E_a^2, \ldots, E_a^{k'}, E_b^1, E_b^2, \ldots, E_b^{k'}\} \subseteq GEvents$ and pairwise distinct elements in $\{E_a^1, E_a^2, \ldots, E_a^k, E_b^1, E_b^2, \ldots, E_b^k\}$), for any $h \in \mathscr{G}$, $X_1 \subseteq \overline{GActions_1}$, ..., $X_n \subseteq \overline{GActions_n}$, by Lemma 3.1.53, the result is

$$filter_\epsilon^{k\cdot(\alpha^{k'}+\beta^{k'})}(h, X_\epsilon, X_1, \ldots, X_n) = \{E_a^{k'}\}. \tag{3.165}$$

However, applying $filter_\epsilon^{k\cdot(\alpha^{k'}+\beta^{k'})}$ again leads to

$$filter_\epsilon^{k\cdot(\alpha^{k'}+\beta^{k'})}\left(h, \{E_a^{k'}\}, X_1, \ldots, X_n\right) = \varnothing. \tag{3.166}$$

Therefore $filter_\epsilon^{k\cdot(\alpha^{k+1}+\beta^{k+1})}$ is not idempotent for any $k \geq 1$. □

Lemma 3.1.55. *k-filter intersection is generally not associative for any $k \in \mathbb{N} \setminus \{0\}$.*

Proof. Using the filters from Definition 3.1.43 and the event set

$$X_\epsilon = \{E_a, E_a^c, E_a^b, E_a^{cb}, E_a^{bc}, E_b, E_b^c, E_b^a, E_b^{ca}, E_b^{ac}, E_c, E_c^a, E_c^b, E_c^{ab}, E_c^{ba}\}, \tag{3.167}$$

for some $k > 1$ we get that

$$filter_\epsilon^{k\cdot(\overline{\beta}+\overline{\gamma})}(h, X_\epsilon, X_1, \ldots, X_n) = X_\epsilon \setminus \{E_b, E_c, E_c^b, E_b^c\}, \tag{3.168}$$

thus

$$filter_\epsilon^{k\cdot(\overline{\beta}+\overline{\gamma})\circ\overline{\alpha}}(h, X_\epsilon, X_1, \ldots, X_n) = X_\epsilon \setminus \{E_b, E_c, E_c^b, E_b^c, E_a, E_a^c, E_a^b, E_a^{cb}, E_a^{bc}\}. \tag{3.169}$$

Also

$$filter_\epsilon^{\overline{\alpha}}(h, X_\epsilon, X_1, \ldots, X_n) = X_\epsilon \setminus \{E_a\} \tag{3.170}$$

and

$$filter_\epsilon^{\overline{\alpha}\circ k\cdot(\overline{\beta}+\overline{\gamma})}(h, X_\epsilon, X_1, \ldots, X_n) = X_\epsilon \setminus \{E_a, E_b, E_c, E_b^a, E_c^a, E_b^{ac}, E_c^{ab}\}. \tag{3.171}$$

Therefore

$$filter_\epsilon^{k\cdot(\overline{\alpha}+k\cdot(\overline{\beta}+\overline{\gamma}))}(h, X_\epsilon, X_1, \ldots, X_n) = \{E_b^{ca}, E_c^{ba}\}. \tag{3.172}$$

Regarding $filter_\epsilon^{k\cdot(k\cdot(\overline{\alpha}+\overline{\beta})+\overline{\gamma})}$

$$filter_\epsilon^{k\cdot(\overline{\alpha}+\overline{\beta})}\,(h, X_\epsilon, X_1, \ldots, X_n) = X_\epsilon \setminus \{E_a, E_b, E_b^a, E_a^b\}, \tag{3.173}$$

thus

$$filter_\epsilon^{k\cdot(\overline{\alpha}+\overline{\beta})\circ\overline{\gamma}}\,(h, X_\epsilon, X_1, \ldots, X_n) = X_\epsilon \setminus \{E_a, E_b, E_b^a, E_a^b, E_c, E_c^a, E_c^b, E_c^{ab}, E_c^{ba}\}. \tag{3.174}$$

Also

$$filter_\epsilon^{\overline{\gamma}}\,(h, X_\epsilon, X_1, \ldots, X_n) = X_\epsilon \setminus \{E_c\} \tag{3.175}$$

and

$$filter_\epsilon^{\overline{\gamma}\circ k\cdot(\overline{\alpha}+\overline{\beta})}\,(h, X_\epsilon, X_1, \ldots, X_n) = X_\epsilon \setminus \{E_a, E_b, E_c, E_a^c, E_b^c, E_b^{ca}, E_a^{cb}\}. \tag{3.176}$$

Therefore

$$filter_\epsilon^{k\cdot(k\cdot(\overline{\alpha}+\overline{\beta})+\overline{\gamma})}\,(h, X_\epsilon, X_1, \ldots, X_n) = \{E_a^{bc}, E_b^{ac}\}. \tag{3.177}$$

□

Lemma 3.1.56. *k-filter intersection is commutative.*

Proof. Since by (3.6) and (3.11) k-filter intersection is just k compositions of the same filter intersection (3.5), (3.10) and filter intersection is commutative, so is k-filter intersection. More specifically, for two filters $filter_\epsilon^\alpha$, $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$, $filter_i^\beta$ for $i \in \mathcal{A}$) and some $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$, $k \geq 1$

$$\begin{aligned}
&filter_\epsilon^{\alpha+\beta}\,(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^{\beta+\alpha}\,(h, X_\epsilon, X_1, \ldots, X_n)\\
&\Longrightarrow\\
&filter_\epsilon^{k\cdot(\alpha+\beta)}\,(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^{\alpha+\beta}\left(h, filter_\epsilon^{\alpha+\beta}(\ldots)X_1, \ldots, X_n\right) =\\
&filter_\epsilon^{\beta+\alpha}\left(h, filter_\epsilon^{\beta+\alpha}(\ldots), X_1, \ldots, X_n\right) = filter_\epsilon^{k\cdot(\beta+\alpha)}\,(h, X_\epsilon, X_1, \ldots, X_n)
\end{aligned} \tag{3.178}$$

and equivalently for action filters

$$\begin{aligned}
&filter_i^{\alpha+\beta}\,(X_1, \ldots, X_n, X_\epsilon) = filter_i^{\beta+\alpha}\,(X_1, \ldots, X_n, X_\epsilon)\\
&\Longrightarrow\\
&filter_i^{k\cdot(\alpha+\beta)}\,(X_1, \ldots, X_n, X_\epsilon) =\\
&filter_i^{\alpha+\beta}\left(X_1, \ldots, X_{i-1}, filter_i^{\alpha+\beta}(\ldots), X_{i+1}, \ldots, X_n, X_\epsilon\right) =\\
&filter_i^{\beta+\alpha}\left(X_1, \ldots, X_{i-1}, filter_i^{\beta+\alpha}(\ldots), X_{i+1}, \ldots, X_n, X_\epsilon\right) =\\
&filter_i^{k\cdot(\beta+\alpha)}\,(X_1, \ldots, X_n, X_\epsilon)
\end{aligned} \tag{3.179}$$

The case $k = 0$ is even more trivial, as

$$filter_\epsilon^{0\cdot(\alpha+\beta)}\,(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^{N}\,(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^{0\cdot(\beta+\alpha)}\,(h, X_\epsilon, X_1, \ldots, X_n)$$

and

$$filter_i^{0\cdot(\alpha+\beta)}\,(X_1, \ldots, X_n, X_\epsilon) = filter_i^{N}\,(X_1, \ldots, X_n, X_\epsilon) = filter_i^{0\cdot(\beta+\alpha)}\,(X_1, \ldots, X_n, X_\epsilon)\,. \tag{3.180}$$

□

Lemma 3.1.57. *Given filter functions* $filter_\epsilon^\alpha$, $filter_\epsilon^\beta$ *(respectively* $filter_i^\alpha$, $filter_i^\beta$ *for* $i \in \mathcal{A}$*) for some* $k, k' \in \mathbb{N}$, *where* $k' \leq k$, $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$, *it holds that*

$$\begin{aligned} filter_\epsilon^{k\cdot(\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) &\subseteq filter_\epsilon^{k'\cdot(\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{k\cdot(\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon) &\subseteq filter_i^{k'\cdot(\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon) \end{aligned} \tag{3.181}$$

Proof. Immediately follows from the definition of k-filter intersection (3.6), (3.11), Lemmas 3.1.39 and 3.1.9. □

Lemma 3.1.58. *For two filter functions* $filter_\epsilon^\alpha$ *(respectively* $filter_i^\alpha$ *for some* $i \in \mathcal{A}$*) and* $filter_\epsilon^\beta$ *(respectively* $filter_i^\beta$*), some* $k \geq 1$, *where for some downward closed domain* $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$ $filter_\epsilon^\alpha$ *(respectively* $filter_i^\alpha$*) is stricter than* $filter_\epsilon^\beta$ *(respectively* $filter_i^\beta$*), idempotent and simply monotonic, the combined filter* $filter_\epsilon^{k\cdot(\alpha+\beta)}$ *(respectively* $filter_i^{k\cdot(\alpha+\beta)}$*) simplifies to*

$$\begin{aligned} filter_\epsilon^{k\cdot(\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) &= filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{k\cdot(\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon) &= filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon) \end{aligned} \tag{3.182}$$

for this domain.

Proof. From Lemma 3.1.47, it follows that for any $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$

$$\begin{aligned} filter_\epsilon^{\alpha+\beta}(h, X_\epsilon, X_1, \ldots, X_n) &= filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{\alpha+\beta}(X_1, \ldots, X_n, X_\epsilon) &= filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon). \end{aligned} \tag{3.183}$$

Further by idempotence of $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$), we get that for any $k \geq 1$

$$\begin{aligned} filter_\epsilon^{k\cdot\alpha}(h, X_\epsilon, X_1, \ldots, X_n) &= filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{k\cdot\alpha}(X_1, \ldots, X_n, X_\epsilon) &= filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon), \end{aligned} \tag{3.184}$$

from which the statement follows. □

What follows is a special case of Lemma 3.1.58.

Corollary 3.1.59. *For two filter functions* $filter_\epsilon^\alpha$ *(respectively* $filter_i^\alpha$ *for some* $i \in \mathcal{A}$*) and* $filter_\epsilon^\beta$ *(respectively* $filter_i^\beta$*), some* $k \geq 1$, *where for some downward closed domain* $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$, $filter_\epsilon^\alpha$ *(*$filter_i^\alpha$*) is equal to* $filter_\epsilon^\beta$ *(respectively* $filter_i^\beta$*) and idempotent, the combined filter* $filter_\epsilon^{k\cdot(\alpha+\beta)}$ *(*$filter_i^{k\cdot(\alpha+\beta)}$*) simplifies to*

$$\begin{aligned} filter_\epsilon^{k\cdot(\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) &= filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^\beta(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{k\cdot(\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon) &= filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon) = filter_i^\beta(X_1, \ldots, X_n, X_\epsilon) \end{aligned} \tag{3.185}$$

in this domain.

Proof. The proof is analogous to Lemma 3.1.58, where instead of Lemma 3.1.47, Corollary 3.1.48 is used. □

3.1.7 Fixpoint Filter Intersection

To reiterate, given two event filter functions $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$ for some $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$, we defined *fixpoint filter intersection* as (3.8)

$$filter_\epsilon^{\alpha*\beta}(h, X_\epsilon, X_1, \ldots, X_n) := \lim_{k\to\infty} filter_\epsilon^{k\cdot(\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) \tag{3.186}$$

and for two action filter functions $filter_i^\alpha$ and $filter_i^\beta$ (for $i \in \mathcal{A}$) (3.13) as

$$filter_i^{\alpha*\beta}(X_1, \ldots, X_n, X_\epsilon) := \lim_{k\to\infty} filter_i^{k\cdot(\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon). \tag{3.187}$$

Lemma 3.1.60. *For arbitrary filters $filter_\epsilon^\alpha$, $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$, $filter_i^\beta$ for some $i \in \mathcal{A}$) the fixpoint filter intersections $filter_\epsilon^{\alpha*\beta}$ (respectively $filter_i^{\alpha*\beta}$) is well-defined, i.e., the limits*

$$\begin{aligned} filter_\epsilon^{\alpha*\beta}(h, X_\epsilon, X_1, \ldots, X_n) &= \lim_{k\to\infty} filter_\epsilon^{k\cdot(\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{\alpha*\beta}(X_1, \ldots, X_n, X_\epsilon) &= \lim_{k\to\infty} filter_i^{k\cdot(\alpha+\beta)}(X_\epsilon, X_1, \ldots, X_n, X_\epsilon) \end{aligned} \tag{3.188}$$

always exist.

Proof. By Lemma 3.1.57 repeated nesting of $filter_\epsilon^{\alpha+\beta}$ (respectively $filter_i^{\alpha+\beta}$ for some $i \in \mathcal{A}$) is non-increasing, i.e., can only ever lead to a smaller and smaller subset, from which the statement follows, as

$$\liminf_{k\to\infty} filter_\epsilon^{k\cdot(\alpha+\beta)} = \bigcup_{k\geq 1}\bigcap_{j\geq k} filter_\epsilon^{j\cdot(\alpha+\beta)} = \limsup_{k\to\infty} filter_\epsilon^{k\cdot(\alpha+\beta)} = \bigcap_{k\geq 1}\bigcup_{j\geq k} filter_\epsilon^{j\cdot(\alpha+\beta)}.$$

□

Lemma 3.1.61. *$filter_\epsilon^N$ (respectively $filter_i^N$ for some $i \in \mathcal{A}$) is the neutral element w.r.t. fixpoint intersection for some $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$ for some $i \in \mathcal{A}$) in the domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$ if and only if $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$) is idempotent in this domain.*

Proof. Follows from Lemma 3.1.49. □

Lemma 3.1.62. *Fixpoint filter intersection (3.8), (3.13) preserves the basic filter property: For any $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ and arbitrary filters $filter_\epsilon^\alpha$, $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$, $filter_i^\beta$) it holds that*

$$\begin{aligned} filter_\epsilon^{\alpha*\beta}(h, X_\epsilon, X_1, \ldots, X_n) &\subseteq X_\epsilon \\ filter_i^{\alpha*\beta}(X_1, \ldots, X_n, X_\epsilon) &\subseteq X_i. \end{aligned} \tag{3.189}$$

Thus $filter_\epsilon^{\alpha*\beta}$ and $filter_i^{\alpha*\beta}$ are again filter functions.

Proof. Since fixpoint filter intersection, as k-filter intersection, is defined as nestings of the filter intersection the statement follows from from Lemma 3.1.57, as for any $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$

$$filter_\epsilon^{k\cdot(\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) \subseteq X_\epsilon$$
$$filter_i^{k\cdot(\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon) \subseteq X_i$$

for all $k \in \mathbb{N}$, implies the same for the limit, i.e.,

$$filter_\epsilon^{\alpha*\beta}(h, X_\epsilon, X_1, \ldots, X_n) \subseteq X_\epsilon$$
$$filter_i^{\alpha*\beta}(X_1, \ldots, X_n, X_\epsilon) \subseteq X_i.$$

□

Lemma 3.1.63. *Fixpoint filter intersection (3.8), (3.13) preserves monotonicity for a given downward closed domain: Given any two filter functions $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$ and $filter_i^\beta$ for some $i \in \mathcal{A}$), which are monotonic for a given downward closed domain ($PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$), the combined filter $filter_\epsilon^{\alpha*\beta}$ (respectively $filter_i^{\alpha*\beta}$) is also monotonic for this domain.*

Proof. Follows directly from Lemma 3.1.51, since for any $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X'_\epsilon \subseteq X_\epsilon$, $X_1 \in PD_1$, $X'_1 \subseteq X_1$, ..., $X_n \in PD_n$, $X'_n \subseteq X_n$,

$$filter_\epsilon^{k\cdot(\alpha+\beta)}(h, X'_\epsilon, X'_1, \ldots, X'_n) \subseteq filter_\epsilon^{k\cdot(\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n)$$
$$filter_i^{k\cdot(\alpha+\beta)}(X'_1, \ldots, X'_n, X'_\epsilon) \subseteq filter_i^{k\cdot(\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon)$$

for all $k \in \mathbb{N}$ implies

$$filter_\epsilon^{\alpha*\beta}(h, X'_\epsilon, X'_1, \ldots, X'_n) \subseteq filter_\epsilon^{\alpha*\beta}(h, X_\epsilon, X_1, \ldots, X_n)$$
$$filter_i^{\alpha*\beta}(X'_1, \ldots, X'_n, X'_\epsilon) \subseteq filter_i^{\alpha*\beta}(X_1, \ldots, X_n, X_\epsilon).$$

□

Lemma 3.1.64. *Fixpoint filter intersection (3.8), (3.13) preserves simple monotonicity for a given downward closed domain: Given any two filter functions $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$ and $filter_i^\beta$ for some $i \in \mathcal{A}$), which are simply monotonic for a given downward closed domain ($PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions}_1}$, ..., $PD_n \subseteq 2^{\overline{GActions}_n}$), the combined filter $filter_\epsilon^{\alpha*\beta}$ (respectively $filter_i^{\alpha*\beta}$) is also simply monotonic for this domain.*

Proof. Analogous to Lemma 3.1.63. □

Lemma 3.1.65. *Fixpoint filter intersection always leads to idempotent filters for any downward closed domain: Given any two filter functions $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$ (respectively $filter_i^\alpha$ and $filter_i^\beta$ for some $i \in \mathcal{A}$), the combined filter $filter_\epsilon^{\alpha*\beta}$ (respectively $filter_i^{\alpha*\beta}$) is always idempotent for the whole domain $2^{GEvents}$, $2^{\overline{GActions}_1}$, ..., $2^{\overline{GActions}_n}$.*

Proof. Follows from the definition of fixpoint filter intersection (3.8), (3.13) and Lemma 3.1.60. □

Lemma 3.1.66. *Fixpoint filter intersection is not associative.*

Proof. Analogous to the proof of Lemma 3.1.66. □

Lemma 3.1.67. *Fixpoint filter intersection is commutative.*

Proof. Follows from definition of fixpoint filter intersection (3.8), (3.13) and Lemma 3.1.56. □

Lemma 3.1.68. *For two filter functions $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$ for some $i \in \mathcal{A}$) and $filter_\epsilon^\beta$ (respectively $filter_i^\beta$), where for some downward closed domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$, $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$) is stricter than $filter_\epsilon^\beta$ (respectively $filter_i^\beta$), idempotent and simply monotonic, the combined filter $filter_\epsilon^{\alpha*\beta}$ (respectively $filter_i^{\alpha*\beta}$) simplifies to*

$$\begin{aligned} filter_\epsilon^{\alpha*\beta}(h, X_\epsilon, X_1, \ldots, X_n) &= filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{\alpha*\beta}(X_1, \ldots, X_n, X_\epsilon) &= filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon) \end{aligned} \tag{3.190}$$

for this domain.

Proof. Follows directly from Lemma 3.1.58. □

Corollary 3.1.69. *For two filter functions $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$ for some $i \in \mathcal{A}$) and $filter_\epsilon^\beta$ (respectively $filter_i^\beta$), where for some downward closed domain $PD_\epsilon \subseteq 2^{GEvents}$, $PD_1 \subseteq 2^{\overline{GActions_1}}$, ..., $PD_n \subseteq 2^{\overline{GActions_n}}$, $filter_\epsilon^\alpha$ (respectively $filter_i^\alpha$) is equal to $filter_\epsilon^\beta$ (respectively $filter_i^\beta$) and idempotent, the combined filter $filter_\epsilon^{\alpha*\beta}$ (respectively $filter_i^{\alpha*\beta}$) simplifies to*

$$\begin{aligned} filter_\epsilon^{\alpha*\beta}(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^\beta(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{\alpha*\beta}(X_1, \ldots, X_n, X_\epsilon) = filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon) = filter_i^\beta(X_1, \ldots, X_n, X_\epsilon) \end{aligned} \tag{3.191}$$

in this domain.

Proof. Follows from Lemma 3.1.59. □

3.1.8 General Filter Combination Properties

Lemma 3.1.70. *Given two event filter functions $filter_\epsilon^\alpha$ and $filter_\epsilon^\beta$, for the combined filters it holds that (for some $k \geq 1$)*

$$filter_\epsilon^{\alpha*\beta} \subseteq filter_\epsilon^{k\cdot(\alpha+\beta)} \subseteq filter_\epsilon^{\alpha+\beta} \subseteq filter_\epsilon^{\beta\circ\alpha} \subseteq filter_\epsilon^\alpha \tag{3.192}$$

$$filter_\epsilon^{\alpha*\beta} \subseteq filter_\epsilon^{k\cdot(\alpha+\beta)} \subseteq filter_\epsilon^{\alpha+\beta} \subseteq filter_\epsilon^{\alpha\circ\beta} \subseteq filter_\epsilon^\beta. \tag{3.193}$$

Similarly for $i \in \mathcal{A}$ given two action filter functions $filter_i^\alpha$ and $filter_i^\beta$, for the combined filters it holds that (again for some $k \geq 1$)

$$filter_i^{\alpha*\beta} \subseteq filter_i^{k\cdot(\alpha+\beta)} \subseteq filter_i^{\alpha+\beta} \subseteq filter_i^{\beta\circ\alpha} \subseteq filter_i^\alpha \tag{3.194}$$

$$filter_i^{\alpha*\beta} \subseteq filter_i^{k\cdot(\alpha+\beta)} \subseteq filter_i^{\alpha+\beta} \subseteq filter_i^{\alpha\circ\beta} \subseteq filter_i^\beta. \tag{3.195}$$

Proof. From Definitions 3.1.2, 3.1.3 of event and action filter function combination for some global history $h \in \mathcal{G}$, agent $i \in \mathcal{A}$, sets $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ and $k \geq 1$ since filter functions satisfy the basic filter property (in accordance with Definition 3.1.7) (meaning they only remove elements)

$$filter_\epsilon^{\beta \circ \alpha}(h, X_\epsilon, X_1, \ldots, X_n) \subseteq filter_\epsilon^{\alpha}(h, X_\epsilon, X_1, \ldots, X_n)$$

$$filter_\epsilon^{\alpha \circ \beta}(h, X_\epsilon, X_1, \ldots, X_n) \subseteq filter_\epsilon^{\beta}(h, X_\epsilon, X_1, \ldots, X_n)$$

and

$$filter_i^{\beta \circ \alpha}(X_1, \ldots, X_n, X_\epsilon) \subseteq filter_i^{\alpha}(X_1, \ldots, X_n, X_\epsilon)$$

$$filter_i^{\alpha \circ \beta}(X_1, \ldots, X_n, X_\epsilon) \subseteq filter_i^{\beta}(X_1, \ldots, X_n, X_\epsilon).$$

clearly hold as well. By semantics of set intersection we get that

$$\begin{aligned} filter_\epsilon^{\alpha+\beta}(h, X_\epsilon, X_1, \ldots, X_n) &\subseteq filter_\epsilon^{\alpha \circ \beta}(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_\epsilon^{\alpha+\beta}(h, X_\epsilon, X_1, \ldots, X_n) &\subseteq filter_\epsilon^{\beta \circ \alpha}(h, X_\epsilon, X_1, \ldots, X_n) \end{aligned}$$

and again similarly for action filters

$$\begin{aligned} filter_i^{\alpha+\beta}(X_1, \ldots, X_n, X_\epsilon) &\subseteq filter_i^{\alpha \circ \beta}(X_1, \ldots, X_n, X_\epsilon) \\ filter_i^{\alpha+\beta}(X_1, \ldots, X_n, X_\epsilon) &\subseteq filter_i^{\beta \circ \alpha}(X_1, \ldots, X_n, X_\epsilon). \end{aligned}$$

Since $k \geq 1$ by Lemma 3.1.57 it follows that

$$\begin{aligned} filter_\epsilon^{k \cdot (\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) &\subseteq filter_\epsilon^{\alpha+\beta}(h, X_\epsilon, X_1, \ldots, X_n) \\ filter_i^{k \cdot (\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon) &\subseteq filter_i^{\alpha+\beta}(X_1, \ldots, X_n, X_\epsilon). \end{aligned} \tag{3.196}$$

Lastly by (3.8), (3.13) (definition of $filter_\epsilon^{\alpha * \beta}$ and $filter_i^{\alpha * \beta}$) and Lemma 3.1.57 we get that for any $k' \in \mathbb{N}$

$$\begin{aligned} \lim_{\widetilde{k} \to \infty} filter_\epsilon^{\widetilde{k} \cdot (\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) &\subseteq filter_\epsilon^{k' \cdot (\alpha+\beta)}(h, X_\epsilon, X_1, \ldots, X_n) \\ \lim_{\widetilde{k} \to \infty} filter_i^{\widetilde{k} \cdot (\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon) &\subseteq filter_i^{k' \cdot (\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon). \end{aligned} \tag{3.197}$$

□

In order to demonstrate that our introduced notion of filter intersection (for event filters by Definition 3.1.2) is consistent with the combined filter from Chapter 2, we will now show that the Fully Byzantine asynchronous agents event filter (2.93) can be constructed with (3.5).

Lemma 3.1.71. *For some global history $h \in \mathcal{G}$, sets $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$*

$$filter_\epsilon^{B_f}(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^{B \circ \leq f}(h, X_\epsilon, X_1, \ldots, X_n) = filter_\epsilon^{B + \leq f}(h, X_\epsilon, X_1, \ldots, X_n) \tag{3.198}$$

Proof. By (2.92), (2.93),

$$\begin{aligned} &filter_\epsilon^{B_f}(h, X_\epsilon, X_1, \ldots, X_n) = \\ &filter_\epsilon^{B}\left(h, filter_\epsilon^{\leq f}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right) \end{aligned} \tag{3.199}$$

and by Definition 3.1.2

$$filter_\epsilon^{B+\leq f}(h, X_\epsilon, X_1, \ldots, X_n) = \tag{3.200}$$

$$filter_\epsilon^{\leq f \circ B}(h, X_\epsilon, X_1, \ldots, X_n) \cap filter_\epsilon^{B \circ \leq f}(h, X_\epsilon, X_1, \ldots, X_n) = \tag{3.201}$$

$$filter_\epsilon^{\leq f}\left(h, filter_\epsilon^{B}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right) \cap \tag{3.202}$$

$$filter_\epsilon^{B}\left(h, filter_\epsilon^{\leq f}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right) \tag{3.203}$$

we see that (3.199) is equal to line (3.203). Therefore it remains to show that (3.203) is a subset of (3.202).

We start by examining (3.202). By (2.24), $filter_\epsilon^B$ only removes correct receive events and by (2.91), $filter_\epsilon^{\leq f}$ removes only Byzantine events (including $sleep(i)$ and $hibernate(i)$ for some agent i) depending on the current Byzantine events (and obviously on the number of Byzantine agents so far via the history, which however for this reasoning is unimportant), but is independent of any *grecv* events. Thus

$$\begin{aligned} &X_\epsilon \setminus filter_\epsilon^{\leq f}(h, X_\epsilon, X_1, \ldots, X_n) = \\ &filter_\epsilon^{B}(h, X_\epsilon, X_1, \ldots, X_n) \setminus filter_\epsilon^{\leq f}\left(h, filter_\epsilon^{B}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right) \end{aligned} \tag{3.204}$$

and let

$$D_\epsilon := X_\epsilon \setminus filter_\epsilon^{\leq f}(h, X_\epsilon, X_1, \ldots, X_n). \tag{3.205}$$

Regarding (3.203), since by (2.24) $filter_\epsilon^B$ only removes correct receive events, depending on possibly removed fake send events and since $filter_\epsilon^{\leq f}$ (2.91) does not remove any correct receive events however might remove fake send events, and since we assumed X_ϵ to be t-coherent (meaning that especially no *go* event can occur simultaneously with *sleep* or *hibernate*), it follows that

$$\begin{aligned} &X_\epsilon \setminus filter_\epsilon^{B}(h, X_\epsilon, X_1, \ldots, X_n) \subseteq \\ &filter_\epsilon^{\leq f}(h, X_\epsilon, X_1, \ldots, X_n) \setminus filter_\epsilon^{B}\left(h, filter_\epsilon^{\leq f}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right). \end{aligned} \tag{3.206}$$

Let

$$E_\epsilon := filter_\epsilon^{\leq f}(h, X_\epsilon, X_1, \ldots, X_n) \setminus filter_\epsilon^{B}\left(h, filter_\epsilon^{\leq f}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right) \tag{3.207}$$

and

$$E'_\epsilon := X_\epsilon \setminus filter_\epsilon^{B}(h, X_\epsilon, X_1, \ldots, X_n). \tag{3.208}$$

We can rewrite (3.202) as

$$filter_\epsilon^{\leq f}\left(h, filter_\epsilon^{B}(h, X_\epsilon, X_1, \ldots, X_n), X_1, \ldots, X_n\right) = (X_\epsilon \setminus E'_\epsilon) \setminus D_\epsilon \tag{3.209}$$

and (3.203) as

$$filter_{\epsilon}^{B}\left(h, filter_{\epsilon}^{\leq f}\left(h, X_{\epsilon}, X_1, \ldots, X_n\right), X_1, \ldots, X_n\right) = \left(X_{\epsilon} \setminus D_{\epsilon}\right) \setminus E_{\epsilon}. \tag{3.210}$$

Thus by (3.206), (3.210), (3.209), (3.207), (3.208), semantics of set intersection and set difference it indeed holds that

$$filter_{\epsilon}^{B \circ \leq f}\left(h, X_{\epsilon}, X_1, \ldots, X_n\right) \subseteq filter_{\epsilon}^{\leq f \circ B}\left(h, X_{\epsilon}, X_1, \ldots, X_n\right), \tag{3.211}$$

from which the Lemma follows. □

3.2 Extension Combination

Definition 3.2.1. For two extensions $\mathscr{E}^{\alpha} = (PP^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha})$ and $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta})$ with sets of pairs of protocols $PP^{\alpha}, PP^{\beta} \subseteq \mathscr{C}_{\epsilon} \times \mathscr{C}$, two transition templates $\tau^{\alpha}, \tau^{\beta}$, two admissibility conditions $\Psi^{\alpha}, \Psi^{\beta}$ and two sets of sets of global initial states IS^{α}, IS^{β}, we define their

- composition as

$$\mathscr{E}^{\alpha \circ \beta} := (PP^{\alpha} \cap PP^{\beta}, IS^{\alpha} \cap IS^{\beta}, \tau^{\alpha \circ \beta}, \Psi^{\alpha} \cap \Psi^{\beta}), \tag{3.212}$$

- intersection as

$$\mathscr{E}^{\alpha + \beta} := (PP^{\alpha} \cap PP^{\beta}, IS^{\alpha} \cap IS^{\beta}, \tau^{\alpha + \beta}, \Psi^{\alpha} \cap \Psi^{\beta}), \tag{3.213}$$

- k-intersection (for $k \in \mathbb{N}$) as

$$\mathscr{E}^{k \cdot (\alpha + \beta)} := (PP^{\alpha} \cap PP^{\beta}, IS^{\alpha} \cap IS^{\beta}, \tau^{k \cdot (\alpha + \beta)}, \Psi^{\alpha} \cap \Psi^{\beta}), \tag{3.214}$$

- fixpoint intersection as

$$\mathscr{E}^{\alpha * \beta} := (PP^{\alpha} \cap PP^{\beta}, IS^{\alpha} \cap IS^{\beta}, \tau^{\alpha * \beta}, \Psi^{\alpha} \cap \Psi^{\beta}). \tag{3.215}$$

Note that the combinations $\mathscr{E}^{\alpha \circ \beta}$, $\mathscr{E}^{\alpha + \beta}$,$\mathscr{E}^{k \cdot (\alpha + \beta)}$,$\mathscr{E}^{\alpha * \beta}$ need not necessarily be valid extensions again. Therefore we introduce the notion of (in-)compatibility of extensions. Informally two extensions are said to be *compatible* if their combination can produce some runs. It is formally defined as follows:

Definition 3.2.2 (Compatibility)**.** For a number of $l \geq 2$ extensions $\mathscr{E}^{\alpha_1}$, $\mathscr{E}^{\alpha_2}$, ..., $\mathscr{E}^{\alpha_l}$ we say the extensions $\mathscr{E}^{\alpha_1}$, $\mathscr{E}^{\alpha_2}$, ..., $\mathscr{E}^{\alpha_l}$ are compatible w.r.t. to some series of extension combinations $\star_1$, $\star_2$, ..., $\star_{l-1}$ [1] iff

- $PP_1^{\alpha} \cap \ldots \cap PP_l^{\alpha} \neq \varnothing$ and

[1] For some $l' \in \mathbb{N}$ we use $\alpha \star_{l'} \beta$ or just $\star$ to represent either composition $(\alpha \circ \beta)$, reversed composition $(\beta \circ \alpha)$, k-intersection for some $k \geq 1$ $(k \cdot (\alpha + \beta))$ or fixpoint intersection $(\alpha * \beta)$.

- $IS_1^\alpha \cap \ldots \cap IS_l^\alpha \neq \varnothing$ and
- $\Psi^{\alpha_1} \cap \ldots \cap \Psi^{\alpha_l} \neq \varnothing$ and
- $\exists \chi \in \mathscr{E}^{\alpha_1 \star_1 \alpha_2 \star_2 \ldots \star_{l-1} \alpha_l}$.

Reciprocally, we define the incompatibility relation between extensions $\mathscr{E}^{\alpha_1}$, $\mathscr{E}^{\alpha_2}$, ..., $\mathscr{E}^{\alpha_l}$ w.r.t. some series of extension combinations $\star_1$, $\star_2$, ..., $\star_{l-1}$: $\mathscr{E}^{\alpha_1}$, $\mathscr{E}^{\alpha_2}$, ..., $\mathscr{E}^{\alpha_l}$ are *incompatible* w.r.t. $\star_1$, $\star_2$, ..., $\star_{l-1}$ iff they are not compatible w.r.t. $\star_1$, $\star_2$, ..., $\star_{l-1}$.

Definition 3.2.3. Iff extensions $\mathscr{E}^{\alpha_1}$, $\mathscr{E}^{\alpha_2}$, ..., $\mathscr{E}^{\alpha_l}$ $(l \geq 2)$ are *compatible* w.r.t. the extension combination series $\star_1$, $\star_2$, ..., $\star_{l-1}$, then $\mathscr{E}^{\alpha_1 \star_1 \alpha_2 \star_2 \ldots \star_{l-1} \alpha_l}$ is also an extension.

Lemma 3.2.4. *Extension composition is associative.*

$$\mathscr{E}^{(\alpha \circ \beta) \circ \gamma} = \mathscr{E}^{\alpha \circ (\beta \circ \gamma)} \tag{3.216}$$

Proof. From Definition 3.2.1 for three extensions $\mathscr{E}^\alpha$, $\mathscr{E}^\beta$, $\mathscr{E}^\gamma$ we get

$$\begin{aligned} \mathscr{E}^{(\alpha \circ \beta) \circ \gamma} &:= ((PP^\alpha \cap PP^\beta) \cap PP^\gamma, (IS^\alpha \cap IS^\beta) \cap IS^\gamma, \tau^{(\alpha \circ \beta) \circ \gamma}, (\Psi^\alpha \cap \Psi^\beta) \cap \Psi^\gamma) = \\ \mathscr{E}^{\alpha \circ (\beta \circ \gamma)} &:= (PP^\alpha \cap (PP^\beta \cap PP^\gamma), IS^\alpha \cap (IS^\beta \cap IS^\gamma), \tau^{\alpha \circ (\beta \circ \gamma)}, \Psi^\alpha \cap (\Psi^\beta \cap \Psi^\gamma)) \end{aligned} \tag{3.217}$$

by associativity of set intersection, where $\tau^{(\alpha \circ \beta) \circ \gamma} = \tau^{\alpha \circ (\beta \circ \gamma)}$ follows from Lemma 3.1.31. □

Lemma 3.2.5. *Extension composition is generally not commutative.*

$$\mathscr{E}^{\alpha \circ \beta} \neq \mathscr{E}^{\beta \circ \alpha} \tag{3.218}$$

Proof. This follows from Lemma 3.1.32. □

Lemma 3.2.6. *Extension intersection is generally not associative.*

$$\mathscr{E}^{(\alpha + \beta) + \gamma} \neq \mathscr{E}^{\alpha + (\beta + \gamma)} \tag{3.219}$$

Proof. This follows from Lemma 3.1.45. □

Lemma 3.2.7. *Extension intersection is commutative.*

$$\mathscr{E}^{\alpha + \beta} = \mathscr{E}^{\beta + \alpha} \tag{3.220}$$

Proof. From Definition 3.2.1 for two extensions $\mathscr{E}^\alpha$, $\mathscr{E}^\beta$ we get

$$\begin{aligned} \mathscr{E}^{\alpha + \beta} &:= (PP^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{\alpha + \beta}, \Psi^\alpha \cap \Psi^\beta) = \\ \mathscr{E}^{\beta + \alpha} &:= (PP^\beta \cap PP^\alpha, IS^\beta \cap IS^\alpha, \tau^{\beta + \alpha}, \Psi^\beta \cap \Psi^\alpha) \end{aligned} \tag{3.221}$$

by commutativity of set intersection, where $\tau^{\alpha + \beta} = \tau^{\beta + \alpha}$ follows from Lemma 3.1.46. □

3.3 Extension Classification

In our framework, extension properties can be implemented in many different ways using a combination of altering the

- environment protocol,
- joint protocol,
- event filter,
- action filters, and
- admissibility condition.

One crucial question arises however: If a particular property could be implemented using different restriction methods [2] - for example either by altering the joint protocol or by changing the action filters: Which of these methods should be favored? A major goal of this framework is to provide a model for extensions that is as modular and as composable as possible. Intuitively, it makes sense that not all possible implementations of properties are easily composable with any other implementation. Therefore, in this section, we provide a classification of extension implementations, which we refer to as *implementation classes*, and analyze their composability to answer our posed question.

3.3.1 Implementation Classes

In this section, we also introduce some implementation classes that utilize neutral action filters. We are well aware that in principle this violates our assumption (2.52). However, note that we are only introducing these kinds of implementation classes to be able to define a downward closed subset of them and reap the benefits of downward closed safety properties (see Section 3.3.3) w.r.t. improved composability, since by Lemma 3.3.53 any extension's safety property that uses the Byzantine action filters is inherently not downward closed.

Definition 3.3.1. *We assume that in the following implementation classes, all filter functions are idempotent for the domain* PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$. Note that the neutral and Byzantine filters are idempotent for PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ by Lemmas 3.1.21 and 3.1.22.

- **Adm**
  The desired extension property is implemented via an appropriate admissibility condition Ψ. An extension $\mathscr{E}^\alpha \in$ **Adm** iff

$$\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}, IS^\alpha, \tau^{N,N}, \Psi^\alpha), \tag{3.222}$$

  where in $\tau^{N,N}$ the filter functions $filter_\epsilon^N$ and $filter_i^N$ (for all $i \in \mathcal{A}$) are used, $IS^\alpha \subseteq 2^{\mathscr{G}(0)}$ and $\Psi^\alpha \subset R$ hold.

[2] Note that any property could always be implemented via the admissibility condition. For non-liveness properties, this would inevitably lead to a not non-excluding extension, however.

- **JP**
  The extension property is implemented via restricting the set of joint protocols $\mathscr{C}$. An extension $\mathscr{E}^\alpha \in$ **JP** iff
  $$\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha, IS^\alpha, \tau^{N,N}, \Psi^\alpha), \tag{3.223}$$
  where in $\tau^{N,N}$ the filter functions $filter_\epsilon^N$ and $filter_i^N$ (for all $i \in \mathcal{A}$) are used, $\mathscr{C}^\alpha \subset \mathscr{C}$, $\Psi^\alpha \subseteq R$ and $IS^\alpha \subseteq 2^{\mathscr{G}(0)}$ hold.

- **JP – AFB**
  An extension $\mathscr{E}^\alpha \in$ **JP – AFB** iff
  $$\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha, IS^\alpha, \tau^{N,B}, \Psi^\alpha), \tag{3.224}$$
  where in $\tau^{N,B}$ the filter functions $filter_\epsilon^N$ and $filter_i^B$ (for all $i \in \mathcal{A}$) are used, $\mathscr{C}^\alpha \subset \mathscr{C}$, $\Psi^\alpha \subseteq R$ and $IS^\alpha \subseteq 2^{\mathscr{G}(0)}$ hold.

- **EnvJP**
  The extension property is implemented via restricting the set of environment protocols $\mathscr{C}_\epsilon$ possibly in conjunction with the set of joint protocols $\mathscr{C}$. An extension $\mathscr{E}^\alpha \in$ **EnvJP** iff
  $$\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^{N,N}, \Psi^\alpha), \tag{3.225}$$
  where in $\tau^{N,N}$ the filter functions $filter_\epsilon^N$ and $filter_i^N$ (for all $i \in \mathcal{A}$) are used, $PP^\alpha \subset \mathscr{C}_\epsilon \times \mathscr{C}$, $\Psi^\alpha \subseteq R$, $IS^\alpha \subseteq 2^{\mathscr{G}(0)}$ and $\mathscr{E}^\alpha \notin$ **JP** hold.

- **EnvJP – AFB**
  An extension $\mathscr{E}^\alpha \in$ **EnvJP – AFB** iff
  $$\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^{N,B}, \Psi^\alpha), \tag{3.226}$$
  where in $\tau^{N,B}$ the filter functions $filter_\epsilon^N$ and $filter_i^B$ (for all $i \in \mathcal{A}$) are used, $PP^\alpha \subset \mathscr{C}_\epsilon \times \mathscr{C}$, $\Psi^\alpha \subseteq R$, $IS^\alpha \subseteq 2^{\mathscr{G}(0)}$ and $\mathscr{E}^\alpha \notin$ **JP – AFB** hold.

- **EvFJP**
  The extension property is implemented via restricting the environment filter $filter_\epsilon$ and possibly also the set of joint protocols $\mathscr{C}$. An extension $\mathscr{E}^\alpha \in$ **EvFJP** iff
  $$\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha, IS^\alpha, \tau^{\alpha,N}, \Psi^\alpha), \tag{3.227}$$
  where in $\tau^{\alpha,N}$ the filter functions $filter_\epsilon^\alpha$ and $filter_i^N$ (for all $i \in \mathcal{A}$) are used, $\mathscr{C}^\alpha \subseteq \mathscr{C}$, $\Psi^\alpha \subseteq R$ and $IS^\alpha \subseteq 2^{\mathscr{G}(0)}$ hold, where for $filter_\epsilon^\alpha$ there exist some $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ such that
  $$filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) \subset filter_\epsilon^N(h, X_\epsilon, X_1, \ldots, X_n). \tag{3.228}$$

- **EvFJP – AFB**
  An extension $\mathscr{E}^\alpha \in$ **EvFJP – AFB** iff
  $$\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha, IS^\alpha, \tau^{\alpha,B}, \Psi^\alpha), \tag{3.229}$$
  where in $\tau^{\alpha,B}$ the filter functions $filter_\epsilon^\alpha \subset filter_\epsilon^N$ and $filter_i^B$ (for all $i \in \mathcal{A}$) are used, $\mathscr{C}^\alpha \subseteq \mathscr{C}$, $\Psi^\alpha \subseteq R$ and $IS^\alpha \subseteq 2^{\mathscr{G}(0)}$ hold, where for $filter_\epsilon^\alpha$ there exist some $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ such that
  $$filter_\epsilon^\alpha(h, X_\epsilon, X_1, \ldots, X_n) \subset filter_\epsilon^N(h, X_\epsilon, X_1, \ldots, X_n). \tag{3.230}$$

- **EvFEnvJP**
  An extension $\mathscr{E}^\alpha \in$ **EvFEnvJP** iff

$$\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^{\alpha,N}, \Psi^\alpha), \tag{3.231}$$

  where in $\tau^{\alpha,N}$ the filter functions $filter_\epsilon^\alpha \subset filter_\epsilon^N$ and the neutral action filters $filter_i^N$ (for all $i \in \mathcal{A}$) are used, $PP^\alpha \subset \mathscr{C}_\epsilon \times \mathscr{C}$, $\Psi^\alpha \subseteq R$ and $IS^\alpha \subseteq 2^{\mathscr{G}(0)}$ hold, where for $filter_\epsilon^\alpha$ there exist some $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ such that

$$filter_\epsilon^\alpha\,(h, X_\epsilon, X_1, \ldots, X_n) \subset filter_\epsilon^N\,(h, X_\epsilon, X_1, \ldots, X_n)\,. \tag{3.232}$$

- **EvFEnvJP – AFB**
  An extension $\mathscr{E}^\alpha \in$ **EvFEnvJP – AFB** iff

$$\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^{\alpha,B}, \Psi^\alpha), \tag{3.233}$$

  where in $\tau^{\alpha,B}$ the filter functions $filter_\epsilon^\alpha \subset filter_\epsilon^N$ and the Byzantine action filters $filter_i^B$ (for all $i \in \mathcal{A}$) are used, $PP^\alpha \subset \mathscr{C}_\epsilon \times \mathscr{C}$, $\Psi^\alpha \subseteq R$ and $IS^\alpha \subseteq 2^{\mathscr{G}(0)}$ hold, where for $filter_\epsilon^\alpha$ there exist some $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ such that

$$filter_\epsilon^\alpha\,(h, X_\epsilon, X_1, \ldots, X_n) \subset filter_\epsilon^N\,(h, X_\epsilon, X_1, \ldots, X_n)\,. \tag{3.234}$$

- **Others**
  This class contains all remaining extension implementations like restrictions via arbitrary action filters $filter_i$ (for $i \in \mathcal{A}$) and possibly something else.

  An extension $\mathscr{E}^\alpha \in$ **Others** iff

$$\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^\alpha, \Psi^\alpha), \tag{3.235}$$

  where in τ^α the filter functions (for all $i \in \mathcal{A}$) $filter_\epsilon^\alpha$ and $filter_i^\alpha$ are used, where $filter_i^\alpha \neq filter_i^B$ and $filter_i^\alpha \subset filter_i^N$, $\Psi^\alpha \subseteq R$, $IS^\alpha \subseteq 2^{\mathscr{G}(0)}$, $PP^\alpha \subseteq \mathscr{C}_\epsilon \times \mathscr{C}$ and where for $filter_i^\alpha$ (for every $i \in \mathcal{A}$) there exist some $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ such that

$$filter_i^\alpha\,(X_1, \ldots, X_n, X_\epsilon) \subset filter_i^N\,(X_1, \ldots, X_n, X_\epsilon)\,. \tag{3.236}$$

Next we list important subsets of the just mentioned implementation classes, which however we will also treat as individual implementation classes (see listing below):

- **JP$_{\mathbf{DC}}$**
  This implementation class is a subset of **JP**, additionally restricted to extensions, whose safety properties (see Section 3.3.3) are downward closed. Formally, this means that

$$\textbf{JP}_{\textbf{DC}} := \{\mathscr{E}^\alpha \in \textbf{JP} \mid S^\alpha \text{ is downward closed}\} \tag{3.237}$$

- **EnvJP$_{\mathbf{DC}}$**
  This implementation class is a subset of **EnvJP**, additionally restricted to extensions, whose safety properties are downward closed. Formally, this means that

$$\textbf{EnvJP}_{\textbf{DC}} := \{\mathscr{E}^\alpha \in \textbf{EnvJP} \mid S^\alpha \text{ is downward closed}\} \tag{3.238}$$

- **EvFJP$_{\mathbf{DC}}$**
  This implementation class is a subset of **EvFJP**, additionally restricted to extensions, whose safety properties are downward closed. Formally, this means that

$$\mathbf{EvFJP_{DC}} := \{\mathscr{E}^\alpha \in \mathbf{EvFJP} \mid S^\alpha \text{ is downward closed}\} \tag{3.239}$$

- **EvFEnvJP$_{\mathbf{DC}}$**
  This implementation class is a subset of **EvFEnvJP**, additionally restricted to extensions, whose safety properties are downward closed. Formally, this means that

$$\mathbf{EvFEnvJP_{DC}} := \{\mathscr{E}^\alpha \in \mathbf{EvFEnvJP} \mid S^\alpha \text{ is downward closed}\} \tag{3.240}$$

- **Others$_{\mathbf{DC}}$**
  This implementation class is a subset of **Others**, additionally restricted to extensions, whose safety properties are downward closed. Formally, this means that

$$\mathbf{Others_{DC}} := \{\mathscr{E}^\alpha \in \mathbf{Others} \mid S^\alpha \text{ is downward closed}\} \tag{3.241}$$

- **EvFEnvJP$_{\mathbf{DC\,mono}}$**
  This implementation class is a subset of **EvFEnvJP$_{\mathbf{DC}}$**, additionally restricted to extensions, whose event filter functions are simply monotonic and whose action filters are monotonic in the domain PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$. Formally, this means that

$$\mathbf{EvFEnvJP_{DC\,mono}} := \{\mathscr{E}^\alpha \in \mathbf{EvFEnvJP_{DC}} \mid (\forall i \in \mathcal{A})\ filter_i^\alpha \text{ and } filter_\epsilon^\alpha \text{ are monotonic for the domain } PD_\epsilon^{t-coh}, 2^{\overline{GActions_1}}, \ldots, 2^{\overline{GActions_n}}\} \tag{3.242}$$

- **Others$_{\mathbf{DC\,mono}}$**
  This implementation class is a subset of **Others$_{\mathbf{DC}}$**, additionally restricted to extensions, whose event filter functions are simply monotonic and whose action filters are monotonic in the domain PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$. Formally, this means that

$$\mathbf{Others_{DC\,mono}} := \{\mathscr{E}^\alpha \in \mathbf{Others_{DC}} \mid (\forall i \in \mathcal{A})\ filter_i^\alpha \text{ and } filter_\epsilon^\alpha \text{ are monotonic for the domain } PD_\epsilon^{t-coh}, 2^{\overline{GActions_1}}, \ldots, 2^{\overline{GActions_n}}\} \tag{3.243}$$

The set of all implementation classes is denoted by $\mathscr{I}$ and to provide a concise overview consists of the following classes:

$$\begin{aligned}\mathscr{I} := \{&\mathbf{Adm},\\ &\mathbf{JP}, \mathbf{JP_{DC}}, \mathbf{JP-AFB},\\ &\mathbf{EnvJP}, \mathbf{EnvJP_{DC}}, \mathbf{EnvJP-AFB},\\ &\mathbf{EvFJP}, \mathbf{EvFJP_{DC}}, \mathbf{EvFJP-AFB},\\ &\mathbf{EvFEnvJP}, \mathbf{EvFEnvJP_{DC}}, \mathbf{EvFEnvJP_{DC\,mono}}, \mathbf{EvFEnvJP-AFB},\\ &\mathbf{Others}, \mathbf{Others_{DC}}, \mathbf{Others_{DC\,mono}}\}.\end{aligned}$$

Before we delve into the analysis of composability of these implementation classes, we first have to formalize the notions of liveness and safety properties in our framework.

3.3.2 Trace-based Safety and Liveness Properties

Definition 3.3.2. We define $R^{trans} \subseteq R$ as the set of all transitional runs.

$$R^{trans} := \{r \in R \mid (\forall t \in \mathbb{N})(\exists X \in (2^{GEvents} \times 2^{\overline{GActions}_1} \times \cdots \times 2^{\overline{GActions}_n})) \\ r(t+1) = update\,(r(t), X)\} \tag{3.244}$$

Definition 3.3.3. We define a liveness property L as

$$L \subseteq R^{trans}, \text{ where} \\ L \neq \varnothing \ \wedge \ (\forall r \in R^{trans})(\forall t \in \mathbb{N})(\exists r' \in L)\ r'(t) = r(t). \tag{3.245}$$

Informally this means every prefix $r(t)$ of a run $r \in R^{trans}$ for some timestamp $t \in \mathbb{N}$, has an extension in L.

Definition 3.3.4. Let

$$PR^{trans} = \{h \mid (\exists r \in R^{trans})(\exists t \in \mathbb{N})\ r(t) = h\}. \tag{3.246}$$

A set $S' \subseteq R^{trans} \sqcup PR^{trans}$ is a safety property if

(i) $S' \neq \varnothing$,

(ii) S' is prefix-closed in that

- $r(t) \in S'$ for $r \in R^{trans}$ and $t \in \mathbb{N}$ also implies that $r(t') \in S'$ for $t' \leq t$ and
- $r' \in S'$ implies that $r'(t'') \in S'$ for all $t'' \in \mathbb{N}$,

(iii) S' is limit-closed in that $r(t) \in S'$ for $r \in R^{trans}$ and all $t \in \mathbb{N}$ implies that $r \in S'$.

Definition 3.3.5 (Adherence to Safety respectively Liveness Property)**.** We say an extension $\mathscr{E}^\alpha$ adheres to a safety property S', respectively liveness property L iff

$$\bigcup_{\chi^\alpha \in \mathscr{E}^\alpha} R^{\chi^\alpha} \ \subseteq \ S', \tag{3.247}$$

respectively

$$\bigcup_{\chi^\alpha \in \mathscr{E}^\alpha} R^{\chi^\alpha} \ \subseteq \ L. \tag{3.248}$$

In the following we will show that any property $P^\alpha \subseteq R^{trans}$ can be written as intersection of a safety and liveness property.

Definition 3.3.6. For a set $P^\alpha \subseteq R^{trans}$ of transitional runs, where $P^\alpha \neq \varnothing$, we define

$$L'^\alpha := \{r \in R^{trans} \mid (\exists t \in \mathbb{N})(\forall r' \in P^\alpha)(\forall t' \in \mathbb{N})\ r(t) \neq r'(t')\} \tag{3.249}$$

$$\overline{L^\alpha} := P^\alpha \cup L'^\alpha. \tag{3.250}$$

Lemma 3.3.7. *$\overline{L^\alpha}$ is a liveness property.*

Proof. Since $P^\alpha \neq \varnothing$, $\overline{L^\alpha} \neq \varnothing$ as well by (3.250).

Take any finite prefix $r(t)$ of a run $r \in R^{trans}$ for some timestamp $t \in \mathbb{N}$. If $r(t)$ has an extension in P^α, then there exists a run $r' \in P^\alpha$, s.t. $r(t) = r'(t)$. Since by Definition 3.3.6 $P^\alpha \subseteq \overline{L^\alpha}$, $r' \in \overline{L^\alpha}$ as well. If $r(t)$ has no extension in P^α, then by Definition 3.3.6 $r \in L'^\alpha$, thus $r \in \overline{L^\alpha}$. □

Lemma 3.3.8. *$P^\alpha = \overline{L^\alpha} \cap S'^\alpha$, where $S'^\alpha \subseteq R^{trans} \sqcup PR^{trans}$ is the prefix and limit closure of P^α.*

Proof. Since $P^\alpha \subseteq S'^\alpha$ and $P^\alpha \subseteq \overline{L^\alpha}$, it follows that $P^\alpha \subseteq \overline{L^\alpha} \cap S'^\alpha$. Hence it remains to show that $\overline{L^\alpha} \cap S'^\alpha \subseteq P^\alpha$. Assume by contradiction that there exists a run $r \in \overline{L^\alpha} \cap S'^\alpha$, but $r \notin P^\alpha$, hence $r \in \overline{L^\alpha}$ — specifically $r \in L'^\alpha$ — and $r \in S'^\alpha$. Since $r \in S'^\alpha$ (by prefix closure of S'^α) for all $t' \in \mathbb{N}$, $r(t') \in S'^\alpha$ as well. This implies (by limit closure of S'^α) that there must exist a run $r' \in P^\alpha$ such that $r(t) = r'(t)$ for all $t \in \mathbb{N}$. This however contradicts that $r \in L'^\alpha$. □

Definition 3.3.9. We define L^α as the liveness property $\overline{L^\alpha}$ of some extension $\mathscr{E}^\alpha$ for

$$P^\alpha = \bigcup_{\chi \in \mathscr{E}^\alpha} R^\chi. \tag{3.251}$$

Lemma 3.3.10. *Consider two arbitrary extensions $\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^\alpha, \Psi^\alpha)$, $\mathscr{E}^\beta = (PP^\beta, IS^\beta, \tau^\beta, \Psi^\beta)$ both encoding some properties (not necessarily limited to liveness properties) in their admissibility conditions $\Psi^\alpha \subseteq R$ and $\Psi^\beta \subseteq R$. If $\mathscr{E}^\alpha$ and $\mathscr{E}^\beta$ are compatible w.r.t. the combination $\star$, then*

$$(\forall \chi \in \mathscr{E}^{\alpha\star\beta})(\forall r \in R^\chi)\ r \in \Psi^\alpha \ \wedge\ r \in \Psi^\beta, \tag{3.252}$$

meaning that any combined extension satisfies the properties encoded by the admissibility conditions of both extensions.

Proof. If the two extensions are compatible w.r.t. $\star$, by Definition 3.2.1 of extension combination the resulting extension is

$$\mathscr{E}^{\alpha\star\beta} = (PP^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{\alpha\star\beta}, \Psi^\alpha \cap \Psi^\beta).$$

By definition of runs strongly consistent with an agent context (2.75), where $\chi^{\alpha\star\beta} \in \mathscr{E}^{\alpha\star\beta}$ (the existence of such an agent contexts follows from Definition 3.2.2 of compatibility, Definition 3.0.1 of extensions and agent contexts being part of an extension and the assumption that the extensions in question are compatible w.r.t. $\star$)

$$R^{\chi^{\alpha\star\beta}} = R^{w(\chi^{\alpha\star\beta})} \cap (\Psi^\alpha \cap \Psi^\beta).$$

Thus by semantics of set intersection the Lemma follows. □

3.3.3 Safety Properties

To ensure that agent contexts χ remain non-excluding, we only encode liveness properties via admissibility conditions. Thus our remaining options to implement safety properties are with protocols and filter functions.

From Lemma 3.3.8 we already know that for any $P^\alpha \subseteq R^{trans}$ ($P^\alpha \neq \varnothing$), $P^\alpha = \overline{L^\alpha} \cap S'^\alpha$, where $\overline{L^\alpha}$ from Definition 3.3.6 is a liveness property (by Lemma 3.3.7) and S'^α is the smallest safety property containing P^α (or equivalently S'^α is the prefix and limit closure of P^α), formally defined as follows:

Definition 3.3.11. We define the smallest trace safety property containing $P \subseteq R^{trans}$, for $P \neq \varnothing$, as the prefix and limit closure of P, formally

$$S'(P) := \{h \in \mathscr{G} \mid (\exists r \in P)(\exists t \in \mathbb{N})\ r(t) = h\} \sqcup \{r \in R^{trans} \mid (\forall t \in \mathbb{N})(\exists r' \in P)\ r(t) = r'(t)\}. \tag{3.253}$$

The set of all trace safety properties is denoted by $\mathscr{T}$ and defined as

$$\mathscr{T} := \{S'(P) \mid \varnothing \neq P \subseteq R^{trans}\}. \tag{3.254}$$

Since safety properties expressed in terms of trace semantics (3.253) are very inconvenient for our purposes, as we usually reason on a round by round basis, we will provide an alternative definition of a safety property that maps global states (i.e. prefixes reached by runs in P^α) to the possible β sets that safely extend them.

Definition 3.3.12. An operational safety property S is defined as a function

$$S \colon \mathscr{G} \to 2^{2^{GEvents \sqcup \overline{GActions}}}, \tag{3.255}$$

which satisfies the following two conditions

(1) $(\exists h \in \mathscr{G})\ h_\epsilon = [] \ \wedge\ S(h) \neq \varnothing$

(2) $(\forall h \in \mathscr{G})\ h_\epsilon \neq [] \ \rightarrow\ \Big(((\exists h' \in \mathscr{G})(\exists X \in S(h'))\ h = update\,(h', X)\,) \ \leftrightarrow\ S(h) \neq \varnothing\Big)$.

Informally the first condition means that there has to exist at least one initial state that is safe. The second condition means that every non-initial state is safely extendable if and only if it is safely reachable.

The set of all operational safety properties is denoted by $\mathscr{O}$.

Definition 3.3.13. A transitional run $r \in R^{trans}$ satisfies an operational safety property $S \in \mathscr{O}$ iff

$$(\forall t \in \mathbb{N})\ \beta^t\,(r) \in S(r(t)). \tag{3.256}$$

Definition 3.3.14. We say an extension $\mathscr{E}$ adheres to an operational safety property $S \in \mathscr{O}$ iff

$$(\forall \chi \in \mathscr{E})(\forall r \in R^\chi)\ r \text{ satisfies } S. \tag{3.257}$$

In order to show that the operational safety property S from Definition 3.3.26 indeed coincides with the standard trace-based Definition 3.3.11 based on traces, we provide a bijective mapping from one to the other.

Definition 3.3.15. We define a construction F' of an operational safety property from a trace safety property $S' \in \mathscr{T}$, where F' is defined as

$$F'(S')(h) := \{\beta^t(r) \mid r \in S' \ \wedge \ t \in \mathbb{N} \ \wedge \ h = r(t)\}. \tag{3.258}$$

Lemma 3.3.16. *$F'(S') \in \mathscr{O}$ for any $S' \in \mathscr{T}$.*

Proof. Suppose by contradiction there exists some $S'^{\alpha} \in \mathscr{T}$ s.t. $F'(S'^{\alpha}) = S^{\alpha}$, where S^{α} violates the first operational safety property attribute ((1)). This implies that

$$(\forall h \in \mathscr{G})\ h_{\epsilon} \neq [] \ \vee \ S^{\alpha}(h) = \varnothing. \tag{3.259}$$

Since by Definition 3.3.11 $P^{\alpha} \neq \varnothing$, we get that there has to exist a run $r \in S'^{\alpha}$. Further, by prefix closure of S'^{α},

$$(\forall t \in \mathbb{N})\ r(t) \in S'^{\alpha}, \tag{3.260}$$

from which by universal instantiation we get that $r(0) \in S'^{\alpha}$. Since $S'^{\alpha} \subseteq R^{trans} \sqcup \mathscr{G}$, r is transitional, hence $r_{\epsilon}(0) = []$, from which by our assumption $S^{\alpha}(r(0)) = \varnothing$ follows. However, by Definition 3.3.15 of construction F', it follows that $\beta^0(r) \in S^{\alpha}(r(0))$, thus $S^{\alpha}(r(0)) \neq \varnothing$.

Next, suppose by contradiction there exists some $S'^{\alpha} \in \mathscr{T}$ s.t. $F'(S'^{\alpha}) = S^{\alpha}$, where S^{α} violates the second operational safety property attribute ((2)). This implies that there exists some $h \in \mathscr{G}$ s.t. $h_{\epsilon} \neq []$ and

$$(((\exists h' \in \mathscr{G})(\exists X \in S^{\alpha}(h'))\ h = update\,(h', X)) \ \wedge \ S^{\alpha}(h) = \varnothing) \vee \tag{3.261}$$
$$(((\forall h'' \in \mathscr{G})(\forall X' \in S^{\alpha}(h''))\ h \neq update\,(h'', X')) \ \wedge \ S^{\alpha}(h) \neq \varnothing). \tag{3.262}$$

Suppose (3.261) is true. This implies that there exists some $h' \in \mathscr{G}$ and some $X \in S^{\alpha}(h')$ such that $h = update\,(h', X)$. By Definition 3.3.15 of F' there exists a run $r \in S'^{\alpha}$ and a timestamp $t \in \mathbb{N}$ s.t.

$$r(t) = h' \ \wedge \ X = \beta^t(r). \tag{3.263}$$

By transitionality of r and Definition 2.2.31 of *update*

$$r(t+1) = h. \tag{3.264}$$

Again by Definition 3.3.15

$$\beta^{t+1}(r) \in S^{\alpha}(h), \tag{3.265}$$

hence $S^{\alpha}(h) \neq \varnothing$ and we conclude that (3.261) is false.

Suppose (3.262) is true. This implies by Definition 3.3.15 that there exists a run $r \in S'^{\alpha}$ and timestamp $t \in \mathbb{N} \setminus \{0\}$, where $h = r(t)$, since r is transitional and $h_{\epsilon} \neq []$. Further we get that

$$\beta^{t-1}(r) \in S^{\alpha}(r(t-1)). \tag{3.266}$$

Thus by Definition 2.2.31 of *update*

$$r(t) = update\left(r(t-1), \beta^{t-1}(r)\right) \tag{3.267}$$

and we conclude that (3.262) is false as well. □

Definition 3.3.17. We define

$$F\colon \mathscr{T} \mapsto \mathscr{O}, \tag{3.268}$$

where for any $S' \in \mathscr{T}$

$$F(S') := F'(S'), \tag{3.269}$$

which is indeed a mapping from $\mathscr{T}$ to $\mathscr{O}$ by Lemma 3.3.16.

Lemma 3.3.18. *F from Definition 3.3.17 is injective.*

Proof. Suppose by contradiction that the opposite is true: There exists $S'^{\alpha}, S'^{\beta} \in \mathscr{T}$ s.t. $S'^{\alpha} \neq S'^{\beta}$, but $F(S'^{\alpha}) = F(S'^{\beta})$. Since $S'^{\alpha} \neq S'^{\beta}$, either

(i) w.l.o.g. there exists some history $h \in S'^{\alpha}$ s.t. $h \notin S'^{\beta}$ or

(ii) w.l.o.g. there exists some run $r \in S'^{\alpha}$ s.t. $r \notin S'^{\beta}$. We show that this implies (i). Suppose by contradiction that there does not exist some $h \in S'^{\alpha}$ s.t. $h \notin S'^{\beta}$, meaning $(\forall h \in S'^{\alpha})\ h \in S'^{\beta}$. By limit closure of S'^{β} however it follows that $r \in S'^{\beta}$, hence there has to exist a history $h \in S'^{\alpha}$ such that $h \notin S'^{\beta}$.

Therefore, we can safely assume (i), i.e., w.l.o.g. that there exists some $h \in S'^{\alpha}$ s.t. $h \notin S'^{\beta}$. By Definition 3.3.17 of F, we get that $F(S'^{\beta})(h) = \varnothing$, as otherwise there would exist a run $r' \in S'^{\beta}$ and time $t' \in \mathbb{N}$ s.t. $r'(t') = h$, from which by prefix closure of S'^{β} it would follow that $h \in S'^{\beta}$. Since S'^{α} is the prefix closure of some non-empty set $P^{\alpha} \subseteq R^{trans}$ by Definition 3.3.11, we get that there exists some run $r \in S'^{\alpha}$ and time $t \in \mathbb{N}$ s.t. $r(t) = h$, additionally by Definition 3.3.17 of F, $\beta^t(r) \in F(S'^{\alpha})(h)$. Therefore $F(S'^{\alpha}) \neq F(S'^{\beta})$ and we are done. □

Definition 3.3.19. For some arbitrary $S \in \mathscr{O}$, we define

$$\widetilde{S'_0}^{S} := R^{trans} \tag{3.270}$$

$$\widetilde{S'_t}^{S} := \widetilde{S'_{t-1}}^{S} \setminus \{r \in R^{trans} \mid \beta^{t-1}(r) \notin S(r(t-1))\} \tag{3.271}$$

$$\widetilde{S'^{\alpha}_{\infty}}^{S} := \lim_{t' \to \infty} \widetilde{S'_{t'}}^{S} \tag{3.272}$$

$$\widetilde{S'}^{S} := \widetilde{S'^{\alpha}_{\infty}}^{S} \sqcup \{h \in \mathscr{G} \mid (\exists r \in \widetilde{S'^{\alpha}_{\infty}}^{S})(\exists t \in \mathbb{N})\ h = r(t)\}. \tag{3.273}$$

Note that the limit in (3.272) exists, as by (3.271) the set $\widetilde{S'_t}^{S}$ is non-increasing in t.

Lemma 3.3.20. *For $\widetilde{S'_m}^{\widetilde{S}}$ (for $m \in \mathbb{N} \setminus \{0\}$ and $\widetilde{S} \in \mathscr{O}$) from Definition 3.3.19, it holds that*

$$\widetilde{S'_m}^{\widetilde{S}} = \{r \in R^{trans} \mid (\forall t < m)\ \beta^t(r) \in \widetilde{S}(r(t))\}. \tag{3.274}$$

Proof. By induction:
Induction Hypothesis:

$$\widetilde{S'_m}^{\widetilde{S}} = \{r \in R^{trans} \mid (\forall t < m)\ \beta^t(r) \in \widetilde{S}(r(t))\}. \tag{3.275}$$

Base Case: For $m = 1$ by Definition 3.3.19 it follows that

$$\widetilde{S'_1}^{\widetilde{S}} = R^{trans} \setminus \{r \in R^{trans} \mid \beta^0(r) \notin \widetilde{S}(r(0))\} = \{r \in R^{trans} \mid \beta^0(r) \in \widetilde{S}(r(0))\}. \tag{3.276}$$

Induction Step: Suppose the induction hypothesis (3.275) holds for m, but by contradiction does not hold for $m+1$. There are two cases:

1. There exists a run $r' \in \widetilde{S'_{m+1}}^{\widetilde{S}}$ s.t. $r' \notin \{r \in R^{trans} \mid (\forall t < m+1)\ \beta^t(r) \in \widetilde{S}(r(t))\}$. This implies that there exists some timestamp $t' < m+1$ s.t.

$$\beta^{t'}(r') \notin \widetilde{S}(r'(t')). \tag{3.277}$$

We distinguish two cases:

a) $t' = m$:

$$r' \in \{r \in R^{trans} \mid \beta^m(r) \notin \widetilde{S}(r(m))\}. \tag{3.278}$$

Hence by Definition 3.3.19 of $\widetilde{S'_{m+1}}^{\widetilde{S}}$, r' would have been removed.

b) $t' < m$: This directly contradicts the induction hypothesis (3.275), as $r' \notin \widetilde{S'_{t'+1}}^{\widetilde{S}}$ and $\widetilde{S'_{m+1}}^{\widetilde{S}} \subseteq \widetilde{S'_{t'+1}}^{\widetilde{S}}$.

2. There exists a run $r' \in \{r \in R^{trans} \mid (\forall t < m+1)\ \beta^t(r) \in \widetilde{S}(r(t))\}$ s.t. $r' \notin \widetilde{S'_{m+1}}^{\widetilde{S}}$. We distinguish two cases regarding at which step r has been removed:

a) $r' \in \widetilde{S'_m}^{\widetilde{S}}$:

$$r' \in \{r \in R^{trans} \mid \beta^m(r) \notin \widetilde{S}(r(m))\}. \tag{3.279}$$

This implies that $\beta^m(r') \notin \widetilde{S}(r'(m))$ contradicting that

$$r' \in \{r \in R^{trans} \mid (\forall t < m+1)\ \beta^t(r) \in \widetilde{S}(r(t))\}. \tag{3.280}$$

b) $r' \notin \widetilde{S'_m}^{\widetilde{S}}$: This directly contradicts the induction hypothesis (3.275), thus concluding the induction step.

□

Lemma 3.3.21. *For $\widetilde{S'_\infty}^{S}$ from Definition 3.3.19 it holds that*

$$\widetilde{S'_\infty}^{S} = \{r \in R^{trans} \mid (\forall t \in \mathbb{N})\ \beta^t(r) \in S(r(t))\} \tag{3.281}$$

Proof. Follows from Lemma 3.3.20 and Definition 3.3.19. □

Lemma 3.3.22. *For $\widetilde{S'}^{S}$, from Definition 3.3.19, where $S \in \mathscr{O}$, it holds that $\widetilde{S'}^{S} \in \mathscr{T}$, i.e. $\widetilde{S'}^{S}$ is a trace safety property.*

Proof. From Definition 3.3.19, particularly (3.272) and (3.273), it follows that $\widetilde{S'^{\alpha}}^{S^{\alpha}}$ is the prefix and limit closure of $\widetilde{S'^{\alpha}}^{S^{\alpha}}$. □

Lemma 3.3.23. *F from Definition 3.3.17 is surjective.*

Proof. Suppose by contradiction that F is not surjective. This implies that there exists some $S \in \mathscr{O}$ s.t. for all $S' \in \mathscr{T}$, $F(S') \neq S$.

To arrive at a contradiction, we use the trace safety property $\widetilde{S'}^{S}$ from Definition 3.3.19. This is safe to use, as by Lemma 3.3.22 $\widetilde{S'}^{S} \in \mathscr{T}$. There are two cases causing $F(\widetilde{S'}^{S}) \neq S$:

1. There exists a run $r' \in \widetilde{S'}^{S}$ and timestamp $t' \in \mathbb{N}$ s.t. $\beta^{t'}(r') \notin S(r'(t'))$. Hence,
$$r' \in \{r \in R^{trans} \mid \beta^{t'}(r) \notin S(r(t'))\}, \tag{3.282}$$
such that by (3.271) $r' \notin \widetilde{S'_{t+1}}^{S}$, from which by (3.271), (3.272) and (3.273) $r' \notin \widetilde{S'}^{S}$ follows, which provides a contradiction.

2. There exists some history $\widetilde{h} \in \mathscr{G}$ and some $X \in S(\widetilde{h})$ s.t. for all runs $r \in \widetilde{S'}^{S}$ and all timestamps $t \in \mathbb{N}$
$$r(t) \neq \widetilde{h} \ \vee \ \beta^{t}(r) \neq X. \tag{3.283}$$
However, by Lemma 3.3.21 and Definition 3.3.19,
$$\widetilde{S'}^{S} = \{r \in R^{trans} \mid (\forall t \in \mathbb{N})\ \beta^{t}(r) \in S(r(t))\} \sqcup \{h \in \mathscr{G} \mid (\exists r \in \widetilde{S'}^{S})(\exists t \in \mathbb{N})\ h = r(t)\}. \tag{3.284}$$
By Definition 3.3.2 and 3.3.13, we get that there exists some run $r' \in R^{trans}$ and timestamp $t' \in \mathbb{N}$ s.t.
$$r'(t) = \widetilde{h} \ \wedge \ \beta^{t'}(r') = X. \tag{3.285}$$
By Definition 3.3.19 in particular (3.272), it follows that $r' \in \widetilde{S'}^{S}$, also providing the required contradiction.

Hence by definition of our construction (3.270), (3.271) and (3.272), $F(\widetilde{S'}^{S}) = S$. □

Lemma 3.3.24. *F from Definition 3.3.15 is bijective.*

Proof. Follows from Lemma 3.3.18 and 3.3.23. □

Remark 3.3.25. From this point on, whenever we refer to a safety property, we refer to the operational safety property. However, since by Lemma 3.3.24 the mapping F of Definition 3.3.17 is bijective, both the operational and the trace safety property can be easily obtained from one another via F.

Definition 3.3.26. As some extension $\mathscr{E}^{\alpha}$'s operational safety property S^{α} we refer to the smallest operational safety property that the extension $\mathscr{E}^{\alpha}$ adheres to. Using
$$P^{\alpha} = \bigcup_{\chi \in \mathscr{E}^{\alpha}} R^{\chi}, \tag{3.286}$$
we can calculate S'^{α} as the prefix and limit closure of P^{α} and then apply the bijective mapping F from Definition 3.3.17.

Adherence to safety properties does not completely capture the expressivity of an extension. One reason is the following: If the smallest safety property S^α of some extension $\mathscr{E}^{\mathscr{P}^\alpha}$ is a (proper) subset of some smallest safety property S^β of another extension $\mathscr{E}^{\mathscr{P}^\beta}$, obviously $\mathscr{E}^{\mathscr{P}^\alpha}$ also satisfies S^β, but cannot produce all runs that $\mathscr{E}^{\mathscr{P}^\beta}$ can produce (formal statements for these issues follow below). However, there is an even more subtle issue.

A demonstrative example is the restriction of the number of Byzantine agents. In (2.91) this restriction is implemented by an appropriate event filter function $filter_\epsilon^{\leq f}$, which is the most natural choice for this task: As the event filter function depends on the global history, it can easily check who is already Byzantine and remove inappropriate events. Nonetheless, one could also put this restriction in the set of environment protocols, where we could only allow protocols that make at most f agents Byzantine, no matter what the adversary chooses. Since environment protocols do not depend on the history however, any such f restriction environment protocol would be quite limited regarding which agents it can allow to become Byzantine and/or the point in time, at which agents become Byzantine, as it has no way to check, who has already failed in the run due to the adversary's choices. Anyway, despite these limitations in what events such a protocol can generate for every round w.r.t. making agents Byzantine, iterating over all such f restriction environment protocols (for a particular bound f) and all runs resulting from using these in appropriate agent contexts, yield the same runs, which an extension with the $\leq f$ filter can produce.

The difference between these implementations lies in what runs a single agent context can produce. Since the environment protocol is fixed for some agent context χ, when using the f environment protocol restriction, the runs strongly consistent with χ are very limited. Intuitively, it should be quite clear that no single agent context with such an f environment protocol exists that can produce all possible runs satisfying the f restriction on Byzantine agents if $1 \leq f < n$.

An easy counterexample is provided by all runs, where during the first two points in time all possible combinations of f agents can fail. An agent context that is part of an extension employing an f restriction environment protocols already fails to produce these runs: If the protocol were to allow any combination of Byzantine failures of a group of f agents during round 0.5 and 1.5, by our assumption that $1 \leq f < n$, we could find a combination of agent failures (by picking an appropriate choice of the adversary) for these two rounds, such that the combined number of Byzantine failures would exceed f.

To formalize this, we introduce the notion of *agent context power*.

Definition 3.3.27. We say an extension $\mathscr{E}^{\mathscr{P}^\beta}$ is *at least as powerful w.r.t. agent contexts* as $\mathscr{E}^{\mathscr{P}^\alpha}$ or $\mathscr{E}^{\mathscr{P}^\beta}$ has *at least the same agent context power* as $\mathscr{E}^{\mathscr{P}^\alpha}$ and write

$$\mathscr{E}^{\mathscr{P}^\alpha} \sqsubseteq \mathscr{E}^{\mathscr{P}^\beta} \tag{3.287}$$

iff

$$\bigcup_{\chi^\alpha \in \mathscr{E}^{\mathscr{P}^\alpha}} R^{\chi^\alpha} = \bigcup_{\chi^\beta \in \mathscr{E}^{\mathscr{P}^\beta}} R^{\chi^\beta} \tag{3.288}$$

and

$$(\forall \chi^\alpha \in \mathscr{E}^{\mathscr{P}^\alpha})(\exists \chi^\beta \in \mathscr{E}^{\mathscr{P}^\beta})\ R^{\chi^\alpha} = R^{\chi^\beta}. \tag{3.289}$$

Similarly we say $\mathscr{E}^\alpha$ is *exactly as powerful w.r.t. agent contexts* or has *exactly the same agent context power* as $\mathscr{E}^\beta$ and write

$$\mathscr{E}^\alpha \equiv \mathscr{E}^\beta \tag{3.290}$$

iff

$$\mathscr{E}^\alpha \sqsubseteq \mathscr{E}^\beta \ \wedge \ \mathscr{E}^\beta \sqsubseteq \mathscr{E}^\alpha. \tag{3.291}$$

Two extensions $\mathscr{E}^\alpha$, $\mathscr{E}^\beta$ are *incomparable w.r.t. agent context power* and we write

$$\mathscr{E}^\alpha \not\equiv \mathscr{E}^\beta \tag{3.292}$$

iff

$$\mathscr{E}^\alpha \not\sqsubseteq \mathscr{E}^\beta \quad \wedge \quad \mathscr{E}^\beta \not\sqsubseteq \mathscr{E}^\alpha \tag{3.293}$$

holds.

Lemma 3.3.28. *Given two compatible (w.r.t. $\star$) extensions $\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^\alpha, \Psi^\alpha)$, where $\mathscr{E}^\alpha$ adheres to safety property S, and $\mathscr{E}^\beta = (PP^\beta, IS^\beta, \tau^\beta, \Psi^\beta)$ such that*

$$\tau^\alpha \subseteq \tau^\beta \text{ (meaning that } filter^\alpha_\epsilon \subseteq filter^\beta_\epsilon \text{ and } (\forall i \in \mathcal{A})\, filter^\alpha_i \subseteq filter^\beta_i) \text{ and} \tag{3.294}$$

$$filter^\alpha_\epsilon \text{ and } (\forall i \in \mathcal{A})\, filter^\alpha_i \text{ are simply monotonic and idempotent for the domain } PD^{t-coh}_\epsilon, \ 2^{\overline{GActions}_1}, \ ..., \ 2^{\overline{GActions}_n} \tag{3.295}$$

the extensions $\mathscr{E}^{\alpha\star\beta}$ adheres to S, and simplifies to

$$\mathscr{E}^{\alpha\star\beta} = (PP^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^\alpha, \Psi^\alpha \cap \Psi^\beta).$$

Proof. From Lemmas 3.1.35, 3.1.36, 3.1.47, 3.1.58 and 3.1.68 it follows that for some $k \geq 1$, $i \in \mathcal{A}$, $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon$, $X_1 \in PD_1$, ..., $X_n \in PD_n$

$$\begin{aligned}
filter^{\alpha\circ\beta}_\epsilon (h, X_\epsilon, X_1, \ldots, X_n) &= filter^\alpha_\epsilon (h, X_\epsilon, X_1, \ldots, X_n) \\
filter^{\alpha\circ\beta}_i (X_1, \ldots, X_n, X_\epsilon) &= filter^\alpha_i (X_1, \ldots, X_n, X_\epsilon) \\
filter^{\beta\circ\alpha}_\epsilon (h, X_\epsilon, X_1, \ldots, X_n) &= filter^\alpha_\epsilon (h, X_\epsilon, X_1, \ldots, X_n) \\
filter^{\beta\circ\alpha}_i (X_1, \ldots, X_n, X_\epsilon) &= filter^\alpha_i (X_1, \ldots, X_n, X_\epsilon) \\
filter^{\alpha+\beta}_\epsilon (h, x_\epsilon, x_1, \ldots, X_n) &= filter^\alpha_\epsilon (h, X_\epsilon, X_1, \ldots, X_n) \\
filter^{\alpha+\beta}_i (X_1, \ldots, X_n, X_\epsilon) &= filter^\alpha_i (X_1, \ldots, X_n, X_\epsilon) \\
filter^{k\cdot(\alpha+\beta)}_\epsilon (h, X_\epsilon, X_1, \ldots, X_n) &= filter^\alpha_\epsilon (h, X_\epsilon, X_1, \ldots, X_n) \\
filter^{k\cdot(\alpha+\beta)}_i (X_1, \ldots, X_n, X_\epsilon) &= filter^\alpha_i (X_1, \ldots, X_n, X_\epsilon) \\
filter^{\alpha*\beta}_\epsilon (h, X_\epsilon, X_1, \ldots, X_n) &= filter^\alpha_\epsilon (h, X_\epsilon, X_1, \ldots, X_n) \\
filter^{\alpha*\beta}_i (X_1, \ldots, X_n, X_\epsilon) &= filter^\alpha_i (X_1, \ldots, X_n, X_\epsilon).
\end{aligned} \tag{3.296}$$

Since also

- $(P_\epsilon, P) \in PP^\alpha \cap PP^\beta \ \Rightarrow (P_\epsilon, P) \in PP^\alpha$ and
- $\mathscr{G}(0) \in IS^\alpha \cap IS^\beta \ \Rightarrow \mathscr{G}(0) \in IS^\alpha$ and

- $r \in \Psi^\alpha \cap \Psi^\beta \Rightarrow r \in \Psi^\alpha$ and

the Lemma follows by Definition 3.2.1 of extension combination. □

What follows is a special case of Lemma 3.3.28.

Corollary 3.3.29. *Given two extensions — compatible w.r.t. $\star$, $\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^\alpha, \Psi^\alpha)$ —, where $\mathscr{E}^\alpha$ adheres to safety property S, and $\mathscr{E}^\beta = (PP^\beta, IS^\beta, \tau^\alpha, \Psi^\beta)$ with matching transition templates and thus matching filter functions, such that*

$$\begin{gathered} filter_\epsilon^\alpha \text{ and } (\forall i \in \mathcal{A})\, filter_i^\alpha \text{ are idempotent for any} \\ X_\epsilon \in PD_\epsilon^{t-coh},\ X_1 \subseteq \overline{GActions}_1,\ \ldots,\ X_n \subseteq \overline{GActions}_n \end{gathered} \tag{3.297}$$

the extension $\mathscr{E}^{\alpha\star\beta}$ adheres to S as well and simplifies to

$$\mathscr{E}^{\alpha\star\beta} = (PP^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^\alpha, \Psi^\alpha \cap \Psi^\beta). \tag{3.298}$$

Proof. The proof is analogous to the proof for Lemma 3.3.28, with the only difference that instead of Lemmas 3.1.35, 3.1.36, 3.1.47, 3.1.58 and 3.1.68 the Corollaries 3.1.37, 3.1.48, 3.1.59 and 3.1.69 are used. □

Lemma 3.3.30. *Given two extensions — compatible w.r.t. reverse composition $(\beta \circ \alpha)$ — $\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha, IS^\alpha, \tau^\alpha, \Psi^\alpha)$ and $\mathscr{E}^\beta = (PP^\beta, IS^\beta, \tau^\beta, \Psi^\beta)$ (for $PP^\beta \subseteq \mathscr{C}_\epsilon \times \mathscr{C}$) such that for PD_ϵ^{t-coh}, $2^{\overline{GActions}_1}$, ..., $2^{\overline{GActions}_n}$ the event filter $filter_\epsilon^{\beta\circ\alpha}$ and the action filters $filter_i^\alpha$, $filter_i^\beta$ (for $i \in \mathcal{A}$) are idempotent and such that*

$$(\forall i \in \mathcal{A})\ filter_i^\alpha = filter_i^\beta, \tag{3.299}$$

it holds that

$$(\forall \chi^{\beta\circ\alpha} \in \mathscr{E}^{\beta\circ\alpha})(\exists \chi^\alpha \in \mathscr{E}^\alpha)\ R^{\chi^{\beta\circ\alpha}} \subseteq R^{\chi^\alpha}. \tag{3.300}$$

Proof. Completely analogous to the proof for Lemma 3.3.38 further below, where instead of Corollary 3.1.59 Corollary 3.1.37 is used to conclude that for all $i \in \mathcal{A}$ $filter_i^{\beta\circ\alpha} = filter_i^\alpha$. □

Corollary 3.3.31. *Given two extensions — compatible w.r.t. reverse composition $(\beta \circ \alpha)$ — $\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha, IS^\alpha, \tau^\alpha, \Psi^\alpha)$ and $\mathscr{E}^\beta = (PP^\beta, IS^\beta, \tau^\beta, \Psi^\beta)$ (for $PP^\beta \subseteq \mathscr{C}_\epsilon \times \mathscr{C}$) such that for PD_ϵ^{t-coh}, $2^{\overline{GActions}_1}$, ..., $2^{\overline{GActions}_n}$ the event filter $filter_\epsilon^{\beta\circ\alpha}$ and the action filters $filter_i^\alpha$, $filter_i^\beta$ (for $i \in \mathcal{A}$) are idempotent and such that*

$$(\forall i \in \mathcal{A})\ filter_i^\alpha = filter_i^\beta, \tag{3.301}$$

the extension $\mathscr{E}^{\beta\circ\alpha}$ adheres to S^α.

Proof. Immediately follows from Lemma 3.3.30. □

Lemma 3.3.32. *Given two extensions — compatible w.r.t. reverse composition* $(\beta \circ \alpha)$ *—* $\mathscr{E}^{\alpha} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha})$ *and* $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta})$ *(for* $PP^{\beta} \subseteq \mathscr{C}_{\epsilon} \times \mathscr{C}$*) such that for* PD_{ϵ}^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ *the event filter* $filter_{\epsilon}^{\beta\circ\alpha}$ *is idempotent and the action filters* $filter_i^{\alpha}$ *(for* $i \in \mathcal{A}$*) are idempotent and such that*

$$(\forall i \in \mathcal{A})\ filter_i^{\alpha} \subseteq filter_i^{\beta}, \tag{3.302}$$

it holds that

$$(\forall \chi^{\beta\circ\alpha} \in \mathscr{E}^{\beta\circ\alpha})(\exists \chi^{\alpha} \in \mathscr{E}^{\alpha})\ R^{\chi^{\beta\circ\alpha}} \subseteq R^{\chi^{\alpha}}. \tag{3.303}$$

Proof. Completely analogous to the proof for Lemma 3.3.40 further below, where instead of Lemma 3.1.58 Lemma 3.1.35 is used to conclude that for all $i \in \mathcal{A}$ $filter_i^{\beta\circ\alpha} = filter_i^{\alpha}$. □

Corollary 3.3.33. *Given two extensions — compatible w.r.t. reverse composition* $(\beta \circ \alpha)$ *—* $\mathscr{E}^{\alpha} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha})$ *and* $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta})$ *(for* $PP^{\beta} \subseteq \mathscr{C}_{\epsilon} \times \mathscr{C}$*) such that for* PD_{ϵ}^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ *the event filter* $filter_{\epsilon}^{\beta\circ\alpha}$ *is idempotent and the action filters* $filter_i^{\alpha}$ *(for* $i \in \mathcal{A}$*) are idempotent and such that*

$$(\forall i \in \mathcal{A})\ filter_i^{\alpha} \subseteq filter_i^{\beta}, \tag{3.304}$$

the extension $\mathscr{E}^{\beta\circ\alpha}$ *adheres to* S^{α}.

Proof. Immediately follows from Lemma 3.3.32. □

Lemma 3.3.34. *Given two extensions — compatible w.r.t. composition* $(\alpha \circ \beta)$ *—* $\mathscr{E}^{\alpha} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha})$ *and* $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta})$ *(for* $PP^{\beta} \subseteq \mathscr{C}_{\epsilon} \times \mathscr{C}$*) such that for* PD_{ϵ}^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ *the action filters* $filter_i^{\alpha}$, $filter_i^{\beta}$ *(for* $i \in \mathcal{A}$*) are idempotent and such that*

$$(\forall i \in \mathcal{A})\ filter_i^{\alpha} = filter_i^{\beta}, \tag{3.305}$$

it holds that

$$(\forall \chi^{\alpha\circ\beta} \in \mathscr{E}^{\alpha\circ\beta})(\exists \chi^{\alpha} \in \mathscr{E}^{\alpha})\ R^{\chi^{\alpha\circ\beta}} \subseteq R^{\chi^{\alpha}}. \tag{3.306}$$

Proof. By Definition 3.2.1 of extension combination, the resulting combined extension is

$$\mathscr{E}^{\alpha\circ\beta} = ((\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha}) \cap PP^{\beta}, IS^{\alpha} \cap IS^{\beta}, \tau^{\alpha\circ\beta}, \Psi^{\alpha} \cap \Psi^{\beta}). \tag{3.307}$$

For some $\chi^{\alpha\circ\beta} \in \mathscr{E}^{\alpha\circ\beta}$, where

$$\chi^{\alpha\circ\beta} = \left(\left(P_{\epsilon}, \mathscr{G}(0), \tau^{\alpha\circ\beta}, \Psi^{\alpha} \cap \Psi^{\beta}\right), P\right), \tag{3.308}$$

since all sets for any $t \in \mathbb{N}$ of $P_{\epsilon}(t)$ are t-coherent and the set of all t-coherent sets is downward closed by Lemma 2.2.3, it follows that

- $(h \in \mathscr{G})(\forall t \in \mathbb{N})(\forall X_{\epsilon} \in P_{\epsilon}(t))(\forall X_1 \in P_1(h_1)) \ldots (\forall X_n \in P_n(h_n))$ $filter_{\epsilon}^{\beta}\left(h, X_{\epsilon}, label_1\left(X_1, t\right), \ldots, label_n\left(X_n, t\right)\right)$ is t-coherent.
- Hence we can find an environment protocol $\widetilde{P_{\epsilon}} \in \mathscr{C}_{\epsilon}$ such that

$$\begin{aligned}(\forall h \in \mathscr{G})(\forall t \in \mathbb{N})(\forall X_{\epsilon} \in P_{\epsilon}(t))(\forall X_1 \in P_1(h_1)) \ldots (\forall X_n \in P_n(h_n))\\ filter_{\epsilon}^{\beta}\left(h, X_{\epsilon}, label_1\left(X_1, t\right), \ldots, label_n\left(X_n, t\right)\right) \in \widetilde{P_{\epsilon}}(t),\end{aligned} \tag{3.309}$$

- thus, with the environment protocol $\widetilde{P_\epsilon}$ from (3.309) we can construct an agent context $\chi^\alpha \in \mathscr{E}^\alpha$, where

$$\chi^\alpha = \left(\left(\widetilde{P_\epsilon}, \mathscr{G}(0), \tau^\alpha, \Psi^\alpha\right), P\right), \tag{3.310}$$

  which we will show to produce all runs from $R^{\chi^{\alpha\circ\beta}}$. Note that $P \in \mathscr{C}^\alpha \cap \mathscr{C}^\beta$ trivially implies (by semantics of set intersection) that $P \in \mathscr{C}^\alpha$ and similarly if $\mathscr{G}(0) \in IS^\alpha \cap IS^\beta$ this again trivially implies that $\mathscr{G}(0) \in IS^\alpha$. Therefore, χ^α can indeed always be constructed as such.

We conduct the remaining part of the proof by induction, where we will first show that $R^{w(\chi^{\alpha\circ\beta})} \subseteq R^{w(\chi^\alpha)}$.
Induction Hypothesis: For some $t \in \mathbb{N}$ it holds that

$$(\forall r \in R^{w(\chi^{\alpha\circ\beta})})(\exists \widetilde{r} \in R^{w(\chi^\alpha)})(\forall t' \leq t)\ r(t') = \widetilde{r}(t'). \tag{3.311}$$

Base Case: For $t = 0$, by definition of the contexts $\chi^{\alpha\circ\beta}$ (3.308) and χ^α (3.310), since both use identical sets of initial states $\mathscr{G}(0)$ the statement trivially follows.
Induction Step: Suppose the induction hypothesis (3.311) holds for some $t \in \mathbb{N}$, in particular for two runs $r' \in R^{w(\chi^{\alpha\circ\beta})}$ and $\widetilde{r'} \in R^{w(\chi^\alpha)}$ who are equal up to time t, we will now show that it also holds for $t+1$, meaning that $r'(t+1) = \widetilde{r'}(t+1)$ or there exists some other run $\widetilde{r''} \in R^{w(\chi^\alpha)}$ such that $(\forall t'' \leq t+1)\ r'(t'') = \widetilde{r''}(t'')$.

First we look at events. From the definition of the protocol $\widetilde{P_\epsilon}$ (3.309) it follows that it provides all possibilities regarding what $filter_\epsilon^\beta$ could return for any combination of global history, event set (from P_ϵ) and action sets (from P_1, ..., P_n). Moreover, suppose that $X'_\epsilon \in P_\epsilon(t)$, $X'_1 \in P_1(r'_1(t))$, ..., $X'_n \in P_n(r'_n(t))$ are the sets chosen by the protocols in run r' for round t.5, we get that

$$(\exists X_\epsilon \in \widetilde{P_\epsilon}(t))\ X_\epsilon = filter_\epsilon^\beta\left(r'(t), X'_\epsilon, label_1\left(X'_1, t\right), \ldots, label_n\left(X'_n, t\right)\right). \tag{3.312}$$

Considering the application of filter $filter_\epsilon^\alpha$, since the same joint protocols are used in both agent contexts, also the same output sets can be produced. Using the induction hypothesis (3.311) $r'(t) = \widetilde{r'}(t)$ and $\widetilde{X_\epsilon}$ obtained by existential instantiation of X_ϵ in (3.312), we find

$$\begin{aligned}
&filter_\epsilon^\alpha\left(\widetilde{r'}(t), \widetilde{X_\epsilon}, label_1\left(X'_1, t\right), \ldots, label_n\left(X'_n, t\right)\right) = \\
&filter_\epsilon^\alpha\left(r'(t), \widetilde{X_\epsilon}, label_1\left(X'_1, t\right), \ldots, label_n\left(X'_n, t\right)\right) = \\
&filter_\epsilon^\alpha\left(r'(t), filter_\epsilon^\beta\left(r'(t), X'_\epsilon, label_1\left(X'_1, t\right), \ldots, label_n\left(X'_n, t\right)\right), label_1\left(X'_1, t\right), \ldots, label_n\left(X'_n, t\right)\right).
\end{aligned} \tag{3.313}$$

This means there exists a run $\widetilde{r''} \in R^{w(\chi^\alpha)}$, such that

$$(\forall t' \leq t)\ \widetilde{r''}(t') = r'(t') \quad \wedge \quad \beta_\epsilon^t\left(\widetilde{r''}\right) = \beta_\epsilon^t\left(r'\right). \tag{3.314}$$

Next we take a look at actions, especially the filtering part. Since (for all $i \in \mathcal{A}$) $filter_i^\alpha = filter_i^\beta$ and idempotence was assumed for all action filters, by Corollary 3.1.37 the combined filter simplifies for any sets $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \in 2^{\overline{GActions_1}}$, ..., $X_n \in 2^{\overline{GActions_n}}$ to

$$(\forall i \in \mathcal{A})\ filter_i^{\alpha\circ\beta}\left(X_1, \ldots, X_n, X_\epsilon\right) = filter_i^\alpha\left(X_1, \ldots, X_n, X_\epsilon\right) = filter_i^\beta\left(X_1, \ldots, X_n, X_\epsilon\right). \tag{3.315}$$

As the agent protocols from agent contexts χ^{α} and $\chi^{\alpha\circ\beta}$ are the same, the only concern left is the set of events passed to the action filter function. However, we have already concluded that the agent context χ^{α} can produce a run $\widetilde{r''}$ for which (3.314) holds. For the two runs $r', \widetilde{r''}$ — using the event and actions sets from above - this means that

$$\begin{aligned}(\forall i \in \mathcal{A})\ filter_i^{\alpha\circ\beta}\left(X_1', \ldots, X_n', filter_\epsilon^{\alpha\circ\beta}\left(r'(t), X_\epsilon', X_1', \ldots, X_n'\right)\right) = \\ filter_i^{\alpha}\left(X_1', \ldots, X_n', filter_\epsilon^{\alpha}\left(\widetilde{r''}, \widetilde{X_\epsilon}, X_1', \ldots, X_n'\right)\right).\end{aligned} \tag{3.316}$$

Hence we are done with the induction step and can now safely conclude that

$$R^{w(\chi^{\alpha\circ\beta})} \subseteq R^{w(\chi^{\alpha})}. \tag{3.317}$$

Since the admissibility condition $\Psi^{\alpha\circ\beta}$ of $\mathscr{E}^{\alpha\circ\beta}$ is $\Psi^{\alpha} \cap \Psi^{\beta}$ and

$$\begin{aligned}R^{\chi^{\alpha\circ\beta}} &= R^{w(\chi^{\alpha\circ\beta})} \cap \Psi^{\alpha\circ\beta} \\ R^{\chi^{\alpha}} &= R^{w(\chi^{\alpha})} \cap \Psi^{\alpha},\end{aligned} \tag{3.318}$$

from (3.317) and (3.318) we finally get that

$$R^{\chi^{\alpha\circ\beta}} \subseteq R^{\chi^{\alpha}}. \tag{3.319}$$

□

Corollary 3.3.35. *Given two extensions — compatible w.r.t. composition $(\alpha \circ \beta)$ — $\mathscr{E}^{\alpha} = (\mathscr{C}_\epsilon \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha})$ and $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta})$ (for $PP^{\beta} \subseteq \mathscr{C}_\epsilon \times \mathscr{C}$) such that for PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ the action filters $filter_i^{\alpha}$, $filter_i^{\beta}$ (for $i \in \mathcal{A}$) are idempotent and such that*

$$(\forall i \in \mathcal{A})\ filter_i^{\alpha} = filter_i^{\beta}, \tag{3.320}$$

the extension $\mathscr{E}^{\alpha\circ\beta}$ adheres to S^{α}.

Proof. Immediately follows from Lemma 3.3.34. □

Lemma 3.3.36. *Given two extensions — compatible w.r.t. composition $(\alpha \circ \beta)$ — $\mathscr{E}^{\alpha} = (\mathscr{C}_\epsilon \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha})$ and $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta})$ (for $PP^{\beta} \subseteq \mathscr{C}_\epsilon \times \mathscr{C}$) such that for PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ the action filters $filter_i^{\alpha}$ (for $i \in \mathcal{A}$) are idempotent, simply monotonic and such that*

$$(\forall i \in \mathcal{A})\ filter_i^{\alpha} \subseteq filter_i^{\beta}, \tag{3.321}$$

it holds that

$$(\forall \chi^{\alpha\circ\beta} \in \mathscr{E}^{\alpha\circ\beta})(\exists \chi^{\alpha} \in \mathscr{E}^{\alpha})\ R^{\chi^{\alpha\circ\beta}} \subseteq R^{\chi^{\alpha}}. \tag{3.322}$$

Proof. Analogous to the proof for Lemma 3.3.34, where instead of Corollary 3.1.37 Lemma 3.1.36 is used to conclude that for all $i \in \mathcal{A}$ $filter_i^{(\alpha\circ\beta)} = filter_i^{\alpha}$. □

Corollary 3.3.37. *Given two extensions — compatible w.r.t. composition $(\alpha \circ \beta)$ — $\mathscr{E}^{\alpha} = (\mathscr{C}_\epsilon \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha})$ and $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta})$ (for $PP^{\beta} \subseteq \mathscr{C}_\epsilon \times \mathscr{C}$) such that for PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ the action filters $filter_i^{\alpha}$ (for $i \in \mathcal{A}$) are idempotent, simply monotonic and such that*

$$(\forall i \in \mathcal{A})\ filter_i^{\alpha} \subseteq filter_i^{\beta}, \tag{3.323}$$

the extension $\mathscr{E}^{\alpha\circ\beta}$ adheres to S^{α}.

Proof. Immediately follows from Lemma 3.3.36. □

Lemma 3.3.38. *Given two extensions — compatible w.r.t. k-intersection for some* $k \geq 1$ $(k \cdot (\alpha+\beta))$ — $\mathscr{E}^{\alpha} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha})$ *and* $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta})$ *(for* $PP^{\beta} \subseteq \mathscr{C}_{\epsilon} \times \mathscr{C}$*) such that for* PD_{ϵ}^{t-coh}, $2^{\overline{GActions}_1}$, ..., $2^{\overline{GActions}_n}$ *the event filter* $filter_{\epsilon}^{k\cdot(\alpha+\beta)}$ *for some* $k \geq 1$ *and its action filters are idempotent and such that*

$$(\forall i \in \mathcal{A})\ filter_i^{\alpha} = filter_i^{\beta}, \tag{3.324}$$

it holds that

$$(\forall \chi^{k\cdot(\alpha+\beta)} \in \mathscr{E}^{k\cdot(\alpha+\beta)})(\exists \chi^{\alpha} \in \mathscr{E}^{\alpha})\ R^{\chi^{k\cdot(\alpha+\beta)}} \subseteq R^{\chi^{\alpha}} \tag{3.325}$$

Proof. By Definition 3.2.1 of extension combination, the resulting combined extension is

$$\mathscr{E}^{k\cdot(\alpha+\beta)} = ((\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha}) \cap PP^{\beta}), IS^{\alpha} \cap IS^{\beta}, \tau^{k\cdot(\alpha+\beta)}, \Psi^{\alpha} \cap \Psi^{\beta}). \tag{3.326}$$

For some $\chi^{k\cdot(\alpha+\beta)} \in \mathscr{E}^{k\cdot(\alpha+\beta)}$, where $\chi^{k\cdot(\alpha+\beta)} = \left(\left(P_{\epsilon}, \mathscr{G}(0), \tau^{k\cdot(\alpha+\beta)}, \Psi^{\alpha} \cap \Psi^{\beta}\right), P\right)$, since all runs $r \in R^{\chi^{k\cdot(\alpha+\beta)}}$ are transitional and every subset of a t-coherent set is t-coherent by Lemma 2.2.3, it follows that

- $(\forall r \in R^{\chi^{k\cdot(\alpha+\beta)}})(\forall t \in \mathbb{N})\ \beta_{\epsilon}^{t}(r)$ is t-coherent.
- Hence, we can find an environment protocol $\widetilde{P_{\epsilon}} \in \mathscr{C}_{\epsilon}$ such that

$$(\forall r \in R^{\chi^{k\cdot(\alpha+\beta)}})(\forall t \in \mathbb{N})\ \beta_{\epsilon}^{t}(r) \in \widetilde{P_{\epsilon}}(t), \tag{3.327}$$

- thus, with the environment protocol $\widetilde{P_{\epsilon}}$ from (3.327) we can construct an agent context $\chi^{\alpha} \in \mathscr{E}^{\alpha}$, where

$$\chi^{\alpha} = \left(\left(\widetilde{P_{\epsilon}}, \mathscr{G}(0), \tau^{\alpha}, \Psi^{\alpha}\right), P\right), \tag{3.328}$$

  which we will show produces all runs from $R^{\chi^{k\cdot(\alpha+\beta)}}$. Note that since the joint protocol P is used in agent context $\chi^{k\cdot(\alpha+\beta)}$, trivially $P \in \mathscr{C}^{\alpha}$ follows.

First we look at the events from all the runs, which the agent context (3.328) produces. Considering the result of the combined event filter for a run $r \in R^{\chi^{k\cdot(\alpha+\beta)}}$ and timestamp $t \in \mathbb{N}$, it has to hold that

$$\begin{gathered}(\exists X_{\epsilon} \in P_{\epsilon}(t))(\exists X_1 \in P_1(r_1(t))) \ldots (\exists X_n \in P_1(r_n(t))) \\ filter_{\epsilon}^{k\cdot(\alpha+\beta)}(r(t), X_{\epsilon}, X_1, \ldots, X_n) = \beta_{\epsilon}^{t}(r).\end{gathered} \tag{3.329}$$

Since by assumption $filter_{\epsilon}^{k\cdot(\alpha+\beta)}$ is idempotent for any $X_{\epsilon} \in P_{\epsilon}(t)$, $X_1 \in P(r_1(t))$, ..., $X_n \in P(r_n(t))$ we get that

$$filter_{\epsilon}^{k\cdot(\alpha+\beta)}\left(r(t), \beta_{\epsilon}^{t}(r), X_1, \ldots, X_n\right) = \beta_{\epsilon}^{t}(r). \tag{3.330}$$

Further from (3.330) and Lemma 3.1.70, it follows that

$$filter_{\epsilon}^{\alpha}\left(r(t), \beta_{\epsilon}^{t}(r), X_1, \ldots, X_n\right) = \beta_{\epsilon}^{t}(r). \tag{3.331}$$

Hence, from the definition of the protocol $\widetilde{P_\epsilon}$ (3.327), we can conclude that the agent context χ^α from (3.328) can produce runs with the same events as runs generated by $\chi^{k\cdot(\alpha+\beta)}$.

Next we take a look at actions. However, since (for all $i \in \mathcal{A}$) $filter_i^\alpha = filter_i^\beta$ and idempotence was assumed for all filters, by Corollary 3.1.59 the combined filter simplifies for any sets $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \in 2^{\overline{GActions_1}}$, ..., $X_n \in 2^{\overline{GActions_n}}$ to

$$(\forall i \in \mathcal{A})\ filter_i^{k\cdot(\alpha+\beta)}(X_1, \ldots, X_n, X_\epsilon) = filter_i^\alpha(X_1, \ldots, X_n, X_\epsilon) = filter_i^\beta(X_1, \ldots, X_n, X_\epsilon). \tag{3.332}$$

As the agent protocols from agent contexts χ^α and $\chi^{k\cdot(\alpha+\beta)}$ are the same, the only concern left is the set of events passed to the action filter function. However we have already concluded that the agent context χ^α can produce runs that have the same β_ϵ sets (for the same attempted actions $X_1 \subseteq \overline{GActions_1}$, ..., $X_n \subseteq \overline{GActions_n}$) as runs from $\chi^{k\cdot(\alpha+\beta)}$. Hence we are done for actions.

Considering the initial state as $\mathscr{G}(0) \in IS^\alpha \cap IS^\beta$, we can conclude that

$$R^{w(\chi^{k\cdot(\alpha+\beta)})} \subseteq R^{w(\chi^\alpha)} \tag{3.333}$$

Since the admissibility condition $\Psi^{k\cdot(\alpha+\beta)}$ of $\mathscr{E}^{k\cdot(\alpha+\beta)}$ is $\Psi^\alpha \cap \Psi^\beta$ and

$$R^{\chi^{k\cdot(\alpha+\beta)}} = R^{w(\chi^{k\cdot(\alpha+\beta)})} \cap \Psi^{k\cdot(\alpha+\beta)}, \tag{3.334}$$

from (3.333) we finally get that

$$R^{\chi^{k\cdot(\alpha+\beta)}} \subseteq R^{\chi^\alpha}. \tag{3.335}$$

□

Corollary 3.3.39. *Given two extensions — compatible w.r.t. k-intersection for some $k \geq 1$ $(k\cdot(\alpha+\beta))$ — $\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha, IS^\alpha, \tau^\alpha, \Psi^\alpha)$ and $\mathscr{E}^\beta = (PP^\beta, IS^\beta, \tau^\beta, \Psi^\beta)$ (for $PP^\beta \subseteq \mathscr{C}_\epsilon \times \mathscr{C}$) such that for PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ the event filter $filter_\epsilon^{k\cdot(\alpha+\beta)}$ for some $k \geq 1$ and the action filters are idempotent and such that*

$$(\forall i \in \mathcal{A})\ filter_i^\alpha = filter_i^\beta, \tag{3.336}$$

the extension $\mathscr{E}^{k\cdot(\alpha+\beta)}$ adheres to S^α.

Proof. Immediately follows from Lemma 3.3.38. □

Lemma 3.3.40. *Given two extensions — compatible w.r.t. k-intersection for some $k \geq 1$ $(k\cdot(\alpha+\beta))$ — $\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha, IS^\alpha, \tau^\alpha, \Psi^\alpha)$ and $\mathscr{E}^\beta = (PP^\beta, IS^\beta, \tau^\beta, \Psi^\beta)$ (for $PP^\beta \subseteq \mathscr{C}_\epsilon \times \mathscr{C}$) such that for PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ the event filter $filter_\epsilon^{k\cdot(\alpha+\beta)}$ for some $k \geq 1$ is idempotent and the action filters $filter_i^\alpha$ (for $i \in \mathcal{A}$) are idempotent, simply monotonic and such that*

$$(\forall i \in \mathcal{A})\ filter_i^\alpha \subseteq filter_i^\beta, \tag{3.337}$$

it holds that

$$(\forall \chi^{k\cdot(\alpha+\beta)} \in \mathscr{E}^{k\cdot(\alpha+\beta)})(\exists \chi^\alpha \in \mathscr{E}^\alpha)\ R^{\chi^{k\cdot(\alpha+\beta)}} \subseteq R^{\chi^\alpha}. \tag{3.338}$$

Proof. Analogous to the proof of Lemma 3.3.38, where instead of Corollary 3.1.59 Lemma 3.1.58 has to be used to conclude that for all $i \in \mathcal{A}$

$$filter_i^{k\cdot(\alpha+\beta)} = filter_i^{\alpha}. \tag{3.339}$$

□

Corollary 3.3.41. *Given two extensions — compatible w.r.t. k-intersection for some $k \geq 1$ $(k\cdot(\alpha+\beta))$ — $\mathscr{E}^{\alpha} = (\mathscr{C}_\epsilon \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha})$ and $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta})$ (for $PP^{\beta} \subseteq \mathscr{C}_\epsilon \times \mathscr{C}$) such that for PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ the event filter $filter_\epsilon^{k\cdot(\alpha+\beta)}$ is idempotent and the action filters $filter_i^{\alpha}$ (for $i \in \mathcal{A}$) are idempotent, simply monotonic and such that*

$$(\forall i \in \mathcal{A})\ filter_i^{\alpha} \subseteq filter_i^{\beta}, \tag{3.340}$$

the extension $\mathscr{E}^{k\cdot(\alpha+\beta)}$ adheres to S^{α}.

Proof. Immediately follows from Lemma 3.3.40. □

Lemma 3.3.42. *Given two extensions — compatible w.r.t. fixpoint intersection $(\alpha * \beta)$ — $\mathscr{E}^{\alpha} = (\mathscr{C}_\epsilon \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha})$ and $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta})$, whose action filters are idempotent for PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ and such that*

$$(\forall i \in \mathcal{A})\ filter_i^{\alpha} = filter_i^{\beta}, \tag{3.341}$$

it holds that

$$(\forall \chi^{\alpha*\beta} \in \mathscr{E}^{\alpha*\beta})(\exists \chi^{\alpha} \in \mathscr{E}^{\alpha})\ R^{\chi^{\alpha*\beta}} \subseteq R^{\chi^{\alpha}}. \tag{3.342}$$

Proof. Analogous to the proof for Lemma 3.3.38, where the idempotence of $filter_\epsilon^{\alpha*\beta}$ follows from Lemma 3.1.65 and instead of Corollary 3.1.59 Corollary 3.1.69 is used to conclude that for all $i \in \mathcal{A}$ $filter_i^{\alpha*\beta} = filter_i^{\alpha}$. □

Corollary 3.3.43. *Given two extensions — compatible w.r.t. fixpoint intersection $(\alpha * \beta)$ — $\mathscr{E}^{\alpha} = (\mathscr{C}_\epsilon \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha})$ and $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta})$ such that the action filters are idempotent for PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ and such that*

$$(\forall i \in \mathcal{A})\ filter_i^{\alpha} = filter_i^{\beta}, \tag{3.343}$$

the resulting extension $\mathscr{E}^{\alpha*\beta}$ adheres to S^{α}.

Proof. Immediately follows from Lemma 3.3.42. □

Lemma 3.3.44. *Given two extensions — compatible w.r.t. fixpoint intersection $(\alpha * \beta)$ — $\mathscr{E}^{\alpha} = (\mathscr{C}_\epsilon \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha})$ and $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta})$ the action filters $filter_i^{\alpha}$ (for all $i \in \mathcal{A}$) are idempotent and simply monotonic for PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ and such that*

$$(\forall i \in \mathcal{A})\ filter_i^{\alpha} \subseteq filter_i^{\beta}, \tag{3.344}$$

it holds that

$$(\forall \chi^{\alpha*\beta} \in \mathscr{E}^{\alpha*\beta})(\exists \chi^{\alpha} \in \mathscr{E}^{\alpha})\ R^{\chi^{\alpha*\beta}} \subseteq R^{\chi^{\alpha}}. \tag{3.345}$$

Proof. Analogous to the proof for Lemma 3.3.38, where the idempotence of $filter_{\epsilon}^{\alpha*\beta}$ follows from Lemma 3.1.65 and instead of Corollary 3.1.59 Lemma 3.1.68 is used to conclude that for all $i \in \mathcal{A}$ $filter_i^{\alpha*\beta} = filter_i^{\alpha}$. □

Corollary 3.3.45. *Given two extensions — compatible w.r.t. fixpoint intersection ($\alpha * \beta$) — $\mathscr{E}^{\alpha} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha})$ and $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta})$ the action filters $filter_i^{\alpha}$ (for all $i \in \mathcal{A}$) are idempotent and simply monotonic for PD_{ϵ}^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ and such that*

$$(\forall i \in \mathcal{A})\ filter_i^{\alpha} \subseteq filter_i^{\beta}, \tag{3.346}$$

the resulting extension $\mathscr{E}^{\alpha*\beta}$ adheres to S^{α}.

Proof. Immediately follows from Lemma 3.3.44. □

Lemma 3.3.46. *Given two safety properties S, S' such that $(\forall h \in \mathscr{G})\ S'(h) \subseteq S(h)$ and an extension $\mathscr{E}$ adhering to S', $\mathscr{E}$ also adheres to S.*

Proof. Suppose by contradiction that $\mathscr{E}$ adheres to S' but not $S \Rightarrow$

$$(\exists \chi \in \mathscr{E})(\exists r \in R^{\chi})(\exists t \in \mathbb{N})(\exists X \notin S(r(t)))\ \beta^t(r) = X \tag{3.347}$$

However, since $(\forall h \in \mathscr{G})\ S'(h) \subseteq S(h)$ is assumed, it follows that for such a particular set X', run r' and time t' (by existential instantiations from from (3.347)) $X' \notin S'(r(t))$ as well. Since $\mathscr{E}$ adheres to S' by Definitions 3.3.13 and 3.3.14 it holds that

$$(\forall \chi \in \mathscr{E})(\forall r \in R^{\chi})(\forall t \in \mathbb{N})(\forall X \notin S'(r(t)))\ \beta^t(r) \neq X$$

meaning X' never appears in any run of an agent context part of $\mathscr{E}$. □

Lemma 3.3.47. *An extension $\mathscr{E}$ that adheres to safety properties S^{α} and S^{β} also adheres to a safety property S^{γ}, where*

$$(\forall h \in \mathscr{G})\ S^{\gamma}(h) = S^{\alpha}(h) \cap S^{\beta}(h) \tag{3.348}$$

Proof. Suppose by contradiction $\mathscr{E}$ does not adhere to S^{γ}. This implies that

$$(\exists \chi \in \mathscr{E})(\exists r \in R^{\chi})(\exists t \in \mathbb{N})\ \beta^t(r) \notin S^{\alpha}(r(t)) \cap S^{\beta}(r(t)). \tag{3.349}$$

By semantics of set intersection we further get

$$(\exists \chi \in \mathscr{E})(\exists r \in R^{\chi})(\exists t \in \mathbb{N})\ \beta^t(r) \notin S^{\alpha}(r(t))\ \vee\ \beta^t(r) \notin S^{\beta}(r(t)), \tag{3.350}$$

which however contradicts our assumption that $\mathscr{E}$ adheres to S^{α} and S^{β}:

$$(\forall \chi \in \mathscr{E})(\forall r \in R^{\chi})(\forall t \in \mathbb{N})\ \beta^t(r) \in S^{\alpha}(r(t))\ \wedge\ \beta^t(r) \in S^{\beta}(r(t)). \tag{3.351}$$

□

Lemma 3.3.48. *For any extension* $\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}, IS^\alpha, \tau^{\alpha,N}, \Psi^\alpha)$ *(that adheres to the smallest safety property* S^α*), where in* $\tau^{\alpha,N}$ *the idempotent (for the domain* PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$*) event filter* $filter_\epsilon^\alpha$ *and the neutral action filters for all* $i \in \mathcal{A}$ *are used, for any* $h \in \mathscr{G}$ *it holds that*

$$X \in S^\alpha(h) \quad \Longrightarrow \quad filter_\epsilon^\alpha\left(h, X_\epsilon, X_1, \ldots, X_n\right) = X_\epsilon, \tag{3.352}$$

where $X = X_\epsilon \sqcup X_1 \sqcup \cdots \sqcup X_n$.

Proof. Suppose by contradiction that this is not true. By the basic filter property of $filter_\epsilon^\alpha$ it follows that there exists a $h \in \mathscr{G}$ and $X \in S^\alpha(h)$ such that

$$filter_\epsilon^\alpha\left(h, X_\epsilon, X_1, \ldots, X_n\right) \subset X_\epsilon, \tag{3.353}$$

where $X = X_\epsilon \sqcup X_1 \sqcup \cdots \sqcup X_n$.

Since $X \in S^\alpha(h)$ and S^α is the smallest safety property of $\mathscr{E}^\alpha$, there has to exist some $X^{sup} \supset X$ (where $X^{sup} = X_\epsilon^{sup} \sqcup X_{ag}^{sup}$, where $X_\epsilon^{sup} \in PD_\epsilon^{t-coh}$ and $X_{ag}^{sup} \in 2^{\overline{GActions}}$) such that

$$filter_\epsilon^\alpha\left(h, X_\epsilon^{sup}, X_1^{sup}, \ldots, X_n^{sup}\right) = X_\epsilon, \tag{3.354}$$

where $X^{sup} = X_\epsilon^{sup} \sqcup X_1^{sup} \sqcup \cdots \sqcup X_n^{sup}$. However, since by our assumption in $\mathscr{E}^\alpha$ the neutral action filters are used and for all $i \in \mathcal{A}$

$$filter_i^N\left(X_1^{sup}, \ldots, X_n^{sup}, X_\epsilon\right) = X_i^{sup}, \tag{3.355}$$

it follows that for all $i \in \mathcal{A}$

$$X_i^{sup} = X_i. \tag{3.356}$$

From (3.353), (3.354) and (3.356), however, we get that

$$filter_\epsilon^\alpha\left(h, filter_\epsilon^\alpha\left(h, X_\epsilon^{sup}, X_1, \ldots, X_n\right), X_1, \ldots, X_n\right) \subset filter_\epsilon^\alpha\left(h, X_\epsilon^{sup}, X_1, \ldots, X_n\right), \tag{3.357}$$

which leads to a contradiction, as $filter_\epsilon^\alpha$ was assumed to be idempotent for the domain PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$. □

3.3.4 Downward Closed Safety Properties

An important category of safety properties are those that are closed with respect to subsets of their elements (downward closed).

Definition 3.3.49. We say a safety property S is *downward closed* iff

$$(\forall h \in \mathscr{G})(\forall X \in S(h))(\forall X' \subseteq X)\ X' \in S(h) \tag{3.358}$$

Lemma 3.3.50. *For every safety property* S *that is not downward closed, there does not necessarily exist some* S' *such that* $(\forall h \in \mathscr{G})\ S'(h) \subseteq S(h)$ *and* S' *is downward closed.*

Proof. An easy counterexample is a safety property for which the following holds

$$(\forall h \in \mathscr{G})\ \varnothing \notin S(h). \tag{3.359}$$

The only downward closed safety property S' that is a subset of S would be

$$(\forall h \in \mathscr{G})\ S'(h) = \varnothing, \tag{3.360}$$

which however conflicts with the Definition 3.3.12 of operational safety properties. □

Lemma 3.3.51. *Given a safety property S that is downward closed, an extension $\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^\alpha, \Psi^\alpha)$ adhering to S, an extension $\mathscr{E}^\beta = (PP^\beta, IS^\beta, \tau^\beta, \Psi^\beta)$ compatible w.r.t. composition $\beta \circ \alpha$ respectively intersection $\alpha + \beta$ respectively k-intersection $k \cdot (\alpha + \beta)$ (for some $k \geq 1$) respectively fixpoint intersection $\alpha * \beta$ with $\mathscr{E}^\alpha$, if all action filter functions $filter_i^\alpha$ are monotonic for PD_ϵ^{t-coh}, $2^{\overline{GActions}_1}$, ..., $2^{\overline{GActions}_n}$, then any run (corresponding with the extensions' compatibility) $r \in R^{\chi^{\beta\circ\alpha}}$ respectively $r' \in R^{\chi^{\alpha+\beta}}$ respectively $r'' \in R^{\chi^{k\cdot(\alpha+\beta)}}$ respectively $r''' \in R^{\chi^{\alpha*\beta}}$ of some agent contexts $\chi^{\beta\circ\alpha} \in \mathscr{E}^{\beta\circ\alpha}$ respectively $\chi^{\alpha+\beta} \in \mathscr{E}^{\alpha+\beta}$ respectively $\chi^{k\cdot(\alpha+\beta)} \in \mathscr{E}^{k\cdot(\alpha+\beta)}$ respectively $\chi^{\alpha*\beta} \in \mathscr{E}^{\alpha*\beta}$ always satisfies S. In other words, if S is a safety property that is downward closed and $\mathscr{E}^\alpha$ is an extension adhering to S, whose action filter functions are all monotonic for PD_ϵ^{t-coh}, $2^{\overline{GActions}_1}$, ..., $2^{\overline{GActions}_n}$, then for $k \geq 1$ the extensions $\mathscr{E}^{\beta\circ\alpha}$ respectively $\mathscr{E}^{\alpha+\beta}$ respectively $\mathscr{E}^{k\cdot(\alpha+\beta)}$ respectively $\mathscr{E}^{\alpha*\beta}$ also adhere to S for any $\mathscr{E}^\beta$ compatible with $\mathscr{E}^\alpha$ w.r.t. $\beta\circ\alpha$ respectively $\alpha+\beta$ respectively $k\cdot(\alpha+\beta)$ respectively $\alpha * \beta$.*

Proof. If the two extensions are compatible w.r.t. $\beta \circ \alpha$, $\alpha + \beta$, $k \cdot (\alpha + \beta)$ or $\alpha * \beta$, the combined extensions (by Definition 3.2.1) are

$$\begin{aligned}
\mathscr{E}^{\beta\circ\alpha} &= (PP^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{\beta\circ\alpha}, \Psi^\alpha \cap \Psi^\beta)\\
\mathscr{E}^{\alpha+\beta} &= (PP^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{\alpha+\beta}, \Psi^\alpha \cap \Psi^\beta)\\
\mathscr{E}^{k\cdot(\alpha+\beta)} &= (PP^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{k\cdot(\alpha+\beta)}, \Psi^\alpha \cap \Psi^\beta)\\
\mathscr{E}^{\alpha*\beta} &= (PP^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{\alpha*\beta}, \Psi^\alpha \cap \Psi^\beta).
\end{aligned}$$

Since (first three points by semantics of set intersection, fourth one by assumption)

- $(P_\epsilon, P) \in PP^\alpha \cap PP^\beta \Rightarrow (P_\epsilon, P) \in PP^\alpha$ and
- $\mathscr{G}(0) \in IS^\alpha \cap IS^\beta \Rightarrow \mathscr{G}(0) \in IS^\alpha$ and
- $r \in \Psi^\alpha \cap \Psi^\beta \Rightarrow r \in \Psi^\alpha$ and
- $\mathscr{E}^\alpha$ adheres to S,

it remains to investigate whether the transition templates $\tau^{\beta\circ\alpha}$, $\tau^{\alpha+\beta}$, $\tau^{k\cdot(\alpha+\beta)}$, $\tau^{\alpha*\beta}$ might contradict our claim.

By Lemma 3.1.70, we get that

$$\begin{aligned}
filter_\epsilon^{\beta\circ\alpha} &\subseteq filter_\epsilon^\alpha\\
filter_\epsilon^{\alpha+\beta} &\subseteq filter_\epsilon^\alpha\\
filter_\epsilon^{k\cdot(\alpha+\beta)} &\subseteq filter_\epsilon^\alpha\\
filter_\epsilon^{\alpha*\beta} &\subseteq filter_\epsilon^\alpha.
\end{aligned} \tag{3.361}$$

For some $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \in 2^{\overline{GActions_1}}$, ..., $X_n \in 2^{\overline{GActions_n}}$, by the assumption of monotonicity of the filter functions $filter_i^\alpha$ (for all $i \in \mathcal{A}$) in the domain PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ and also by Lemma 3.1.70, we get that

$$\begin{aligned}
&filter_i^{\beta\circ\alpha}\left(X_1,\ldots,X_n,filter_\epsilon^{\beta\circ\alpha}\left(h,X_\epsilon,X_1,\ldots,X_n\right)\right)\subseteq\\
&filter_i^{\alpha}\left(X_1,\ldots,X_n,filter_\epsilon^{\beta\circ\alpha}\left(h,X_\epsilon,X_1,\ldots,X_n\right)\right)\subseteq\\
&filter_i^{\alpha}\left(X_1,\ldots,X_n,filter_\epsilon^{\alpha}\left(h,X_\epsilon,X_1,\ldots,X_n\right)\right)\\
&filter_i^{\alpha+\beta}\left(X_1,\ldots,X_n,filter_\epsilon^{\alpha+\beta}\left(h,X_\epsilon,X_1,\ldots,X_n\right)\right)\subseteq\\
&filter_i^{\alpha}\left(X_1,\ldots,X_n,filter_\epsilon^{\alpha+\beta}\left(h,X_\epsilon,X_1,\ldots,X_n\right)\right)\subseteq\\
&filter_i^{\alpha}\left(X_1,\ldots,X_n,filter_\epsilon^{\alpha}\left(h,X_\epsilon,X_1,\ldots,X_n\right)\right)\\
&filter_i^{k\cdot(\alpha+\beta)}\left(X_1,\ldots,X_n,filter_\epsilon^{k\cdot(\alpha+\beta)}\left(h,X_\epsilon,X_1,\ldots,X_n\right)\right)\subseteq\\
&filter_i^{\alpha}\left(X_1,\ldots,X_n,filter_\epsilon^{k\cdot(\alpha+\beta)}\left(h,X_\epsilon,X_1,\ldots,X_n\right)\right)\subseteq\\
&filter_i^{\alpha}\left(X_1,\ldots,X_n,filter_\epsilon^{\alpha}\left(h,X_\epsilon,X_1,\ldots,X_n\right)\right)\\
&filter_i^{\alpha*\beta}\left(X_1,\ldots,X_n,filter_\epsilon^{\alpha*\beta}\left(h,X_\epsilon,X_1,\ldots,X_n\right)\right)\subseteq\\
&filter_i^{\alpha}\left(X_1,\ldots,X_n,filter_\epsilon^{\alpha*\beta}\left(h,X_\epsilon,X_1,\ldots,X_n\right)\right)\subseteq\\
&filter_i^{\alpha}\left(X_1,\ldots,X_n,filter_\epsilon^{\alpha}\left(h,X_\epsilon,X_1,\ldots,X_n\right)\right).
\end{aligned}$$

Hence since S is assumed to be downward closed we conclude that for any timestamp $t \in \mathbb{N}$, agent contexts $\chi^{\beta\circ\alpha} \in \mathscr{E}^{\beta\circ\alpha}$, $\chi^{\alpha+\beta} \in \mathscr{E}^{\alpha+\beta}$, $\chi^{k\cdot(\alpha+\beta)} \in \mathscr{E}^{k\cdot(\alpha+\beta)}$, $\chi^{\alpha*\beta} \in \mathscr{E}^{\alpha*\beta}$ and runs $r \in R^{\chi^{\beta\circ\alpha}}$, $r' \in R^{\chi^{\alpha+\beta}}$, $r'' \in R^{\chi^{k\cdot(\alpha+\beta)}}$, $r''' \in R^{\chi^{\alpha*\beta}}$

$$\begin{aligned}
\beta^t\left(r\right) &\in S(r(t))\\
\beta^t\left(r'\right) &\in S(r'(t))\\
\beta^t\left(r''\right) &\in S(r''(t))\\
\beta^t\left(r'''\right) &\in S(r'''(t)).
\end{aligned} \tag{3.362}$$

□

Lemma 3.3.52. *Given a safety property S that is downward closed, an extension $\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^\alpha, \Psi^\alpha)$ adhering to S, an extension $\mathscr{E}^\beta = (PP^\beta, IS^\beta, \tau^\beta, \Psi^\beta)$ compatible w.r.t. $\alpha \circ \beta$ with $\mathscr{E}^\alpha$, if for PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ $filter_\epsilon^\alpha$ is simply monotonic or $filter_\epsilon^\beta$ is stricter than $filter_\epsilon^\alpha$ and if for PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ the action filters $filter_i^\alpha$ are monotonic (for all $i \in \mathcal{A}$), then any run $r \in R^{\chi^{\alpha\circ\beta}}$ of some agent context $\chi^{\alpha\circ\beta} \in \mathscr{E}^{\alpha\circ\beta}$ satisfies S.*

Proof. The reasoning is analogous to Lemma 3.3.51, with the only difference that instead of Lemma 3.1.70 Lemma 3.1.33 has to be used to conclude that for PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ and all $i \in \mathcal{A}$

$$filter_\epsilon^{\alpha\circ\beta} \subseteq filter_\epsilon^\alpha \tag{3.363}$$

□

Lemma 3.3.53. *Any extension $\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^\alpha, \Psi^\alpha)$, where in τ^α at least one Byzantine action filter for some $i \in \mathcal{A}$ is used if for some $(P_\epsilon, P) \in PP^\alpha$ there exists some $h \in \mathscr{G}$ s.t. there exists some $X_i \in P_i(h_i)$ with $\varnothing \subset X_i$ and there exist sets $X_\epsilon^{nogo}, X_\epsilon^{go} \in P_\epsilon(|h|)$, where $X_\epsilon^{nogo} \subset X_\epsilon^{go}$ and s.t. $go(i) \in X_\epsilon^{go}$, but $go(i) \notin X_\epsilon^{nogo}$, then S^α is not downward closed.*

Proof. We conduct this proof by counterexample. Suppose for some $(P_\epsilon, P) \in PP^\alpha$ there exists some $h \in \mathscr{G}$ s.t. there exists some $X_i \in P_i(h_i)$ with $\varnothing \subset X_i$ and there exist sets $X_\epsilon^{nogo} \subset X_\epsilon^{go} \in P_\epsilon(|h|)$ s.t. $go(i) \in X_\epsilon^{go}, go(i) \notin X_\epsilon^{nogo}$. For any $X_1 \subseteq \overline{GActions}_1$, ..., $X_{i-1} \subseteq \overline{GActions}_{i-1}$, $X_{i+1} \subseteq \overline{GActions}_{i+1}$, ..., $X_n \subseteq \overline{GActions}_n$, by $|h|$-coherence of X_ϵ^{go} and X_ϵ^{nogo}, it follows that

$$filter_i^B\left(X_1, \ldots, X_n, X_\epsilon^{go}\right) = X_i, \tag{3.364}$$

but

$$filter_i^B\left(X_1, \ldots, X_n, X_\epsilon^{nogo}\right) = \varnothing. \tag{3.365}$$

Hence the configuration consisting of X_ϵ^{nogo}, X_i (a subset of the configuration that instead consists of X_ϵ^{go}) cannot be achieved with the Byzantine action filter. □

3.3.5 Composability of Implementation Classes

Definition 3.3.54 (Composability Relation)**.** We define a composability relation $\mathscr{Q}$ between implementation classes:

$$\mathscr{Q} \subseteq \mathscr{I} \times \mathscr{I}. \tag{3.366}$$

For two implementation classes $IC^\alpha, IC^\beta \in \mathscr{I}$ we say IC^α is *composable* with IC^β and write either

$$IC^\alpha \; \mathscr{Q} \; IC^\beta \quad \text{or} \quad (IC^\alpha, IC^\beta) \in \mathscr{Q}$$

iff for all extensions $\mathscr{E}^\alpha \in IC^\alpha$ and $\mathscr{E}^\beta \in IC^\beta$, s.t. $\mathscr{E}^\alpha$ and $\mathscr{E}^\beta$ are compatible w.r.t. $\star$, where $\star$ represents composition $(\alpha \circ \beta)$, reverse composition $(\beta \circ \alpha)$, k-intersection $(k \cdot (\alpha + \beta)$ for $k \geq 1)$ or fixpoint intersection $(\alpha * \beta)$, the extension $\mathscr{E}^{\alpha \star \beta}$ adheres to S^β.

Definition 3.3.55 (Downward Closed Safety Property Composability Relation)**.** We define the DCSP composability relation $\widetilde{\mathscr{Q}}$ between implementation classes:

$$\widetilde{\mathscr{Q}} \subseteq \mathscr{I} \times \mathscr{I}. \tag{3.367}$$

For two implementation classes $IC^\alpha, IC^\beta \in \mathscr{I}$ we say IC^α is *DCSP composable* with IC^β and write either

$$IC^\alpha \; \widetilde{\mathscr{Q}} \; IC^\beta \quad \text{or} \quad (IC^\alpha, IC^\beta) \in \widetilde{\mathscr{Q}}$$

iff for all extensions $\mathscr{E}^\alpha \in IC^\alpha$ and $\mathscr{E}^\beta \in IC^\beta$, s.t. $\mathscr{E}^\alpha$ and $\mathscr{E}^\beta$ are compatible w.r.t. $\star_{DCSP}$, where $\star_{DCSP}$ represents composition $(\alpha \circ \beta)$, k-intersection $(k \cdot (\alpha + \beta)$ for $k \geq 1)$ or fixpoint intersection $(\alpha * \beta)$, the extension $\mathscr{E}^{\alpha \star_{DCSP} \beta}$ adheres to S^β.

Definition 3.3.56 (Fixpoint Composability Relation)**.** Further we define the fixpoint composability relation $\mathscr{F}$ between implementation classes:

$$\mathscr{F} \subseteq \mathscr{I} \times \mathscr{I}. \tag{3.368}$$

For two implementation classes $IC^\alpha, IC^\beta \in \mathscr{I}$ we say IC^α is *fixpoint composable* with IC^β and write either

$$IC^\alpha \ \mathscr{F} \ IC^\beta \quad \text{or} \quad (IC^\alpha, IC^\beta) \in \mathscr{F}$$

iff for all extensions $\mathscr{E}^\alpha \in IC^\alpha$ and $\mathscr{E}^\beta \in IC^\beta$, s.t. $\mathscr{E}^\alpha$ and $\mathscr{E}^\beta$ are compatible w.r.t. $\star_f$, where $\star_f$ represents reversed composition $(\beta \circ \alpha)$ or fixpoint intersection $(\alpha * \beta)$, the extension $\mathscr{E}^{\alpha \star_f \beta}$ adheres to S^β.

Definition 3.3.57 (Readily Composable)**.** For two implementation classes $IC^\alpha, IC^\beta \in \mathscr{I}$ we say IC^α is *readily composable* with IC^β or IC^β is *readily composable* with IC^α iff

$$(IC^\alpha, IC^\beta) \in \mathscr{Q} \ \wedge \ (IC^\beta, IC^\alpha) \in \mathscr{Q},$$

meaning the composability relation is symmetric.

Definition 3.3.58 (Readily DCSP Composable)**.** Similarly for two implementation classes $IC^\alpha, IC^\beta \in \mathscr{I}$ we say IC^α is *readily DCSP composable* with IC^β or IC^β is *readily DCSP composable* with IC^α iff

$$(IC^\alpha, IC^\beta) \in \widetilde{\mathscr{Q}} \ \wedge \ (IC^\beta, IC^\alpha) \in \widetilde{\mathscr{Q}},$$

meaning the DCSP composability relation is symmetric.

Definition 3.3.59 (Readily Fixpoint Composable)**.** Similarly for two implementation classes $IC^\alpha, IC^\beta \in \mathscr{I}$ we say IC^α is *readily fixpoint composable* with IC^β or IC^β is *readily fixpoint composable* with IC^α iff

$$(IC^\alpha, IC^\beta) \in \mathscr{F} \ \wedge \ (IC^\beta, IC^\alpha) \in \mathscr{F},$$

meaning the fixpoint composability relation is symmetric.

Lemma 3.3.60. *Composability implies DCSP composability and fixpoint composability: For some implementation classes* $IC^\alpha, IC^\beta \in \mathscr{I}$

$$(IC^\alpha, IC^\beta) \in \mathscr{Q} \quad \Longrightarrow \quad (IC^\alpha, IC^\beta) \in \widetilde{\mathscr{Q}} \ \wedge \ (IC^\alpha, IC^\beta) \in \mathscr{F} \tag{3.369}$$

Proof. Follows directly from the Definitions 3.3.54, 3.3.55 and 3.3.56. □

Note that since some of our introduced implementation classes are subsets of other classes, any finding for a super set class is obviously valid for the subset class as well. Hence in the following results we only ever denote the biggest classes for which the results hold.

Lemma 3.3.61. *The implementation class* **Adm** *is readily composable with all implementation classes. Formally for all implementation classes* $IC \in \mathscr{I}$*, it holds that*

$$(\mathbf{Adm}, IC) \in \mathscr{Q} \tag{3.370}$$
$$(IC, \mathbf{Adm}) \in \mathscr{Q}. \tag{3.371}$$

Proof. Regarding equation (3.370): For some $\mathscr{E}^\alpha \in \mathbf{Adm}$ it holds that

$$\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}, IS^\alpha, \tau^{N,N}, \Psi^\alpha). \tag{3.372}$$

For any arbitrary extension

$$\mathscr{E}^\beta = (PP^\beta, IS^\beta, \tau^\beta, \Psi^\beta), \tag{3.373}$$

which is compatible (w.r.t. some composition, k intersection for $k \geq 1$ or fixpoint intersection) with $\mathscr{E}^\alpha$, the combined extensions are (for some $k \geq 1$)

$$\mathscr{E}^{\alpha\circ\beta} = (\mathscr{C}_\epsilon \times \mathscr{C} \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{N,N\circ\beta}, \Psi^\alpha \cap \Psi^\beta), \tag{3.374}$$

$$\mathscr{E}^{\beta\circ\alpha} = (\mathscr{C}_\epsilon \times \mathscr{C} \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{\beta\circ N,N}, \Psi^\alpha \cap \Psi^\beta), \tag{3.375}$$

$$\mathscr{E}^{k\cdot(\alpha+\beta)} = (\mathscr{C}_\epsilon \times \mathscr{C} \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{k\cdot(N,N+\beta)}, \Psi^\alpha \cap \Psi^\beta), \tag{3.376}$$

$$\mathscr{E}^{\alpha*\beta} = (\mathscr{C}_\epsilon \times \mathscr{C} \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{N,N*\beta}, \Psi^\alpha \cap \Psi^\beta). \tag{3.377}$$

Since by Lemmas 3.1.24, 3.1.38, 3.1.49 and 3.1.61 the neutral filters are the neutral element for idempotent filters w.r.t. filter composition, filter intersection, k-filter intersection and fixpoint filter intersection, and the filter functions of any implementation class are idempotent in the domain PD_ϵ^{t-coh}, $2^{\overline{GActions}_1}$, ..., $2^{\overline{GActions}_n}$ the transition templates simplify to

$$\tau^{N,N\circ\beta} = \tau^{\beta\circ N,N} = \tau^{k\cdot(N,N+\beta)} = \tau^{N,N*\beta} = \tau^\beta, \tag{3.378}$$

from which (3.370) follows (by semantics of set intersection) as

$$\mathscr{E}^{\alpha\circ\beta} = \mathscr{E}^{\beta\circ\alpha} = \mathscr{E}^{k\cdot(\alpha+\beta)} = \mathscr{E}^{\alpha*\beta} = (\mathscr{C}_\epsilon \times \mathscr{C} \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^\beta, \Psi^\alpha \cap \Psi^\beta). \tag{3.379}$$

Considering (3.371) since the implementation class **Adm** only restricts the admissibility condition the statement follows from Lemma 3.3.10. □

Lemma 3.3.62. *The implementation class* **JP** *is composable with all implementation classes. Formally, for all implementation classes $IC \in \mathscr{I}$ it holds that*

$$(\mathbf{JP}, IC) \in \mathscr{Q} \tag{3.380}$$

Proof. For some $\mathscr{E}^\alpha \in \mathbf{JP}$ it holds that

$$\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha, IS^\alpha, \tau^{N,N}, \Psi^\alpha). \tag{3.381}$$

For any arbitrary extension

$$\mathscr{E}^\beta = (PP^\beta, IS^\beta, \tau^\beta, \Psi^\beta), \tag{3.382}$$

which is compatible (w.r.t. some composition, k-intersection for $k \geq 1$ or fixpoint intersection) with $\mathscr{E}^\alpha$ the combined extensions are (for some $k \geq 1$)

$$\mathscr{E}^{\alpha\circ\beta} = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{N,N\circ\beta}, \Psi^\alpha \cap \Psi^\beta), \tag{3.383}$$

$$\mathscr{E}^{\beta\circ\alpha} = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{\beta\circ N,N}, \Psi^\alpha \cap \Psi^\beta), \tag{3.384}$$

$$\mathscr{E}^{k\cdot(\alpha+\beta)} = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{k\cdot(N,N+\beta)}, \Psi^\alpha \cap \Psi^\beta) \text{ or} \tag{3.385}$$

$$\mathscr{E}^{\alpha*\beta} = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{N,N*\beta}, \Psi^\alpha \cap \Psi^\beta) \tag{3.386}$$

Since by Lemmas 3.1.24, 3.1.38, 3.1.49 and 3.1.61 the neutral filters are the neutral element for idempotent filters w.r.t. filter composition, filter intersection, k-filter intersection and fixpoint filter intersection, and the filter functions of any implementation class are idempotent in the domain PD_{ϵ}^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$ the transition templates simplify to

$$\tau^{N,N\circ\beta} = \tau^{\beta\circ N,N} = \tau^{k\cdot(N,N+\beta)} = \tau^{N,N*\beta} = \tau^{\beta}, \tag{3.387}$$

from which (3.380) follows (by semantics of set intersection) as

$$\mathscr{E}^{\alpha\circ\beta} = \mathscr{E}^{\beta\circ\alpha} = \mathscr{E}^{k\cdot(\alpha+\beta)} = \mathscr{E}^{\alpha*\beta} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha} \cap PP^{\beta}, IS^{\alpha} \cap IS^{\beta}, \tau^{\beta}, \Psi^{\alpha} \cap \Psi^{\beta}). \tag{3.388}$$

□

Lemma 3.3.63. *The implementation class* **EnvJP** *is composable with every implementation class. Formally, for every* $IC \in \mathscr{I}$ *it holds that*

$$(\mathbf{EnvJP}, IC) \in \mathscr{Q}. \tag{3.389}$$

Proof. For some arbitrary $\mathscr{E}^{\alpha} \in \mathbf{EnvJP}$ it holds that

$$\mathscr{E}^{\alpha} = (PP^{\alpha}, IS^{\alpha}, \tau^{N,N}, \Psi^{\alpha}) \tag{3.390}$$

and some arbitrary $\mathscr{E}^{\beta} \in IC$ (for some $IC \in \mathscr{I}$) compatible (w.r.t. some composition, k-intersection for $k \geq 1$ or fixpoint intersection) with $\mathscr{E}^{\alpha}$ it holds that

$$\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta}, \Psi^{\beta}). \tag{3.391}$$

By Definition 3.2.1 of safety properties and Lemmas 3.1.24, 3.1.38, 3.1.49 and 3.1.61 we get (for some $k \geq 1$) that

$$\mathscr{E}^{\alpha\circ\beta} = \mathscr{E}^{\beta\circ\alpha} = \mathscr{E}^{k\cdot(\alpha+\beta)} = \mathscr{E}^{\alpha*\beta} = (PP^{\alpha} \cap PP^{\beta}, IS^{\alpha} \cap IS^{\beta}, \tau^{\beta}, \Psi^{\alpha} \cap \Psi^{\beta}), \tag{3.392}$$

from which (3.389) follows. □

Lemma 3.3.64. *The implementation classes* **EvFEnvJP** *and* **EvFJP** *are composable with* **JP**. *Formally,*

$$(\mathbf{EvFEnvJP}, \mathbf{JP}) \in \mathscr{Q} \tag{3.393}$$

$$(\mathbf{EvFJP}, \mathbf{JP}) \in \mathscr{Q}. \tag{3.394}$$

Proof. Regarding (3.393) for some arbitrary $\mathscr{E}^{\alpha} \in \mathbf{EvFEnvJP}$ it holds that

$$\mathscr{E}^{\alpha} = (PP^{\alpha}, IS^{\alpha}, \tau^{\alpha,N}, \Psi^{\alpha}), \tag{3.395}$$

where in $\tau^{\alpha,N}$ the event filter $filter_{\epsilon}^{\alpha}$ and the neutral action filters $filter_i^N$ (for all $i \in \mathcal{A}$) are used. For some $\mathscr{E}^{\beta} \in \mathbf{JP}$ compatible (w.r.t. some composition, k-intersection for $k \geq 1$ or fixpoint intersection) with $\mathscr{E}^{\alpha}$ it holds that

$$\mathscr{E}^{\beta} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\beta}, IS^{\beta}, \tau^{N,N}, \Psi^{\beta}). \tag{3.396}$$

By Definition 3.2.1 of extension combination and Lemmas 3.1.24, 3.1.38, 3.1.49 and 3.1.61 we get (for some $k \geq 1$) that

$$\mathscr{E}^{\alpha\circ\beta} = \mathscr{E}^{\beta\circ\alpha} = \mathscr{E}^{k\cdot(\alpha+\beta)} = \mathscr{E}^{\alpha*\beta} = (PP^{\alpha} \cap \mathscr{C}_{\epsilon} \times \mathscr{C}^{\beta}, IS^{\alpha} \cap IS^{\beta}, \tau^{\alpha,N}, \Psi^{\alpha} \cap \Psi^{\beta}). \quad (3.397)$$

By Corollary 3.3.43 the extension (3.397) satisfies S^{α}, from which (3.393) follows.

Regarding (3.394) for some arbitrary $\mathscr{E}^{\alpha} \in \mathbf{EvFJP}$ it holds that

$$\mathscr{E}^{\alpha} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha,N}, \Psi^{\alpha}), \quad (3.398)$$

where in $\tau^{\alpha,N}$ some event filter function $filter^{\alpha}_{\epsilon}$ and the neutral action filters are used and some $\mathscr{E}^{\beta} \in \mathbf{JP}$ compatible (w.r.t. some composition, k-intersection for $k \geq 1$ or fixpoint intersection) with $\mathscr{E}^{\alpha}$ it holds that

$$\mathscr{E}^{\beta} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\beta}, IS^{\beta}, \tau^{N,N}, \Psi^{\beta}). \quad (3.399)$$

By Definition 3.2.1 of extension combination and Lemmas 3.1.24, 3.1.38, 3.1.49 and 3.1.61 we get (for some $k \geq 1$) that

$$\mathscr{E}^{\alpha\circ\beta} = \mathscr{E}^{\beta\circ\alpha} = \mathscr{E}^{k\cdot(\alpha+\beta)} = \mathscr{E}^{\alpha*\beta} = (\mathscr{C}_{\epsilon} \times (\mathscr{C}^{\alpha} \cap \mathscr{C}^{\beta}), IS^{\alpha} \cap IS^{\beta}, \tau^{\alpha,N}, \Psi^{\alpha} \cap \Psi^{\beta}). \quad (3.400)$$

Similar to our previous case by Corollary 3.3.43 the extension (3.400) satisfies S^{α}, from which (3.394) follows. □

Lemma 3.3.65. *The implementation class* **EvFJP** *is fixpoint composable with* **EvFJP** *(itself) and* **EvFJP** $-$ **AFB**. *Formally,*

$$(\mathbf{EvFJP}, \mathbf{EvFJP}) \in \mathscr{F} \quad (3.401)$$
$$(\mathbf{EvFJP}, \mathbf{EvFJP} - \mathbf{AFB}) \in \mathscr{F}. \quad (3.402)$$

Proof. Regarding (3.401) for two compatible (w.r.t. composition $\beta \circ \alpha$ or fixpoint intersection) extensions $\mathscr{E}^{\alpha}, \mathscr{E}^{\beta} \in \mathbf{EvFJP}$ it holds that

$$\mathscr{E}^{\alpha} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha,N}, \Psi^{\alpha}) \quad (3.403)$$
$$\mathscr{E}^{\beta} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\beta}, IS^{\beta}, \tau^{\beta,N}, \Psi^{\beta}), \quad (3.404)$$

where in $\tau^{\alpha,N}$ the event filter $filter^{\alpha}_{\epsilon}$ and the neutral action filters and in $\tau^{\beta,N}$ the event filter $filter^{\beta}_{\epsilon}$ and the neutral action filters are used. The statement now directly follows from Corollaries 3.3.35 and 3.3.43.

Regarding (3.402) let

$$\mathscr{E}^{\alpha} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{\alpha,N}, \Psi^{\alpha}) \quad (3.405)$$
$$\mathscr{E}^{\beta} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\beta}, IS^{\beta}, \tau^{\beta,B}, \Psi^{\beta}) \quad (3.406)$$

be some arbitrary extension $\mathscr{E}^{\alpha} \in \mathbf{EvFJP}$ and $\mathscr{E}^{\beta} \in \mathbf{EvFJP} - \mathbf{AFB}$ an extension compatible (w.r.t. composition $\beta \circ \alpha$ or fixpoint intersection) with $\mathscr{E}^{\alpha}$, where in $\tau^{\alpha,N}$ the event filter $filter^{\alpha}_{\epsilon}$ and the neutral action filters and in $\tau^{\beta,B}$ the event filter $filter^{\beta}_{\epsilon}$ and the Byzantine action filters $filter^{B}_{i}$ (for all $i \in \mathcal{A}$) are used. Since the Byzantine filters are monotonic by Lemma 3.1.17, the statement now directly follows from Corollaries 3.3.37 and 3.3.45. □

Lemma 3.3.66. *The implementation class* **EvFEnvJP** *is fixpoint composable with* **EvFJP** *and* **EvFJP − AFB**. *Formally,*

$$(\mathbf{EvFEnvJP}, \mathbf{EvFJP}) \in \mathscr{F} \tag{3.407}$$
$$(\mathbf{EvFEnvJP}, \mathbf{EvFJP - AFB}) \in \mathscr{F} \tag{3.408}$$

Proof. For three arbitrary extensions $\mathscr{E}^\alpha \in \mathbf{EvFEnvJP}$, $\mathscr{E}^\beta \in \mathbf{EvFJP}$, $\mathscr{E}^\gamma \in \mathbf{EvFJP - AFB}$ it holds that

$$\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^{\alpha,N}, \Psi^\alpha) \tag{3.409}$$
$$\mathscr{E}^\beta = (\mathscr{C}_\epsilon \times \mathscr{C}^\beta, IS^\beta, \tau^{\beta,N}, \Psi^\beta) \tag{3.410}$$
$$\mathscr{E}^\gamma = (\mathscr{C}_\epsilon \times \mathscr{C}^\gamma, IS^\gamma, \tau^{\gamma,B}, \Psi^\gamma), \tag{3.411}$$

where in $\tau^{\alpha,N}$ the event filter $filter_\epsilon^\alpha$ and the neutral action filters, in $\tau^{\beta,N}$ the event filter $filter_\epsilon^\beta$ and the neutral actions filters and in $\tau^{\gamma,B}$ the event filter $filter_\epsilon^\gamma$ and the Byzantine action filters $filter_i^B$ (for all $i \in \mathcal{A}$) are used. (3.407) follows directly from Corollaries 3.3.35 and 3.3.43. Since the Byzantine filter functions are monotonic by Lemma 3.1.17 the statement (3.408) follows from Corollaries 3.3.37 and 3.3.45. □

Lemma 3.3.67. *The implementation classes* **EvFJP − AFB** *and* **EvFEnvJP − AFB** *are fixpoint composable with* **EvFJP − AFB**. *Formally*

$$(\mathbf{EvFJP - AFB}, \mathbf{EvFJP - AFB}) \in \mathscr{F} \tag{3.412}$$
$$(\mathbf{EvFEnvJP - AFB}, \mathbf{EvFJP - AFB}) \in \mathscr{F} \tag{3.413}$$

Proof. Let $\mathscr{E}^\alpha, \mathscr{E}^\beta \in \mathbf{EvFJP - AFB}$ and $\mathscr{E}^\gamma \in \mathbf{EvFEnvJP - AFB}$ be three extensions. It holds that

$$\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha, IS^\alpha, \tau^{\alpha,B}, \Psi^\alpha) \tag{3.414}$$
$$\mathscr{E}^\beta = (\mathscr{C}_\epsilon \times \mathscr{C}^\beta, IS^\beta, \tau^{\beta,B}, \Psi^\beta) \tag{3.415}$$
$$\mathscr{E}^\gamma = (PP^\gamma, IS^\beta, \tau^{\gamma,B}, \Psi^\gamma), \tag{3.416}$$

where in $\tau^{\alpha,B}$ the event filter $filter_\epsilon^\alpha$ and the Byzantine action filters $filter_i^B$ (for all $i \in \mathcal{A}$), in $\tau^{\beta,B}$ the event filter $filter_\epsilon^\beta$ and the Byzantine action filters $filter_i^B$ (for all $i \in \mathcal{A}$) and in $\tau^{\gamma,B}$ the event filter $filter_\epsilon^\gamma$ and also the Byzantine action filters $filter_i^B$ (for all $i \in \mathcal{A}$) are used. Since the Byzantine filters are idempotent by Lemma 3.1.22 the statements (3.412), (3.413) now directly follow from Corollaries 3.3.35 and 3.3.43. □

Lemma 3.3.68. *Every implementation class except* **Others** *is composable with* **JP − AFB**. *Formally, for every* $IC \in \mathscr{I} \setminus \{\mathbf{Others}\}$, *it holds that*

$$(IC, \mathbf{JP - AFB}) \in \mathscr{2}. \tag{3.417}$$

Proof. For some arbitrary $\mathscr{E}^\alpha \in \mathbf{JP - AFB}$, it holds that

$$\mathscr{E}^\alpha = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha, IS^\alpha, \tau^{N,B}, \Psi^\alpha). \tag{3.418}$$

For some arbitrary $\mathscr{E}^\beta \in IC$ (for some $IC \in \mathscr{I} \setminus \{\mathbf{Others}\}$) compatible (w.r.t. some composition, k-intersection for $k \geq 1$ or fixpoint intersection) with $\mathscr{E}^\alpha$ there are two possibilities:

1. $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta,N}, \Psi^{\beta})$ or
2. $\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta,B}, \Psi^{\beta})$

However, either way by Definition 3.2.1 of extension combination and Lemmas 3.1.24, 3.1.38, 3.1.49 and 3.1.61, since the Byzantine action filters are idempotent by Lemma 3.1.22, we get (for some $k \geq 1$) that

$$\mathscr{E}^{\alpha\circ\beta} = \mathscr{E}^{\beta\circ\alpha} = \mathscr{E}^{k\cdot(\alpha+\beta)} = \mathscr{E}^{\alpha*\beta} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha} \cap PP^{\beta}, IS^{\alpha} \cap IS^{\beta}, \tau^{\beta,B}, \Psi^{\alpha} \cap \Psi^{\beta}). \quad (3.419)$$

from which (3.417) follows. □

Lemma 3.3.69. *The implementation class* $\mathbf{JP-AFB}$ *is composable with* $\mathbf{EnvJP-AFB}$, $\mathbf{EvFJP-AFB}$ *and* $\mathbf{EvFEnvJP-AFB}$, *or formally*

$$(\mathbf{JP-AFB}, \mathbf{EnvJP-AFB}) \in \mathscr{Q} \quad (3.420)$$
$$(\mathbf{JP-AFB}, \mathbf{EvFJP-AFB}) \in \mathscr{Q} \quad (3.421)$$
$$(\mathbf{JP-AFB}, \mathbf{EvFEnvJP-AFB}) \in \mathscr{Q} \quad (3.422)$$

Proof. Regarding (3.421) and (3.422) let the following be two arbitrary compatible (w.r.t. some composition, k-intersection for $k \geq 1$ or fixpoint intersection) extensions $\mathscr{E}^{\alpha} \in \mathbf{JP-AFB}$ and $\mathscr{E}^{\beta} \in \mathbf{EvFJP-AFB}$ or $\mathscr{E}^{\beta} \in \mathbf{EvFEnvJP-AFB}$. In either case we get

$$\mathscr{E}^{\alpha} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{N,B}, \Psi^{\alpha}) \quad (3.423)$$
$$\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{\beta,B}, \Psi^{\beta}), \quad (3.424)$$

where $\mathscr{C}^{\alpha} \subset \mathscr{C}$, $PP^{\beta} \subseteq \mathscr{C}_{\epsilon} \times \mathscr{C}$, $IS^{\alpha}, IS^{\beta} \subseteq 2^{\mathscr{G}(0)}$, $\Psi^{\alpha}, \Psi^{\beta} \subseteq R$, and for the transition template $\tau^{N,B}$ the neutral event filter $filter_{\epsilon}^{N}$ and the Byzantine action filters $filter_{i}^{B}$ (for all $i \in \mathcal{A}$), and for the transition template $\tau^{\beta,B}$ the event filter $filter_{\epsilon}^{\beta}$ and also the Byzantine action filters $filter_{i}^{B}$ (for all $i \in \mathcal{A}$) are used. By Definition 3.2.1 of extension combination and Lemmas 3.1.24, 3.1.38, 3.1.49 and 3.1.61, since the Byzantine action filters are idempotent by Lemma 3.1.22, we get (for some $k \geq 1$) that

$$\mathscr{E}^{\alpha\circ\beta} = \mathscr{E}^{\beta\circ\alpha} = \mathscr{E}^{k\cdot(\alpha+\beta)} = \mathscr{E}^{\alpha*\beta} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha} \cap PP^{\beta}, IS^{\alpha} \cap IS^{\beta}, \tau^{\beta,B}, \Psi^{\alpha} \cap \Psi^{\beta}), \quad (3.425)$$

from which (3.421) and (3.422) follow.

Regarding (3.420) let the following be two arbitrary compatible (w.r.t. some composition, k-intersection for $k \geq 1$ or fixpoint intersection) extensions $\mathscr{E}^{\alpha} \in \mathbf{JP-AFB}$ and $\mathscr{E}^{\beta} \in \mathbf{EnvJP-AFB}$. In either case we get

$$\mathscr{E}^{\alpha} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha}, IS^{\alpha}, \tau^{N,B}, \Psi^{\alpha}) \quad (3.426)$$
$$\mathscr{E}^{\beta} = (PP^{\beta}, IS^{\beta}, \tau^{N,B}, \Psi^{\beta}), \quad (3.427)$$

where $\mathscr{C}^{\alpha} \subset \mathscr{C}$, $PP^{\beta} \subseteq \mathscr{C}_{\epsilon} \times \mathscr{C}$, $IS^{\alpha}, IS^{\beta} \subseteq 2^{\mathscr{G}(0)}$, $\Psi^{\alpha}, \Psi^{\beta} \subseteq R$ and for the transition template $\tau^{N,B}$ the neutral event filter $filter_{\epsilon}^{N}$ and the Byzantine action filters $filter_{i}^{B}$ (for all $i \in \mathcal{A}$) are used. By Definition 3.2.1 of extension combination and Lemmas 3.1.24, 3.1.38, 3.1.49 and 3.1.61, since the Byzantine action filters are idempotent by Lemma 3.1.22, we get (for some $k \geq 1$) that

$$\mathscr{E}^{\alpha\circ\beta} = \mathscr{E}^{\beta\circ\alpha} = \mathscr{E}^{k\cdot(\alpha+\beta)} = \mathscr{E}^{\alpha*\beta} = (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\alpha} \cap PP^{\beta}, IS^{\alpha} \cap IS^{\beta}, \tau^{N,B}, \Psi^{\alpha} \cap \Psi^{\beta}), \quad (3.428)$$

from which (3.420) follows. □

Lemma 3.3.70. *The implementation class* **EnvJP − AFB** *is composable with* **EnvJP − AFB** *(itself),* **EvFJP − AFB** *and* **EvFEnvJP − AFB** *or formally*

$$(\mathbf{EnvJP-AFB}, \mathbf{EnvJP-AFB}) \in \mathscr{Q} \tag{3.429}$$
$$(\mathbf{EnvJP-AFB}, \mathbf{EvFJP-AFB}) \in \mathscr{Q} \tag{3.430}$$
$$(\mathbf{EnvJP-AFB}, \mathbf{EvFEnvJP-AFB}) \in \mathscr{Q} \tag{3.431}$$

Proof. Regarding (3.429) let the following be two arbitrary compatible (w.r.t. some composition, k-intersection for $k \geq 1$ or fixpoint intersection) extensions $\mathscr{E}^\alpha, \mathscr{E}^\beta \in \mathbf{EnvJP-AFB}$. We get that

$$\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^{N,B}, \Psi^\alpha) \tag{3.432}$$
$$\mathscr{E}^\beta = (PP^\beta, IS^\beta, \tau^{N,B}, \Psi^\beta), \tag{3.433}$$

where $PP^\alpha, PP^\beta \subseteq \mathscr{C}_\epsilon \times \mathscr{C}$, $IS^\alpha, IS^\beta \subseteq 2^{\mathscr{G}(0)}$, $\Psi^\alpha, \Psi^\beta \subseteq R$ and for the transition template $\tau^{N,B}$ the neutral event filter $filter_\epsilon^N$ and the Byzantine action filters $filter_i^B$ (for all $i \in \mathcal{A}$) are used. By Definition 3.2.1 of extension combination and Lemmas 3.1.24, 3.1.38, 3.1.49 and 3.1.61, since the Byzantine action filters are idempotent by Lemma 3.1.22, we get (for some $k \geq 1$) that

$$\mathscr{E}^{\alpha\circ\beta} = \mathscr{E}^{\beta\circ\alpha} = \mathscr{E}^{k\cdot(\alpha+\beta)} = \mathscr{E}^{\alpha*\beta} = (PP^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{N,B}, \Psi^\alpha \cap \Psi^\beta), \tag{3.434}$$

from which (3.429) follows.

Regarding (3.430) and (3.431) let the following be two arbitrary compatible (w.r.t. some composition, k-intersection for $k \geq 1$ or fixpoint intersection) extensions $\mathscr{E}^\alpha \in \mathbf{EnvJP-AFB}$ and either $\mathscr{E}^\beta \in \mathbf{EvFJP-AFB}$ or $\mathscr{E}^\beta \in \mathbf{EvFEnvJP-AFB}$. In either case we get

$$\mathscr{E}^\alpha = (PP^\alpha, IS^\alpha, \tau^{N,B}, \Psi^\alpha) \tag{3.435}$$
$$\mathscr{E}^\beta = (PP^\beta, IS^\beta, \tau^{\beta,B}, \Psi^\beta), \tag{3.436}$$

where $PP^\alpha, PP^\beta \subseteq \mathscr{C}_\epsilon \times \mathscr{C}$, $IS^\alpha, IS^\beta \subseteq 2^{\mathscr{G}(0)}$, $\Psi^\alpha, \Psi^\beta \subseteq R$ and for the transition template $\tau^{N,B}$ the neutral event filter $filter_\epsilon^N$ and the Byzantine action filters $filter_i^B$ (for all $i \in \mathcal{A}$) and in $\tau^{\beta,B}$ some arbitrary event filter $filter_\epsilon^\beta$ and the Byzantine action filters $filter_i^B$ (for all $i \in \mathcal{A}$) are used. By Definition 3.2.1 of extension combination and Lemmas 3.1.24, 3.1.38, 3.1.49 and 3.1.61, since the Byzantine action filters are idempotent by Lemma 3.1.22, we get (for some $k \geq 1$) that

$$\mathscr{E}^{\alpha\circ\beta} = \mathscr{E}^{\beta\circ\alpha} = \mathscr{E}^{k\cdot(\alpha+\beta)} = \mathscr{E}^{\alpha*\beta} = (PP^\alpha \cap PP^\beta, IS^\alpha \cap IS^\beta, \tau^{\beta,B}, \Psi^\alpha \cap \Psi^\beta), \tag{3.437}$$

from which (3.430) and (3.431) follow. □

Lemma 3.3.71. *All implementation classes* $IC^\alpha \in \mathscr{I}$ *are DCSP composable with* $\mathbf{JP_{DC}}$, $\mathbf{EvFJP_{DC}}$, $\mathbf{EvFJP_{DC}}$ *and* $\mathbf{EvFEnvJP_{DC}}$. *Formally for any class* $IC \in \mathscr{I}$

$$(IC, \mathbf{JP_{DC}}) \in \widetilde{\mathscr{Q}} \tag{3.438}$$
$$(IC, \mathbf{EnvJP_{DC}}) \in \widetilde{\mathscr{Q}} \tag{3.439}$$
$$(IC, \mathbf{EvFJP_{DC}}) \in \widetilde{\mathscr{Q}} \tag{3.440}$$
$$(IC, \mathbf{EvFEnvJP_{DC}}) \in \widetilde{\mathscr{Q}}. \tag{3.441}$$

Proof. Since the implementation classes $\mathbf{JP_{DC}}$, $\mathbf{EvFJP_{DC}}$, $\mathbf{EvFJP_{DC}}$ and $\mathbf{EvFEnvJP_{DC}}$ all use the neutral action filter functions and the neutral action filters are monotonic by Lemma 3.1.16 in the domain PD_{ϵ}^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$, the statement directly follows from Lemma 3.3.51. □

Lemma 3.3.72. *All implementation classes $IC^{\alpha} \in \mathscr{I}$ are composable with any implementation class $IC^{\beta} \in \mathscr{I}$ if we restrict IC^{β} to extensions $\mathscr{E}^{\beta} \in IC^{\beta}$ s.t. S^{β} is downward closed and the event filter of $\mathscr{E}^{\beta}$ is simply monotonic and all its action filters are monotonic for the domain PD_{ϵ}^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$.*

Proof. Follows directly from Lemma 3.3.51 and 3.3.52. □

Corollary 3.3.73. *For any class $IC^{\alpha} \in \mathscr{I}$*

$$(IC^{\alpha}, \mathbf{EvFEnvJP_{DC\,mono}}) \in \mathscr{Q} \tag{3.442}$$
$$(IC^{\alpha}, \mathbf{Others_{DC\,mono}}) \in \mathscr{Q}. \tag{3.443}$$

Proof. Follows directly from Lemma 3.3.72. □

Lemma 3.3.74. *Every implementation class is composable with $\mathbf{JP_{DC}}$, $\mathbf{EnvJP_{DC}}$, $\mathbf{EvFJP_{DC}}$. Formally for any implementation class $IC \in \mathscr{I}$*

$$(IC, \mathbf{JP_{DC}}) \in \mathscr{Q} \tag{3.444}$$
$$(IC, \mathbf{EnvJP_{DC}}) \in \mathscr{Q} \tag{3.445}$$
$$(IC, \mathbf{EvFJP_{DC}}) \in \mathscr{Q}. \tag{3.446}$$

Proof. Let $\mathscr{E}^{\alpha}$ be some arbitrary extension compatible (w.r.t. some composition, k-intersection for $k \geq 1$ or fixpoint intersection) with $\mathscr{E}^{\beta} \in \mathbf{JP_{DC}}$, $\mathscr{E}^{\gamma} \in \mathbf{EnvJP_{DC}}$, $\mathscr{E}^{\delta} \in \mathbf{EvFJP_{DC}}$, where

$$\begin{aligned}
\mathscr{E}^{\alpha} &= (PP^{\alpha}, IS^{\alpha}, \tau^{\alpha}, \Psi^{\alpha}) \\
\mathscr{E}^{\beta} &= (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\beta}, IS^{\beta}, \tau^{N,N}, \Psi^{\beta}) \\
\mathscr{E}^{\gamma} &= (PP^{\gamma}, IS^{\gamma}, \tau^{N,N}, \Psi^{\gamma}) \\
\mathscr{E}^{\delta} &= (\mathscr{C}_{\epsilon} \times \mathscr{C}^{\delta}, IS^{\delta}, \tau^{\delta,N}, \Psi^{\delta}),
\end{aligned}$$

where in τ_{α} some event filter $filter_{\epsilon}^{\alpha}$ and for all $i \in \mathcal{A}$ some action filters $filter_i^{\alpha}$, in $\tau_{N,N}$ the neutral event and action filters (for all $i \in \mathcal{A}$) and in $\tau_{\delta,N}$ some event filter $filter_{\epsilon}^{\delta}$ for all $i \in \mathcal{A}$ the neutral action filters are used.

Regarding (3.444) any combination (for $k \geq 1$) of the extensions $\mathscr{E}^{\alpha}$, $\mathscr{E}^{\beta}$ by Lemmas 3.1.24, 3.1.38, 3.1.49 and 3.1.61 leads to

$$\mathscr{E}^{\alpha \circ \beta} = \mathscr{E}^{\beta \circ \alpha} = \mathscr{E}^{k \cdot (\alpha + \beta)} = \mathscr{E}^{\alpha * \beta} = (PP^{\alpha} \cap \mathscr{C}_{\epsilon} \times \mathscr{C}^{\beta}, IS^{\alpha} \cap IS^{\beta}, \tau^{\alpha}, \Psi^{\alpha} \cap \Psi^{\beta}). \tag{3.447}$$

Since it has to hold that

$$filter_{\epsilon}^{\alpha} \subseteq filter_{\epsilon}^{N} \quad \wedge \quad (\forall i \in \mathcal{A})\ filter_i^{\alpha} \subseteq filter_i^{N}, \tag{3.448}$$

(3.444) follows from the downward closure of S^β. If by contradiction (3.448) did not hold, by Definition 3.1.1 of the neutral filters one of the filters from τ_α would violate the basic filter property and thus would not be a valid filter function.

Regarding (3.445) again similarly any combination (for $k \geq 1$) of the extensions $\mathscr{E}^\alpha$, $\mathscr{E}^\gamma$ by Lemmas 3.1.24, 3.1.38, 3.1.49 and 3.1.61 leads to

$$\mathscr{E}^{\alpha\circ\beta} = \mathscr{E}^{\beta\circ\alpha} = \mathscr{E}^{k\cdot(\alpha+\beta)} = \mathscr{E}^{\alpha*\beta} = (PP^\alpha \cap PP^\gamma, IS^\alpha \cap IS^\gamma, \tau^\alpha, \Psi^\alpha \cap \Psi^\gamma). \tag{3.449}$$

Again by (3.448) statement (3.445) follows from the downward closure of S^γ.

Lastly regarding (3.446) the different combinations of the extensions $\mathscr{E}^\alpha$, $\mathscr{E}^\delta$ result in (for $k \geq 1$)

$$\mathscr{E}^{\alpha\circ\delta} = (PP^\alpha \cap \mathscr{C}_\epsilon \times \mathscr{C}^\delta, IS^\alpha \cap IS^\delta, \tau^{\alpha\circ\delta,\alpha}, \Psi^\alpha \cap \Psi^\delta) \tag{3.450}$$

$$\mathscr{E}^{\delta\circ\alpha} = (PP^\alpha \cap \mathscr{C}_\epsilon \times \mathscr{C}^\delta, IS^\alpha \cap IS^\delta, \tau^{\delta\circ\alpha,\alpha}, \Psi^\alpha \cap \Psi^\delta) \tag{3.451}$$

$$\mathscr{E}^{k\cdot(\alpha+\delta)} = (PP^\alpha \cap \mathscr{C}_\epsilon \times \mathscr{C}^\delta, IS^\alpha \cap IS^\delta, \tau^{k\cdot(\alpha+\delta),\alpha}, \Psi^\alpha \cap \Psi^\delta) \tag{3.452}$$

$$\mathscr{E}^{\alpha*\delta} = (PP^\alpha \cap \mathscr{C}_\epsilon \times \mathscr{C}^\delta, IS^\alpha \cap IS^\delta, \tau^{\alpha*\delta,\alpha}, \Psi^\alpha \cap \Psi^\delta), \tag{3.453}$$

where

- in $\tau^{\alpha\circ\delta,\alpha}$ the event filter is $filter_\epsilon^{\alpha\circ\delta}$ and the action filters for all $i \in \mathcal{A}$ are $filter_i^\alpha$
- in $\tau^{\delta\circ\alpha,\alpha}$ the event filter is $filter_\epsilon^{\delta\circ\alpha}$ and the action filters for all $i \in \mathcal{A}$ are $filter_i^\alpha$
- in $\tau^{k\cdot(\alpha+\delta),\alpha}$ the event filter is $filter_\epsilon^{k\cdot(\alpha+\delta)}$ and the action filters for all $i \in \mathcal{A}$ are $filter_i^\alpha$
- in $\tau^{\alpha*\delta,\alpha}$ the event filter is $filter_\epsilon^{\alpha*\delta}$ and the action filters for all $i \in \mathcal{A}$ are $filter_i^\alpha$.

The adherence of the extensions (3.451), (3.452) and (3.453) to S^α follows from Lemma 3.3.51. The proof for the adherence of extension (3.450) to S^α is almost identical to the proof of Lemma 3.3.34 with the only difference that since the combined action filters for all $i \in \mathcal{A}$ are $filter_i^\alpha$ and by Definition 3.1.1 of the neutral action filters it holds that for any $i \in \mathcal{A}$, $X_\epsilon, X'_\epsilon \subseteq GEvents$, $X_1, X'_1 \subseteq \overline{GActions}_1$, ..., $X_n, X'_n \subseteq \overline{GActions}_n$

$$filter_i^\alpha\left(X_1, \ldots, X_n, X_\epsilon\right) \subseteq X_i = filter_i^N\left(X'_1, \ldots, X_i, \ldots, X'_n, X'_\epsilon\right), \tag{3.454}$$

by downward closure of S^α extension (3.450) also satisfies it. □

Lemma 3.3.75. *The implementation classes* $\mathbf{JP-AFB}$ *and* $\mathbf{EnvJP-AFB}$ *are composable with* $\mathbf{EvFEnvJP_{DC}}$. *Formally*

$$(\mathbf{JP-AFB}, \mathbf{EvFEnvJP_{DC}}) \in \mathscr{Q} \tag{3.455}$$

$$(\mathbf{EnvJP-AFB}, \mathbf{EvFEnvJP_{DC}}) \in \mathscr{Q}. \tag{3.456}$$

Proof. Let $\mathscr{E}^\alpha \in \mathbf{JP-AFB}$, $\mathscr{E}^\beta \in \mathbf{EnvJP-AFB}$ be two extensions that are compatible (w.r.t. some composition, k-intersection for $k \geq 1$ or fixpoint intersection) with $\mathscr{E}^\gamma \in \mathbf{EvFEnvJP_{DC}}$. Generally it holds that

$$\begin{aligned} \mathscr{E}^\alpha &= (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha, IS^\alpha, \tau^{N,B}, \Psi^\alpha) \\ \mathscr{E}^\beta &= (PP^\beta, IS^\beta, \tau^{N,B}, \Psi^\beta) \\ \mathscr{E}^\gamma &= (PP^\gamma, IS^\gamma, \tau^{\gamma,N}, \Psi^\gamma). \end{aligned} \tag{3.457}$$

Regarding (3.455) the combined extensions (for some $k \geq 1$) simplify to

$$\mathscr{E}^{\alpha\circ\gamma} = \mathscr{E}^{\gamma\circ\alpha} = \mathscr{E}^{k\cdot(\alpha+\gamma)} = \mathscr{E}^{\alpha*\gamma} = (\mathscr{C}_\epsilon \times \mathscr{C}^\alpha \cap PP^\gamma, IS^\alpha \cap IS^\gamma, \tau^{\gamma,B}, \Psi^\alpha \cap \Psi^\gamma), \quad (3.458)$$

where in $\tau^{\gamma,B}$ the event filter $filter_\epsilon^\gamma$ and the Byzantine action filters (for all $i \in \mathcal{A}$) $filter_i^B$ are used. Since the event filter in all combined extensions is the same as for $\mathscr{E}^\gamma$ and the Byzantine action filters are stricter than the neutral filters, by downward closure of S^γ the statement follows.

The reasoning for (3.456) is very similar. In this case the combined extensions (for some $k \geq 1$) simplify to

$$\mathscr{E}^{\beta\circ\gamma} = \mathscr{E}^{\gamma\circ\beta} = \mathscr{E}^{k\cdot(\beta+\gamma)} = \mathscr{E}^{\beta*\gamma} = (PP^\beta \cap PP^\gamma, IS^\beta \cap IS^\gamma, \tau^{\gamma,B}, \Psi^\beta \cap \Psi^\gamma) \quad (3.459)$$

where again in $\tau^{\gamma,B}$ the event filter $filter_\epsilon^\gamma$ and the Byzantine action filters (for all $i \in \mathcal{A}$) $filter_i^B$ are used. Since the event filter in all combined extensions is the same as for $\mathscr{E}^\gamma$ and the Byzantine action filters are stricter than the neutral filters, by downward closure of S^γ (3.456) follows and we are done. □

We have added Table 3.1 to paint a clear picture of the composability between the different implementation classes. The entries in this table are to be read as follows supposing x is the content of the entry, LC is the implementation class to the left and TC is the implementation class on the top:

- $x = c$ means that LC is composable with TC.
- $x = d$ means that LC is DCSP composable with TC.
- $x = f$ means that LC is fixpoint composable with TC.
- An empty entry means that LC can generally not be safely combined with TC.

Table 3.1: composability matrix of implementation classes

	Adm	JP	Env JP	EvF JP	EvF Env JP	JP - AFB	Env JP - AFB	EvF JP - AFB	EvF Env JP - AFB	Others	JP *DC*	Env JP *DC*	EvF JP *DC*	EvF Env JP *DC*	Others *DC*	EvF Env JP *DC mono*	Others *DC mono*
Adm	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
JP	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
EnvJP	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
EvFJP	c	c		f		c		f			c	c	c	d	d	c	c
EvFEnvJP	c	c		f		c		f			c	c	c	d	d	c	c
JP – AFB	c					c	c	c	c		c	c	c	c	d	c	c
EnvJP – AFB	c					c	c	c	c		c	c	c	c	d	c	c
EvFJP – AFB	c					c		f			c	c	c	d	d	c	c
EvFEnvJP – AFB	c					c		f			c	c	c	d	d	c	c
Others	c										c	c	c	d	d	c	c
JP$_{DC}$	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
EnvJP$_{DC}$	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c
EvFJP$_{DC}$	c	c		f		c		f			c	c	c	d	d	c	c
EvFEnvJP$_{DC}$	c	c		f		c		f			c	c	c	d	d	c	c
Others$_{DC}$	c										c	c	c	d	d	c	c
EvFEnvJP$_{DC\ mono}$	c	c		f		c		f			c	c	c	d	d	c	c
Others$_{DC\ mono}$	c										c	c	c	d	d	c	c

3.3.6 Guide to Extension Creation and Combination

In this section we aim at providing a clean guide on how to go about creating extensions as well as combining them.

Extension Creation

Given some property that we want to implement as an extension, there are the following possibilities. If the property is ...

- a liveness property $\Rightarrow$ use **Adm** (formulate an appropriate admissibility condition).
- a combination of safety S and liveness L properties, then continue with the next point below and simply put the liveness property L into the admissibility condition of whatever implementation class we end up with.
- a safety property S:

  Is S downward closed?

  - Yes, then implement S in
    1. $\mathbf{JP_{DC}}$ if not possible then
    2. $\mathbf{EnvJP_{DC}}$ if not possible then
    3. $\mathbf{EvFJP_{DC}}$ if not possible then
    4. $\mathbf{EvFEnvJP_{DC\,mono}}$ if not possible then
    5. $\mathbf{EvFEnvJP_{DC}}$ if not possible then
    6. $\mathbf{Others_{DC\,mono}}$ if not possible then
    7. $\mathbf{Others_{DC}}$
  - No

    Can S be implemented with $filter_i^B$ (for all $i \in \mathcal{A}$)?

    * Yes, then implement S in
      1. **JP – AFB** if not possible then
      2. **EvFJP – AFB** [3] if not possible then
      3. **EnvJP – AFB** if not possible then
      4. **EvFEnvJP – AFB**
    * No

      Such an extension would not make much sense, as it does not allow the environment control over agents' processing time (whether they are allowed to act). (The reason why we allow it in the case of downward closed safety properties is that such an extension can be combined with another one that uses the Byzantine action filter without violating its own safety property.) However if this is still desired, it is suggested to be implemented S in

[3] The reason why we prefer **EvFJP – AFB** over **EnvJP – AFB** is that the most prominent extensions that we will want to combine are the Byzantine extension $\mathscr{E}^B$ (enforces causality and control of the environment over whether agents are allowed to act) and the at most f-Byzantine extension $\mathscr{E}^{\leq f}$ (restricts the number of Byzantine agents). Since however $\mathscr{E}^B \in$ **EvFJP – AFB** and $\mathscr{E}^{\leq f} \in \mathbf{EvFJP_{DC}}$ by Corollary 3.5.4, both of them would generally not be composable with an extension in the implementation class **EnvJP – AFB** by Table 3.1.

1. **JP** if not possible then
2. **EvFJP** if not possible then
3. **EnvJP** if not possible then
4. **EvFEnvJP** if not possible then
5. **Others**

Extension Combination

Suppose we have a set EX of extensions that we want to combine. We distinguish between the following cases. If EX contains ...

- only extensions from implementation classes $\mathbf{JP_{DC}}$, $\mathbf{EnvJP_{DC}}$, $\mathbf{JP-AFB}$, $\mathbf{EnvJP-AFB}$, then any extension composition order will be safe by Lemmas 3.3.62, 3.3.63, 3.3.68, 3.3.69, 3.3.70, 3.3.74.
- extensions from other implementation classes, we do the following (obviously if certain implementation classes are not present in the set EX, we can skip the corresponding step):
  - We compose all the extensions of EX from the classes $\mathbf{EvFJP_{DC}}$ and $\mathbf{JP_{DC}}$. By Lemma 3.3.74, any such combination is safe. Let $\mathscr{E}^1 \in \mathbf{EvFJP_{DC}}$ be the extension resulting from this combination.
  - Next we combine all extensions $\mathscr{E}^{\overline{2}_{\overline{l}}} \in \mathbf{EvFEnvJP_{DC\,mono}}$ ($1 \leq \overline{l} \leq \overline{m}$, where $\overline{m}$ is the number of such extensions present in EX) via extension composition. By Lemma 3.3.73, any such combination is safe. Let $\mathscr{E}^{\overline{2}} \in \mathbf{EvFEnvJP_{DC\,mono}}$ be the extension resulting from this combination, meaning
  $$\mathscr{E}^{\overline{2}} = \mathscr{E}^{\overline{2}_1 \circ \overline{2}_2 \circ \ldots \circ \overline{2}_{\overline{m}}}. \tag{3.460}$$
  - Next we combine all extensions $\mathscr{E}^{2_l} \in \mathbf{EvFEnvJP_{DC}}$ ($1 \leq l \leq m$, where m is the number of such extensions present in EX) of EX via extension intersection. By Lemma 3.3.71, any such combination is safe. Let $\mathscr{E}^2 \in \mathbf{EvFEnvJP_{DC}}$ be the extension resulting from this combination, meaning
  $$\mathscr{E}^2 = \mathscr{E}^{2_1 + 2_2 + l \cdots + 2_m}. \tag{3.461}$$
  - Now we combine the two extensions $\mathscr{E}^2 \in \mathbf{EvFEnvJP_{DC}}$ and $\mathscr{E}^{\overline{2}} \in \mathbf{EvFEnvJP_{DC\,mono}}$ via (reverse) extension composition. Such a combination is safe by Lemma 3.3.71 and 3.3.73. Let $\mathscr{E}^3 \in \mathbf{EvFEnvJP_{DC}}$ be the extension resulting from this combination, meaning
  $$\mathscr{E}^3 = \mathscr{E}^{\overline{2} \circ 2}. \tag{3.462}$$
  - Next we compose $\mathscr{E}^1$ with $\mathscr{E}^3$. Again by Lemma 3.3.71, any such combination is safe. Let $\mathscr{E}^4 \in \mathbf{EvFEnvJP_{DC}}$ be the extension resulting from this combination, meaning
  $$\mathscr{E}^4 = \mathscr{E}^{1 \circ 3}. \tag{3.463}$$

– Now we compose all extensions $\mathscr{E}^{4_{l'}} \in \mathbf{EnvJP_{DC}}$ $(1 \leq l' \leq m'$, where m' is the number of such extensions present in $EX)$ of EX. By Lemma 3.3.74, any such combination is safe. The resulting extension we call $\mathscr{E}^5 \in \mathbf{EnvJP_{DC}}$,

$$\mathscr{E}^5 = \mathscr{E}^{4_1 \circ 4_2 \circ \ldots \circ 4_{m'}}. \tag{3.464}$$

– In the next step, we compose $\mathscr{E}^5$ with $\mathscr{E}^4$. This combination is safe by Lemma 3.3.74. We will refer to the resulting extension as $\mathscr{E}^6$

$$\mathscr{E}^6 = \mathscr{E}^{5 \circ 4}. \tag{3.465}$$

– Next we compose all extension $\mathscr{E}^{7_{l''}} \in \mathbf{JP - AFB}$ $(1 \leq l'' \leq m''$, where m'' is the number of such extensions present in $EX)$ of EX. By Lemma 3.3.68, any such combination is safe. We denote by $\mathscr{E}^8 \in \mathbf{JP - AFB}$ the resulting extension

$$\mathscr{E}^8 = \mathscr{E}^{7_1 \circ 7_2 \circ \ldots \circ 7_{m''}}. \tag{3.466}$$

– Next we combine all extension $\mathscr{E}^{9_{l'''}} \in \mathbf{EvFJP - AFB}$ $(1 \leq l''' \leq m'''$, where m''' is the number of such extensions present in $EX)$ of EX either by a suitable composition or fixpoint intersection.

  For this, a dependence analysis on the event filter functions of all the extensions $\mathscr{E}^{9_{l'''}}$ is needed, which checks, which kinds of events a certain event filter depends on and which kinds of events it removes. Based on these dependencies, a dependence graph between the extensions' event filters can be drawn. If one gets a circular dependence, it can be resolved via fixpoint intersection (by Lemma 3.3.67, fixpoint intersection of these extensions is always safe), where in the graph all nodes, part of the circular dependence, are joined into one node and all incoming and outgoing edges from the individual former nodes are attached to the new node. Note that the new node now represents the extension, which results from fixpoint intersection of its individual parts. After having resolved all circular dependencies, one can finally find an extension composition order based on the resulting dependence graph that is safe.

  We denote by $\mathscr{E}^{10} \in \mathbf{EvFJP - AFB}$ the resulting extension

$$\mathscr{E}^{10} = \mathscr{E}^{9_1 \circ 9_2 \circ \ldots \circ 9_{m'''}}. \tag{3.467}$$

– Now we compose the extension $\mathscr{E}^{10}$ with $\mathscr{E}^8$. Any such composition is safe by Lemmas 3.3.68 and 3.3.69. We denote the resulting extension by $\mathscr{E}^{11} \in \mathbf{EvFJP - AFB}$

$$\mathscr{E}^{11} = \mathscr{E}^{10 \circ 8}. \tag{3.468}$$

– In a next step, we compose $\mathscr{E}^{11}$ with $\mathscr{E}^6$. By Lemmas 3.3.66, 3.3.71 such a composition is safe. The resulting extension we denote as $\mathscr{E}^{12} \in \mathbf{EvFEnvJP - AFB}$

$$\mathscr{E}^{12} = \mathscr{E}^{11 \circ 6}. \tag{3.469}$$

– On the off chance that there are still extensions $\mathscr{E}^{13_j} \in \mathbf{Others_{DC}}$ $(1 \leq j \leq q$, where q is the number of such extensions $EX)$, $\mathscr{E}^{14_{j'}} \in \mathbf{EnvJP - AFB}$ $(1 \leq j' \leq q'$,

where q' is the number of such extensions EX) and $\mathscr{E}^{15_{j''}} \in \mathbf{EvFEnvJP - AFB}$ ($1 \leq j'' \leq q''$, where q'' is the number of such extensions in EX) present, we have to look for a fixpoint intersection order — that is a permutation p of the sequence $(13_1, 13_2, \ldots, 13_q, 14_1, 14_2, \ldots, 14_{q'}, 15_1, 15_2, \ldots, 15_{q''}$ — and check whether for this p it holds that (we denote by $\pi_k p$ the kth element in the sequence p) for all extensions $\mathscr{E}^\alpha \in EX$

$$\bigcup_{\chi \in \mathscr{E}^{\pi_1 p * \pi_2 p * \ldots \pi_{q+q'+q''} p}} R^\chi \quad \subseteq \quad \bigcup_{\chi' \in \mathscr{E}^\alpha} R^{\chi'}. \tag{3.470}$$

If no permutation p can be found that satisfies (3.470) for all $\mathscr{E}^\alpha \in EX$, then a safe combination of the extensions in EX is not possible.

Otherwise we denote the resulting extension as $\mathscr{E}^{16} \in \mathbf{Others}$

$$\mathscr{E}^{16} = \mathscr{E}^{\pi_1 p * \pi_2 p * \ldots * \pi_{q+q'+q''} p}. \tag{3.471}$$

– In our second to last step, we combine the extensions $\mathscr{E}^{12}$ and $\mathscr{E}^{16}$ via fixpoint intersection. Since this combination is generally not safe anyway a different combination method (other than fixpoint intersection) can be chosen: as in the case above we have to show that

$$\bigcup_{\chi \in \mathscr{E}^{12 * 16}} R^\chi \quad \subseteq \quad \bigcup_{\chi' \in \mathscr{E}^{12}} R^{\chi'} \tag{3.472}$$

and

$$\bigcup_{\chi \in \mathscr{E}^{12 * 16}} R^\chi \quad \subseteq \quad \bigcup_{\chi' \in \mathscr{E}^{16}} R^{\chi'} \tag{3.473}$$

hold. If both equations (3.472) and (3.473) cannot be satisfied, then a safe combination of the extensions in EX is not possible.

Otherwise we denote the resulting extension as $\mathscr{E}^{17}$

$$\mathscr{E}^{17} = \mathscr{E}^{12 * 16} \tag{3.474}$$

– Finally, the only extensions left in EX should be from the class **Adm**. [4] These can be safely composed (in any order) with $\mathscr{E}^{17}$ by Lemma 3.3.10 and we are done.

3.4 Liveness Property Extensions

In this section we introduce extensions to our framework, which restrict the general model by adding liveness properties. For encoding liveness properties, we will solely use the admissibility condition. Therefore, in this section we will only use the implementation class **Adm**.

[4] Note that we do not consider the non downward closed extensions with neutral action filters, as they are generally not composable with any extension that utilizes the Byzantine action filters.

3.4.1 Reliable Communication

In the **reliable communication** extension agents can behave arbitrarily. However the communication — the transmission of messages by the environment — is reliable for a particular set of (reliable) channels, i.e., a message that was sent through one of these (reliable) channels, is guaranteed to be delivered by the environment in finite time. This also holds for the delivery of messages to and from Byzantine agents. Since a Byzantine agent can always "choose" to ignore any messages it receives anyway, this does not restrict its Byzantine power to exhibit arbitrary behaviour. Formally, we define a set of (reliable) channels as $C \subseteq \mathcal{A}^2$.

The reliable communication property will be ensured by the admissibility condition $EDel_C$, which is a liveness property.

Definition 3.4.1 (Eventual Message Delivery)**.**

$$EDel_C = \left\{ r \in R \mid \left\{ \begin{array}{l} \left\{ \begin{array}{l} gsend(i,j,\mu,id) \ \in r_\epsilon(t) \\ \text{or} \\ (\exists A \in \{\text{🗓}\} \sqcup \overline{GActions_i})\, fake\,(i, gsend(i,j,\mu,id) \mapsto A) \in r_\epsilon(t) \end{array} \right\} \text{and} \\ (i,j) \in C \end{array} \right. \\ \left. \implies (\exists t' \in \mathbb{N})\ grecv(j,i,\mu,id) \in r_\epsilon(t') \right\} \tag{3.475}$$

Definition 3.4.2. We define by

$$\mathscr{E}^{RC_C} := \left(\mathscr{C}_\epsilon \times \mathscr{C}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^N, EDel_C\right) \tag{3.476}$$

the **reliable communication** extension.

Additionally, we define a variant of the reliable communication extension, the **eventual reliable communication** extension, where the communication is only reliable after a certain unknown point in time. Every message sent after this time is eventually received.

Definition 3.4.3 (Eventual Eventual Message Delivery)**.**

$$\begin{aligned} EEDel_C = \{ & r \in R \mid (\exists t \in \mathbb{N})(\forall t' \in \mathbb{N})(t' \geq t) \\ & \left\{ \begin{array}{l} \left\{ \begin{array}{l} gsend(i,j,\mu,id) \ \in r_\epsilon(t') \\ \text{or} \\ (\exists A \in \{\text{🗓}\} \sqcup \overline{GActions_i})\, fake\,(i, gsend(i,j,\mu,id) \mapsto A) \in r_\epsilon(t') \end{array} \right\} \text{and} \\ (i,j) \in C \end{array} \right. \\ & \implies (\exists t'' \in \mathbb{N})\, grecv(j,i,\mu,id) \in r_\epsilon(t'') \} \end{aligned} \tag{3.477}$$

Definition 3.4.4. We define by

$$\mathscr{E}^{ERC_C} := (\mathscr{C}_\epsilon \times \mathscr{C}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^N, EEDel_C) \tag{3.478}$$

the **eventual reliable communication** extension.

3.4.2 Stubborn Channels

The **stubborn channels** extension captures systems, in which a given message that is sent infinitely many times, eventually will be delivered. The crucial difference to the reliable communication extension is that here messages that have been sent can be discarded by the environment. However, sending the same message over and over again will eventually lead to its delivery even for Byzantine agents, or those that may become Byzantine in the future. As for the reliable communication extension, this property can only be ensured by an appropriate admissibility condition.

Definition 3.4.5 (Infinite Send Delivery)**.**

$$
\begin{aligned}
&ISDel = \{r \in R \mid ((\forall t \in \mathbb{N})(\exists t' \in \mathbb{N})(t' > t) \\
&\begin{cases}
\Big(\big(\ gsend(i,j,\mu,id)\ \in \Lambda_t \Rightarrow gsend(i,j,\mu,id') \in \Lambda_{t'}\big) \text{ and } \Big((\exists \widetilde{t} \in \mathbb{N})\ gsend(i,j,\mu,\widetilde{id}) \in \Lambda_{\widetilde{t}}\Big)\Big) \\
\text{or} \\
\Big(\Big((\exists A, A' \in \{\boxminus\} \sqcup \overline{GActions_i})\, fake\,(i, gsend(i,j,\mu,id) \mapsto A) \in \Lambda_t \Rightarrow fake\,(i, gsend(i,j,\mu,id') \mapsto A') \in \Lambda_{t'}\Big) \\
\text{and } (\exists \widetilde{t} \in \mathbb{N})(\exists \widetilde{A} \in \{\boxminus\} \sqcup \overline{GActions_i})\, fake\,\Big(i, gsend(i,j,\mu,\widetilde{id}) \mapsto \widetilde{A}\Big) \in \Lambda_{\widetilde{t}}\Big)
\end{cases} \\
&\qquad\qquad \Longrightarrow (\exists t'' \in \mathbb{N})\, grecv(j,i,\mu,id'') \in \Lambda_{t''}\}
\end{aligned}
\tag{3.479}
$$

Definition 3.4.6. We denote by

$$
\mathscr{E}^{StC} := (\mathscr{C}_\epsilon \times \mathscr{C}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^N, ISDel) \tag{3.480}
$$

the **stubborn channels** extension.

As for the reliable communication extension we also define a variant of the stubborn channels extension, where the stubborn channels property only holds after some unknown point in time.

Definition 3.4.7 (Eventual Infinite Send Delivery)**.**

$$
\begin{aligned}
&EISDel = \{r \in R \mid (\exists t \in \mathbb{N})(\forall t' \in \mathbb{N})(\exists t'' \in \mathbb{N})(t'' > t')(t' \geq t) \\
&\begin{cases}
\Big(\big(\ gsend(i,j,\mu,id)\ \in \Lambda_{t'} \Rightarrow gsend(i,j,\mu,id') \in \Lambda_{t''}\big) \text{ and } \Big((\exists \widetilde{t} \in \mathbb{N})\, gsend(i,j,\mu,\widetilde{id}) \in \Lambda_{\widetilde{t}}\Big)\Big) \\
\text{or} \\
\Big(\Big((\exists A, A' \in \{\boxminus\} \sqcup \overline{GActions_i})\, fake\,(i, gsend(i,j,\mu,id) \mapsto A) \in \Lambda_{t'} \Rightarrow fake\,(i, gsend(i,j,\mu,id') \mapsto A') \in \Lambda_{t''}\Big) \\
\text{and } (\exists \widetilde{t} \in \mathbb{N})(\exists \widetilde{A} \in \{\boxminus\} \sqcup \overline{GActions_i})\, fake\,\Big(i, gsend(i,j,\mu,\widetilde{id}) \mapsto \widetilde{A}\Big) \in \Lambda_{\widetilde{t}}\Big)
\end{cases} \\
&\qquad\qquad \Longrightarrow (\exists t''' \in \mathbb{N})\, grecv(j,i,\mu,id'') \in \Lambda_{t'''}\}
\end{aligned}
\tag{3.481}
$$

Definition 3.4.8. We denote by $\mathscr{E}^{EStC} = (\mathscr{C}_\epsilon \times \mathscr{C}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^N, EISDel)$ the **eventual stubborn channels** extension.

3.4.3 Fair Lossy Links

The **fair lossy links** extension models a communication medium, which ensures that when an infinite number of messages is sent from agent i to agent j, j receives an infinite number of those messages. Note that unlike for the reliable communication extension, messages can still be dropped, and unlike the stubborn channels extension, messages do not need to be the same.

Definition 3.4.9 (Infinite Send Infinite Receive)**.**

$$\begin{aligned} ISIR = \{r \in R \mid (\forall t \in \mathbb{N})(\exists t' \in \mathbb{N})(t' > t) \\ \Bigg(\Big(\big(gsend(i,j,\mu,id) \ \in \Lambda_t \Rightarrow gsend(i,j,\mu',id') \in \Lambda_{t'} \big) \text{ and } \Big((\exists \widetilde{t} \in \mathbb{N})\, gsend(i,j,\mu,\widetilde{id}) \in \Lambda_{\widetilde{t}} \Big) \Big) \\ \text{or} \\ \Big(\Big((\exists A, A' \in \{\text{📅}\} \sqcup \overline{GActions_i})\, fake\,(i, gsend(i,j,\mu,id) \mapsto A) \in \Lambda_t \Rightarrow fake\,(i, gsend(i,j,\mu',id') \mapsto A') \in \Lambda_{t'} \Big) \\ \text{and } (\exists \widetilde{t} \in \mathbb{N})(\exists \widetilde{A} \in \{\text{📅}\} \sqcup \overline{GActions_i})\, fake\left(i, gsend(i,j,\mu,\widetilde{id}) \mapsto \widetilde{A}\right) \in \Lambda_{\widetilde{t}} \Big) \Bigg) \\ \implies (\forall t'' \in \mathbb{N})(\exists t''' \in \mathbb{N})(t''' > t'') \Big(grecv(j,i,\mu'',id'') \in \Lambda_{t''} \Rightarrow grecv(j,i,\mu''',id''') \in \Lambda_{t'''} \Big) \text{ and} \\ \Big((\exists \widetilde{t'} \in \mathbb{N})\, grecv(j,i,\widetilde{\mu},\widetilde{id}) \in \Lambda_{\widetilde{t'}} \Big) \Big\} \end{aligned} \tag{3.482}$$

Definition 3.4.10. We denote by

$$\mathscr{E}^{FLL} := (\mathscr{C}_\epsilon \times \mathscr{C}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^N, ISIR) \tag{3.483}$$

the **fair lossy links** extension.

3.5 Safety Property Extensions

Here we present other extensions of our general framework, which encode safety properties. In order to make our framework as modular as possible, we implement these extensions according to Section 3.3, especially Section 3.3.5 (composability of implementation classes).

3.5.1 Asynchronous Byzantine Agents

In this section we define a framework extension corresponding to the base model from Chapter 2, mainly for the purpose of enabling combinations with other extensions.

Definition 3.5.1. We denote by

$$\mathscr{E}^{B} := (\mathscr{C}_\epsilon \times \mathscr{C}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^B, R) \tag{3.484}$$

the **asynchronous Byzantine agents** extension.

3.5.2 $\leq f$ Byzantine Agents

In this extension the number of Byzantine agents for any run is limited to some number $f \in \mathbb{N}$.

Definition 3.5.2. We denote by

$$\mathscr{E}^{\leq f} := (\mathscr{C}_\epsilon \times \mathscr{C}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^{\leq f,N}, R) \tag{3.485}$$

the $\leq f$ **Byzantine agents** extension (where in $\tau^{\leq f,N}$ the event filter $filter_\epsilon^{\leq f}$ (2.91) and the neutral action filters are used).

The reason why we define the $\leq f$ Byzantine agents extension with neutral action filters unlike in Definition 2.6.7, is to accommodate easier composability, as without the Byzantine action filters by Lemma 3.5.3 below, this extension's safety property is downward closed. If we were to include the Byzantine action filters, by Lemma 3.3.53, this would not be the case anymore.

Lemma 3.5.3. *$S^{\leq f}$ is downward closed (for any $f \in \mathbb{N}$).*

Proof. Suppose by contradiction that $S^{\leq f}$ is not downward closed. This implies that there exists some $h \in \mathscr{G}$ s.t. there exists some $X \in S^{\leq f}(h)$ s.t. there exists some $X' \subseteq X$ s.t.

$$X' \notin S^{\leq f}(h). \tag{3.486}$$

From this it becomes immediately clear that

$$X \neq \varnothing \quad \wedge \quad X \neq X', \tag{3.487}$$

which implies that

$$X' \subset X. \tag{3.488}$$

Since by Definition 3.5.2 the extension $\mathscr{E}^{\leq f}$ does not restrict the protocol sets

$$\begin{gathered}(\exists P_\epsilon, P'_\epsilon \in \mathscr{C}_\epsilon)(\exists P, P' \in \mathscr{C})(\exists X_\epsilon \in P_\epsilon(|h|))(\exists X_1 \in P_1(h_1)) \dots (\exists X_n \in P_n(h_n)) \\ (\exists X'_\epsilon \in P'_\epsilon(|h|))(\exists X'_1 \in P'_1(h_1)) \dots (\exists X'_n \in P'_n(h_n)) \\ \bigsqcup_{i=1}^{n} label\left(X_i, |h|\right) \sqcup X_\epsilon \;=\; X \quad \wedge \quad \bigsqcup_{i=1}^{n} label\left(X'_i, |h|\right) \sqcup X'_\epsilon \;=\; X'\end{gathered} \tag{3.489}$$

holds. Since $X' \notin S^{\leq f}(h)$ was assumed, we get that

$$filter_\epsilon^{\leq f}\left(h, X'_\epsilon, X'_1, \dots, X'_n\right) = \widetilde{X'_\epsilon} \subset X'_\epsilon, \tag{3.490}$$

as the neutral action filters do not remove any elements

$$(\forall i \in \mathcal{A})\ filter_i^N\left(X'_1, \dots, X'_n, \widetilde{X'_\epsilon}\right) = X'_i. \tag{3.491}$$

This means that by definition of $filter_\epsilon^{\leq f}$ (2.91)

$$\left|\mathcal{A}(Failed\,(h)) \cup \left\{i \in \mathcal{A} \mid X'^B_{\epsilon_i} \neq \varnothing\right\}\right| > f, \tag{3.492}$$

where for $X'^B_{\epsilon_i}$ the definition from (2.90) is used. However, since $X' \subseteq X$, we also get that (for all $i \in \mathcal{A}$) $X'^B_{\epsilon_i} \subseteq X^B_{\epsilon_i}$ and as a result

$$\left\{i \in \mathcal{A} \mid X'^B_{\epsilon_i} \neq \varnothing\right\} \subseteq \left\{i \in \mathcal{A} \mid X^B_{\epsilon_i} \neq \varnothing\right\} \tag{3.493}$$

holds, which would imply that also

$$\left|\mathcal{A}(Failed\,(h)) \cup \left\{i \in \mathcal{A} \mid X^B_{\epsilon_i} \neq \varnothing\right\}\right| > f \tag{3.494}$$

and as a result lead to

$$filter^{\leq f}_{\epsilon}\,(h, X_\epsilon, X_1, \ldots, X_n) = \widetilde{X_\epsilon} \subset X_\epsilon. \tag{3.495}$$

This however is impossible due to Lemma 3.3.48. □

Corollary 3.5.4. $\mathscr{E}^{\leq f} \in \mathbf{EvFJP_{DC}}$.

Proof. Follows immediately from Definition 3.5.2 and Lemma 3.5.3. □

3.5.3 Time-bounded Communication

We say that communication is time-bounded if for every channel and for every message there is an upper-bound (possibly infinite) on the transmission time. Since the transmission is not reliable a priori, the **time-bounded communication** extension only specifies the time window during which the delivery of a message can occur. In order to gain flexibility, bounds can be changed depending on the sending time and depending on the message too — for instance a byte of data and picture will not have the same time bound. We encode these bounds in an upper-bound structure defined as follows:

Definition 3.5.5. For the first ordinal number ω, the agents $(i, j) \in \mathcal{A}^2$ and the channel $i \mapsto j$, we define the message transmission upper-bound for the channel $i \mapsto j$ as follows

$$\delta_{i\mapsto j} : \quad Msgs \times \mathbb{N} \quad \to \quad \mathbb{N} \cup \{\omega\}. \tag{3.496}$$

We define an upper bound structure as

$$\Delta := \bigcup_{(i,j)\in\mathcal{A}^2} \{\delta_{i\mapsto j}\}. \tag{3.497}$$

Since the time-bounded safety property is downward closed (we will show this formally below), we implement it by restriction of the set of environment protocols.

Definition 3.5.6 (Time-bounded Communication Environment Protocols)**.** For an upper-bound structure Δ, we define the set of time-bounded communication environment protocols as

$$\begin{aligned}\mathscr{C}_\epsilon^{TC_\Delta} := \{P_\epsilon \in \mathscr{C}_\epsilon \,|(\forall t \in \mathbb{N})(\forall X_\epsilon \in P_\epsilon(t)) \\ grecv(j, i, \mu, id(i, j, \mu, k, t')) \in X_\epsilon \;\to\; t' + \delta_{i\mapsto j}\,(\mu, t') \geq t\}.\end{aligned} \tag{3.498}$$

Definition 3.5.7. For an upper-bound structure Δ

$$\mathscr{E}^{TC_\Delta} := (\mathscr{C}_\epsilon^{TC_\Delta} \times \mathscr{C}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^N, R) \tag{3.499}$$

denotes the **Time-bounded Communication** extension.

Lemma 3.5.8. *S^{TC_Δ} is downward closed.*

Proof. Suppose that by contradiction S^{TC_Δ} is not downward closed. This implies that

$$(\exists h \in \mathscr{G})(\exists X \in S^{TC_\Delta})(\exists X' \subseteq X)\ X' \notin S^{TC_\Delta}(h). \tag{3.500}$$

It immediately follows that $X' \subset X$. Since in τ^N the neutral (event and action) filters are used we further get that

$$\begin{aligned}(\exists P_\epsilon \in \mathscr{C}_\epsilon^{TC_\Delta})(\exists P \in \mathscr{C})(\exists t \in \mathbb{N})\ X_\epsilon \in P_\epsilon(t) \wedge X_1 \in P_1(h_1) \wedge \cdots \wedge X_n \in P_n(h_n) \wedge \\ X = X_\epsilon \sqcup X_1 \sqcup \cdots \sqcup X_n.\end{aligned} \tag{3.501}$$

Since the set of joint protocols is unrestricted there exists some joint protocol P' ensuring that together with P_ϵ

$$(\forall X_1' \subseteq X_1) \ldots (\forall X_n' \subseteq X_n)\ X_\epsilon \sqcup X_1' \sqcup \cdots \sqcup X_n' \in S^{TC_\Delta}(h). \tag{3.502}$$

Therefore, we conclude that the violation of X' has to be caused by some $X_\epsilon' \subset X_\epsilon = X \sqcup \mathit{GEvents}$.

From $X \in S^{TC_\Delta}(h)$ we conclude that

$$(\forall t \in \mathbb{N})\ \ \mathit{grecv}(j, i, \mu, \mathit{id}(i, j, \mu, k, t')) \in X_\epsilon\ \rightarrow\ t' + \delta_{i \mapsto j}(\mu, t') \geq t. \tag{3.503}$$

By semantics of "$\rightarrow$" and since $X_\epsilon' \subseteq X_\epsilon$ we get

$$\Big(\mathit{grecv}(j, i, \mu, \mathit{id}(i, j, \mu, k, t')) \in X_\epsilon'\Big) \rightarrow \Big(\mathit{grecv}(j, i, \mu, \mathit{id}(i, j, \mu, k, t')) \in X_\epsilon\Big). \tag{3.504}$$

Using (3.504) in (3.503) we get

$$\begin{aligned}(\forall t \in \mathbb{N})\ \Big(\mathit{grecv}(j, i, \mu, \mathit{id}(i, j, \mu, k, t')) \in X_\epsilon'\Big) \rightarrow \Big(\mathit{grecv}(j, i, \mu, \mathit{id}(i, j, \mu, k, t')) \in X_\epsilon\Big) \rightarrow \\ \Big(t' + \delta_{i \mapsto j}(\mu, t') \geq t\Big).\end{aligned} \tag{3.505}$$

Finally from (3.505) by transitivity of "$\rightarrow$" we get that

$$(\forall t \in \mathbb{N})\ \Big(\mathit{grecv}(j, i, \mu, \mathit{id}(i, j, \mu, k, t')) \in X_\epsilon'\Big) \rightarrow \Big(t' + \delta_{i \mapsto j}(\mu, t') \geq t\Big). \tag{3.506}$$

Hence, we conclude that $X' \in S^{TC_\Delta}(h)$ and we are done. □

3.5.4 Synchronous Communication

The **Synchronous Communication** extension guarantees for a set of synchronous communication channels $C \subseteq \mathcal{A}^2$ that whenever a message is correctly received, it has been sent during the same round. This means that it is a special case of the time-bounded communication extension.

Definition 3.5.9 (Synchronous Communication Environment Protocols)**.** We define the synchronous message delay as

$$\delta_{i \mapsto j}^{SC_C}(\mu, t) := \begin{cases} 0 & \text{if } (i, j) \in C \\ \omega & \text{otherwise} \end{cases} \tag{3.507}$$

We define the synchronous communication upper bound structure Δ^{SC_C} as

$$\Delta^{SC_C} := \bigcup_{(i,j)\in\mathcal{A}^2} \{\delta_{i\mapsto j}^{SC_C}\}. \tag{3.508}$$

$$\mathscr{C}_\epsilon^{SC_C} := \mathscr{C}_\epsilon^{TC_{\Delta^{SC_C}}} \tag{3.509}$$

Definition 3.5.10. We denote by

$$\mathscr{E}^{SC_C} := \left(\mathscr{C}_\epsilon^{SC_C} \times \mathscr{C}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^N, R\right) \tag{3.510}$$

the **Synchronous Communication** extension.

Lemma 3.5.11. *S^{SC_C} is downward closed.*

Proof. Follows from Lemma 3.5.8, as the synchronous communication extension is just an instance of the time-bounded communication extension (3.509). □

Corollary 3.5.12. $\mathscr{E}^{SC_C} \in \mathbf{EnvJP_{DC}}$.

Proof. Follows from Definition 3.5.10 and Lemma 3.5.11. □

3.5.5 Multicast Communication

In the **multicast communication** paradigm, each agent has several multicast channels at its disposal and is restricted to sending messages using these particular channels. In this section, we provide a software based multicast, meaning that only correct agents have to adhere to this behavior (further along we provide a hardware based multicast as well, where also Byzantine agents are forced to exhibit this multicast behavior).

First, we define a **multicast communication problem**. For each agent $i \in \mathcal{A}$ we define a collection Mc_i of groups of agents it can send messages to.

Definition 3.5.13. For each agent $i \in \mathcal{A}$ the set of available multicast channels is

$$Mc_i \subseteq 2^{\mathcal{A}} \setminus \{\varnothing\}. \tag{3.511}$$

The **multicast communication problem** is the tuple of these collections of communication channels:

$$Ch = (Mc_1, \ldots, Mc_n) \tag{3.512}$$

We denote the set of recipients for the kth copy of a message μ, i.e., for μ_k that has been sent according to some set $X \subseteq \overline{Actions}$ by

$$Rec_X(\mu_k) = \{j \mid send(j, \mu_k) \in X\}. \tag{3.513}$$

Since we implement a software based multicast (and since we want our extensions to be modular) we will naturally use a restriction of the joint protocol to do so.

Definition 3.5.14 (Multicast Joint Protocols)**.** For a multicast communication problem Ch, we define the set of multicast joint protocols as

$$\mathscr{C}^{MC_{Ch}} = \{(P_1, \ldots, P_n) \in \mathscr{C} \mid (\forall i \in \mathcal{A})(\forall h_i \in \mathscr{L}_i)(\forall X \in P_i(h_i))(\forall \mu \in Msgs)(\forall k \in \mathbb{N}) \\ Rec_X(\mu_k) \neq \varnothing \;\rightarrow\; Rec_X(\mu_k) \in Mc_i\}. \tag{3.514}$$

Definition 3.5.15. For a multicast communication problem Ch, the **multicast communication** extension is defined by

$$\mathscr{E}^{MC_{Ch}} := \left(\mathscr{C}_\epsilon \times \mathscr{C}^{MC_{Ch}}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^{N,B}, R\right), \tag{3.515}$$

where in $\tau^{N,B}$ the neutral event and the Byzantine action filters are used (for all $i \in \mathcal{A}$).

An important special case of the multicast communication problem is the broadcast communication problem, where each agent must broadcast each message to all the agents:

Definition 3.5.16. The **broadcast communication extension** $\mathscr{E}^{BC}$ is a multicast communication extension $\mathscr{E}^{MC_{BCh}}$ for

$$BCh = (\underbrace{\{\mathcal{A}\}, \ldots, \{\mathcal{A}\}}_{n}) \tag{3.516}$$

$$\mathscr{E}^{BC} = (\mathscr{C}_\epsilon \times \mathscr{C}^{MC_{BCh}}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^{N,B}, R), \tag{3.517}$$

where in $\tau^{N,B}$ the neutral event filter and the Byzantine action filters (for all $i \in \mathcal{A}$) are used.

Corollary 3.5.17. $\mathscr{E}^{MC_{Ch}} \in \mathbf{JP - AFB}$.

Proof. Follows from Definition 3.5.15. □

3.5.6 Synchronous Agents

For the **synchronous agents** extension, we first introduce the notion of **virtual rounds**.

Definition 3.5.18 (Virtual Round)**.** For a given transitional run $r \in R$ and timestamp $t \in \mathbb{N}$, the round $t.5$ is called a **virtual round** iff

$$(\forall i \in \mathcal{A}) \quad \beta^t_{g_i}(r) \neq \varnothing, \tag{3.518}$$

where $\beta^t_{g_i}(r) \subseteq SysEvents_i$, see Definition 2.2.22.

Definition 3.5.19 (Number of Virtual Rounds)**.** We denote the number of virtual rounds for a history $h = r(t) \in \mathscr{G}$ given some transitional run $r \in R^{trans}$ and timestamp $t \in \mathbb{N}$ by $NVR(r(t))$.

Informally, a virtual round is a round, in which for all agents i some event $\in \{go(i), sleep(i), hibernate(i)\}$ occurs. By forcing correct agents to act only during virtual rounds, we will ensure that all correct agents execute their protocol in synchronous steps, while still allowing Byzantine agents to both skip virtual rounds or take any number of additional fake actions in

between them. Since the synchronous agents safety property is generally not downward closed, it will be implemented with an appropriate event filter function (and set of joint protocols) that removes $go(i)$ events for all agents in a given round t unless for every agent i one of the system events $(go(i), sleep(i), hibernate(i))$ occurs.

Definition 3.5.20 (Synchronous agents event filter function)**.** Given a global history $h \in \mathscr{G}$ and sets $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ we define the event filter function for the synchronous agents extension as

$$filter^S_\epsilon(h, X_\epsilon, X_1, \ldots, X_n) := \begin{cases} X_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\} & (\exists j \in \mathcal{A})\{go(j), sleep(j), hibernate(j)\} \cap X_\epsilon = \varnothing, \\ X_\epsilon & \text{otherwise.} \end{cases} \tag{3.519}$$

Lemma 3.5.21. *For the general asynchronous Byzantine framework given two $\tau^B_{P_\epsilon,P}$-transitional runs $r, r' \in R$ and timestamps $t, t' \in \mathbb{N} \setminus \{0\}$, an agent $i \in \mathcal{A}$ cannot distinguish*

- *a round $t.5$ in run r, where a set of events $Q \subseteq \overline{GEvents}_i \sqcup BEvents_i$, $Q \neq \varnothing$ was observed by i, but no $go(i)$ occurred $\Rightarrow$*

$$go(i) \notin \beta^t_{g_i}(r), \quad \beta^t_i(r) = \varnothing, \quad \overline{\beta}^t_{\epsilon_i}(r) \sqcup \beta^t_{b_i}(r) = Q$$

- *from a round $t'.5$ in run r', where the same set of events Q was observed by i, $go(i)$ occurred, but the protocol prescribed the empty set ($\varnothing \in P_i(r'(t'))$), which was chosen by the adversary $\Rightarrow$*

$$go(i) \in \beta^{t'}_{g_i}(r'), \quad \beta^{t'}_i(r') = \varnothing, \quad \overline{\beta}^{t'}_{\epsilon_i}(r') \sqcup \beta^{t'}_{b_i}(r') = Q.$$

Proof. This immediately follows from the definition of the update function (2.39) and (2.19), as in this scenario (for $r_i(t+1) = [\lambda_m, \ldots, \lambda_1, \lambda_0]$ and $r'_i(t'+1) = [\lambda'_{m'}, \ldots, \lambda'_1, \lambda'_0]$) $\lambda_m = \lambda'_{m'} = Q$. □

Since Lemma 3.5.21 holds, we need to restrict the behavior of synchronous agents in such a way that enables them to distinguish these two scenarios, as it is desirable for at least non Byzantine agents in a synchronous setting to correctly infer from their local state the number of virtual rounds that have passed. We do this by modifying the joint protocol, where we specify that the sets of actions that an agent protocol returns for a given history have to be supersets of the set containing the special internal action $\{⏱\}$, which we assume to be available for all agents:

$$(\forall i \in \mathcal{A})\ ⏱ \in \overline{Actions}_i. \tag{3.520}$$

Definition 3.5.22 (Synchronous joint protocols)**.** We define the synchronous joint protocols as

$$\mathscr{C}^S := \left\{(P_1, \ldots P_n) \in \mathscr{C} \mid (\forall i \in \mathcal{A})(\forall h_i \in \mathscr{L}_i)(\forall D \in P_i(h_i))\ \{⏱\} \subseteq D\right\}. \tag{3.521}$$

Definition 3.5.23. We denote by

$$\mathscr{E}^S := \left(\mathscr{C}_\epsilon \times \mathscr{C}^S, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^{S,B}, R\right) \tag{3.522}$$

the **synchronous agents extension**, where in the transition template $\tau^{S,B}$ the synchronous agents event filter Definition 3.5.20 and the Byzantine action filters (2.25) (for all $i \in \mathcal{A}$) are used.

Corollary 3.5.24. $\mathscr{E}^S \in$ **EvFJP – AFB**.

Proof. Follows immediately from Definition 3.5.23. □

Lemma 3.5.25. *For any global history $h \in \mathscr{G}$ and domain PD_ϵ^{t-coh}, $2^{\overline{GActions}_1}$, ..., $2^{\overline{GActions}_n}$ the filter functions of the synchronous agents extension are monotonic.*

Proof. By Definition 3.5.23 of the synchronous agents extension, it uses the Byzantine action filters. Since by Lemma 3.1.17 the Byzantine action filter functions are monotonic (for the domain PD_ϵ^{t-coh}, $2^{\overline{GActions}_1}$, ..., $2^{\overline{GActions}_n}$), we are done for the action filters.

Regarding the synchronous agents event filter function (3.519), for some $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \in 2^{\overline{GActions}_1}$, ..., $X_n \in 2^{\overline{GActions}_n}$ and $X'_\epsilon \subseteq X_\epsilon$, $X'_1 \subseteq X_1$, ..., $X'_n \subseteq X_n$, let

$$filter_\epsilon^S\,(h, X_\epsilon, X_1, \ldots, X_n) = Y_\epsilon \tag{3.523}$$

and

$$filter_\epsilon^S\,(h, X'_\epsilon, X'_1, \ldots, X'_n) = Y'_\epsilon. \tag{3.524}$$

Since the synchronous event filter is independent of the sets of actions, we only need to focus on the event set. We can differentiate between four scenarios:

1. $Y_\epsilon = X_\epsilon \ \wedge \ Y'_\epsilon = X'_\epsilon$
   In this case the statement follows from our assumption that $X'_\epsilon \subseteq X_\epsilon$.

2. $Y_\epsilon = X_\epsilon \ \wedge \ Y'_\epsilon \subset X'_\epsilon$
   The statement holds by transitivity of "$\subseteq$", as $Y'_\epsilon \subseteq X'_\epsilon \subseteq X_\epsilon = Y_\epsilon$.

3. $Y_\epsilon \subset X_\epsilon \ \wedge \ Y'_\epsilon = X'_\epsilon$
   By definition of the synchronous agents event filter (3.519), this implies that

   $$(\exists j \in \mathcal{A})\,\{go(j), sleep(j), hibernate(j)\} \cap X_\epsilon = \varnothing \tag{3.525}$$

   is true. But

   $$(\exists j \in \mathcal{A})\,\{go(j), sleep(j), hibernate(j)\} \cap X'_\epsilon = \varnothing \tag{3.526}$$

   has to hold as well, as otherwise for some $i \in \mathcal{A}$ for which $\{go(i), sleep(i), hibernate(i)\} \cap X_\epsilon = \varnothing$ is true, we would have that

   $$\{go(i), sleep(i), hibernate(i)\} \cap X'_\epsilon \neq \varnothing, \tag{3.527}$$

which however would contradict our assumption that $X'_\epsilon \subseteq X_\epsilon$. Hence, the results of the filters are

$$\begin{aligned} Y_\epsilon &= X_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\} \\ Y'_\epsilon &= X'_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\}. \end{aligned} \tag{3.528}$$

Since $X'_\epsilon \subseteq X_\epsilon$ it follows that

$$Y'_\epsilon = X'_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\} \subseteq X_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\} = Y_\epsilon \subset X_\epsilon \tag{3.529}$$

and we are done.

4. $Y_\epsilon \subset X_\epsilon \ \wedge \ Y'_\epsilon \subset X'_\epsilon$
In this case the results of the filters are

$$\begin{aligned} Y_\epsilon &= X_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\} \\ Y'_\epsilon &= X'_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\}. \end{aligned} \tag{3.530}$$

Just as in the previous case, since $X'_\epsilon \subseteq X_\epsilon$, it follows that

$$X'_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\} \subseteq X'_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\} \tag{3.531}$$

and we are done.

□

Lemma 3.5.26. *For any global history $h \in \mathscr{G}$ and sets $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \in 2^{\overline{GActions_1}}$, ..., $X_n \in 2^{\overline{GActions_n}}$, the filter functions of the synchronous agents extension are idempotent.*

Proof. Since the synchronous agents extension uses the Byzantine action filter functions, which are idempotent by Lemma 3.1.22 in our domain, the statement trivially follows for the action filters.

Regarding the synchronous agents event filter, let

$$\begin{aligned} filter_\epsilon^S(h, X_\epsilon, X_1, \ldots, X_n) &= Y_\epsilon \\ filter_\epsilon^S(h, Y_\epsilon, X_1, \ldots, X_n) &= Y'_\epsilon. \end{aligned} \tag{3.532}$$

We need to show that $Y_\epsilon = Y'_\epsilon$. By Definition 3.5.20 of the synchronous agents event filter, there are two possibilities:

- $(\exists j \in \mathcal{A})\, \{go(j), sleep(j), hibernate(j)\} \cap X_\epsilon = \varnothing$
In this case the result of the filtering is

$$Y_\epsilon = X_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\}. \tag{3.533}$$

Regarding a second application of the synchronous agents filter, we again differentiate between two possibilities:

– $(\exists j \in \mathcal{A})\,\{go(j), sleep(j), hibernate(j)\} \cap Y_\epsilon = \varnothing$
In this case we get by definition of the synchronous agents filter that

$$Y'_\epsilon = Y_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\}. \tag{3.534}$$

Using (3.533) in (3.534) we get

$$Y'_\epsilon = (X_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\}) \setminus \{go(i) \mid i \in \mathcal{A}\}. \tag{3.535}$$

Since by semantics of set difference "\"

$$(X_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\}) \setminus \{go(i) \mid i \in \mathcal{A}\} = X_\epsilon \setminus \{go(i) \mid i \in \mathcal{A}\} \tag{3.536}$$

holds, we conclude that $Y_\epsilon = Y'_\epsilon$.

– $(\forall j \in \mathcal{A})\,\{go(j), sleep(j), hibernate(j)\} \cap Y_\epsilon \neq \varnothing$
By definition of the synchronous agents filter

$$Y'_\epsilon = Y_\epsilon, \tag{3.537}$$

from which the statement follows.

- $(\forall j \in \mathcal{A})\,\{go(j), sleep(j), hibernate(j)\} \cap X_\epsilon \neq \varnothing$
In this case by definition of the synchronous agents event filter, we get

$$Y_\epsilon = X_\epsilon, \tag{3.538}$$

therefore the statement $Y'_\epsilon = Y_\epsilon$ trivially holds and we are done.

□

Lemma 3.5.27. *An agent i in a synchronous agents context executes its protocol only during virtual rounds, i.e., for every $\chi \in \mathscr{E}^S$ and $r \in R^\chi$, $go(i) \in \beta^t_{g_i}(r)$ if $t.5$ is a virtual round.*

Proof. From the definition of virtual rounds (3.518) and the synchronous agents event filter function (3.519), it immediately follows that in a synchronous agents context $go(i)$ events can only ever occur during a virtual round. Hence the Lemma follows. □

Lemma 3.5.28. *Given a correct agent $i \in \mathcal{A}$, a $\tau^{S,B}_{P_\epsilon,P^S}$-transitional run $r \in R$ (where $P^S \in \mathscr{C}^S$) and any $t \geq 0$ it holds that $NVR(r(t)) \leq |r_i(t)| - 1$.*

Proof. Since by Lemma 3.5.27 it follows that a correct agent has to receive $go(i)$ in a virtual round and by definition of the update function (2.19), (2.39), for every update the length of the agent history increases by one, the lemma follows from Definition 2.1.22. □

Lemma 3.5.29. *For a correct agent i, a $\tau^{S,B}_{P_\epsilon,P^S}$-transitional run r (where $P^S \in \mathscr{C}^S$), some timestamp $t' \geq 1$, agent i's local history $r_i(t') = h_i = [\lambda_m, \ldots, \lambda_1, \lambda_0]$ (given the global history $h = r(t') \in \mathscr{G}$) and some round $(t-1).5$ ($t' \geq t \geq 1$), there exists some $a \in \overline{Actions_i}$ such that $a \in \lambda_{k_t}$ where $\lambda_{k_t} = \sigma(\beta^{t-1}_{\epsilon_i}(r) \sqcup \beta^{t-1}_i(r))$ if and only if $(t-1).5$ is a virtual round.*

Proof. $\Rightarrow$
From Lemma 3.5.27, we know that an agent can only execute its protocol during virtual rounds. Therefore, since agent i is assumed to be correct and $(t-1).5$ is a virtual round, it follows that $\{go(i)\} = \beta_{g_i}^{t-1}(r)$ ($sleep(i)$ or $hibernate(i)$ would make the agent Byzantine). By (2.19), (2.39) (the definition of the update function) and Definition 3.5.22 (the definition of the synchronous agents joint protocols, which dictates that at least ⏱ has to be among the attempted actions, hence the empty set can never be issued) an action $a \in \overline{Actions_i}$ such that $a \in \lambda_{k_t}$ has to exist.
$\Leftarrow$
Suppose there exists an action $a \in \overline{Actions_i}$ such that $a \in \lambda_{k_t}$. Since agent i is assumed to be correct, by Lemma 3.5.27 agents only execute their protocol during virtual rounds and by the definition of the update function (2.39) (and $aware(i, X_\epsilon)$ (2.19)) round $(t-1).5$ has to be a virtual round. □

Now we show that agents in the synchronous agents context can actually distinguish the two scenarios in Lemma 3.5.21 due to the joint protocol from Definition 3.5.22

Lemma 3.5.30. *For any agent $i \in \mathcal{A}$, any run $r \in R^\chi$, where $\chi \in \mathscr{E}^S$ and any timestamp $t \in \mathbb{N}$, it holds that*

$$\{go(i)\} = \beta_{g_i}^t(r) \iff (\exists A \in \overline{GActions_i})\ A \in \beta_i^t(r). \tag{3.539}$$

Proof. This directly follows from Definition 3.5.22 of the synchronous agents joint protocol and the Byzantine action filter function (2.25). As no synchronous agents protocol can prescribe the empty set, whenever an agent i receives a $go(i)$ event during some round $t.5$ ($t \in \mathbb{N}$), it will perform some action $a \in \overline{Actions_i}$, as by t-coherence (in accordance to Definition 2.2.1) of the environment protocol's event sets (in accordance with Definition 2.2.4, specifically point 3), there can always only be one system event present for any agent during one round. Similarly, if $(\exists A \in \overline{GActions_i})\ A \in \beta_i^t(r)$ by definition of the Byzantine action filter, i must have gotten a $go(i)$. □

Lemma 3.5.31. *Lemma 3.5.21 does not hold for runs $r, r' \in R^\chi$ for $\chi \in \mathscr{E}^S$.*

Proof. This follows directly from Lemma 3.5.30. □

We will now proceed and formulate a version of the "Brain-in-the-Vat" Lemma 2.7.16 for the synchronous agents extension.

Lemma 3.5.32 (Synchronous Brain-in-the-Vat Lemma)**.** *Let $\mathcal{A} = [\![1; n]\!]$ be a set of agents with joint protocol $P^S = ({P_1}^S, \ldots, {P_n}^S) \in \mathscr{C}^S$, let $P_\epsilon \in \mathscr{C}_\epsilon$ be the protocol of the environment, let $r \in R^\chi$ be a run from the synchronous agents extension for $\chi \in \mathscr{E}^S$, let $i \in \mathcal{A}$ be an agent, let $t > 0$ be a timestamp and let $adj = [B_{t-1}; \ldots; B_0]$ be an adjustment of extent $t-1$ satisfying*

$$B_m = (\rho_1^m, \ldots, \rho_n^m)$$

for all $0 \leq m \leq t-1$ with

$$\rho_i^m = PFake_i^m \qquad \text{and} \qquad \text{for all } j \neq i \quad \rho_j^m \in \{CFreeze, BFreeze_j\}.$$

If the protocol P_ϵ makes

- *agent i gullible,*
- *every agent $j \neq i$ delayable and fallible if $\rho_j^m = BFreeze_j$ for some m,*
- *all remaining agents delayable,*

then each run $r' \in R\left(\tau^{S,B}_{P_\epsilon,P^S}, r, adj\right)$ satisfies the following properties:

1. *r' is $\tau^{S,B}_{P_\epsilon,P^S}$-transitional;*
2. *$(\forall m \leq t)\ r'_i(m) = r_i(m)$;*
3. *$(\forall m \leq t)(\forall j \neq i)\ \ r'_j(m) = r'_j(0)$;*
4. *$(i,1) \in Bad(r',1)$ and consequently $(i,m) \in Failed(r',m')$ for all $m' \geq m > 0$;*
5. *$\mathcal{A}(Failed(r'(t))) = \{i\} \cup \{j \neq i \mid (\exists m \leq t-1)\ \rho_j^m = BFreeze_j\}$;*
6. *$(\forall m < t)\ (\forall j \neq i)\ \ \beta^m_{\epsilon_j}(r') \subset \{fail(j)\}$.*
   More precisely, $\beta^m_{\epsilon_j}(r') = \varnothing$ iff $\rho_j^m = CFreeze$ and $\beta^m_{\epsilon_j}(r') = \{fail(j)\}$ iff $\rho_j^m = BFreeze_j$;
7. *$(\forall m < t)\ \beta^m_{\epsilon_i}(r') \setminus \beta^m_{f_i}(r') = \varnothing$;*
8. *$(\forall m < t)(\forall j \in \mathcal{A})\ \beta^m_j(r') = \varnothing$.*

Proof. The proof is analogous to the original Brain-in-the-Vat Lemma 2.7.16, since by Definition 2.7.8 of the intervention $PFake_i^t$, Definition 2.7.11 of *CFreeze* and Definition 2.7.12 of $BFreeze_i$ it holds that

$$\left(\forall r' \in R\left(\tau^{S,B}_{P_\epsilon,P^S}, r, adj\right)\right)(\forall j \in \mathcal{A})(\forall m \in \mathbb{N} \text{ s.t. } 0 \leq m < t)\ go(j) \notin \beta^m_{\epsilon_j}(r'). \tag{3.540}$$

Because the synchronous agents extension by Definition 3.5.23 does not restrict the environment protocol and because the synchronous agents event filter function by Definition 3.5.20 only additionally removes *go* events, which by Definition 2.7.8 of the interventions (from Definitions 2.7.11, 2.7.12 and 3.540) are irrelevant for such runs $r' \in R\left(\tau^{S,B}_{P_\epsilon,P^S}, r, adj\right)$ ((3.540) also makes the synchronous agents joint protocol irrelevant for this lemma), the proof of Lemma 2.7.16 applies for the synchronous agents extension as well. □

Local Introspection

Lemma 3.5.33. *For $adj \in Adjusts$ with extent $t-1$ from Lemma 3.5.32, some $o \in \overline{Actions} \sqcup \overline{Events}$, $\chi \in \mathscr{E}^S$, where $\chi = ((P_\epsilon, \mathscr{G}(0), \tau^{S,B}, R), P^S)$, where P_ϵ makes i gullible and all other agents $j \neq i$ delayable, a run $r \in R^\chi$, an interpreted system $I = (R^\chi, \pi)$, it holds that*

$$\left(\forall r' \in R\left(\tau^{S,B}_{P_\epsilon,P^S}, r, adj\right)\right)(\forall m \in \mathbb{N},\ s.t.\ m \leq t)\ (I, r', m) \not\models \overline{occurred}(o). \tag{3.541}$$

Proof. By Definitions 2.7.8, 2.7.11, 2.7.12 of the intervention $PFake_i^t$, *CFreeze* and $BFreeze_i$ it immediately follows that

$$\left(\forall r' \in R\left(\tau^S_{P_\epsilon,P^S}, r, adj\right)\right)(\forall m \leq t-1)(\forall j \in \mathcal{A}) \quad \overline{\beta}^m_{\epsilon_j}(r') = \varnothing \quad \wedge \quad \beta^m_j(r') = \varnothing. \quad (3.542)$$

By definition of $\overline{occurred}(o)$ from (2.84), (2.83) and (2.80), in order to satisfy $(I, r', t') \not\models \overline{occurred}(o)$, the following has to hold:

$$(\forall i \in \mathcal{A})(\forall m \leq t'-1)\ o \notin label^{-1}\left(\beta^m_j(r') \sqcup \overline{\beta}^m_{\epsilon_i}(r')\right) \quad (3.543)$$

Since (3.542) holds, it however becomes clear that for such runs $r' \in R\left(\tau^{S,B}_{P_\epsilon,P^S}, r, adj\right)$ (3.543) is always satisfied if $t' \leq t$. □

Lemma 3.5.34. *For an agent $i \in \mathcal{A}$, for a non-excluding agent-context $\chi \in \mathscr{E}^S$, where $\chi = \left((P_\epsilon, \mathscr{G}(0), \tau^{S,B}, \Psi), P^S\right)$, such that P_ϵ makes i gullible and all other agents $j \neq i$ delayable, for a χ-based interpreted system $I = (R^\chi, \pi)$, for a run $r \in R^\chi$, for a timestamp $t \in \mathbb{N}$, and for an action or event $o \in \overline{Actions} \sqcup \overline{Events}$,*

$$(I, r, t) \not\models K_i \overline{occurred}(o). \quad (3.544)$$

Proof. For $t = 0$ the statement is obvious. For $t > 0$ the Lemma follows from Lemma 3.5.32 (Synchronous Brain-in-the-Vat Lemma): If for $m \leq t$, $(I, r, m) \not\models \overline{occurred}(o)$, then by reflexivity of the possible worlds relation and semantics of the knowledge operator (from Definition 2.4.8), the statement holds.

Suppose for $m \leq t$, $(I, r, m) \models \overline{occurred}(o)$. For all runs $r' \in R\left(\tau^{S,B}_{P_\epsilon,P^S}, r, adj\right)$, where adj is from Lemma 3.5.32, by Property 2 of Lemma 3.5.32 it holds that

$$(\forall m' \leq t)\ r_i(m') = r'_i(m').$$

Hence by Definition 2.4.4 of the possible worlds relation

$$(\forall m' \leq t)\ r(m') \sim_i r'(m'),$$

from which we get (by universal instantiation of m')

$$r(m) \sim_i r'(m).$$

Additionally, from Lemma 3.5.33, it follows that

$$(I, r', m) \not\models \overline{occurred}(o).$$

By semantics of the knowledge operator (from Definition 2.4.8), in order for $(I, r, t) \models K_i \overline{occurred}(o)$ to be true, $\overline{occurred}(o)$ has to hold in all (i-)accessible states. However, since it does not hold at $r'(m)$ (for $m \leq t$), the Lemma follows. □

Lemma 3.5.35. *For an agent $i \in \mathcal{A}$ and a non-excluding agent-context $\chi = \left((P_\epsilon, \mathscr{G}(0), \tau^{S,B}, R), P^S\right)$ such that $\chi \in \mathscr{E}^S$ and P_ϵ makes i gullible and all other agents $j \neq i$ delayable, for a χ-based interpreted system $I = (R^\chi, \pi)$, for a run $r \in R^\chi$, and for a timestamp $t > 0$,*

$$(I, r, t) \not\models K_i correct_i. \quad (3.545)$$

Proof. Consider a run $r' \in R\left(\tau^{S,B}_{P_\epsilon,P^S}, r, adj\right)$, where adj with extent $t-1$ is from Lemma 3.5.32. By (2.78),

$$(I, r', t) \models correct_i \iff (i, t) \notin Failed\,(r', t).$$

Thus the statement follows directly from Properties 2 and 4 of Lemma 3.5.32, since (for $m \leq t$) $r(m) \sim_i r'(m)$ and $(I, r', t) \models \neg correct_i$. □

As for the fully Byzantine extension, it is sometimes possible for a synchronous agent to learn of its own defectiveness given the right circumstances.

Lemma 3.5.36. *For some agent $i \in \mathcal{A}$, an agent-context $\chi = \left((P_\epsilon, \mathscr{G}(0), \tau^{S,B}, R), P^S\right)$, it is possible that for some (weakly) χ-based interpreted system I, some (weakly) χ-consistent run r and some timestamp $t > 0$*

$$(I, r, t) \models K_i faulty_i. \tag{3.546}$$

Proof. As for the Byzantine extension, this is the case when for example there is a mismatch between actions recorded in the agent's local history and actions prescribed by the agent's protocol for the preceding local state. Formally, such a scenario can be described as follows: If it is the case that for some $t' < t$

$$r_i(t'+1) \cap \overline{Actions_i} \notin P_i\left(r_i(t')\right). \tag{3.547}$$

□

Definition 3.5.37. In addition to the propositions from Section 2.5, we further restrict the interpretation π to adhere to the following meaning for the atomic proposition $nvr(h)$ for $h \in \mathscr{G}$, given some interpreted system $I = (R', \pi)$ (for $R' \subseteq R$):

$$(I, r, t) \models nvr(h) \;\Leftrightarrow\; NVR\left(r(t)\right) = NVR\left(h\right). \tag{3.548}$$

Lemma 3.5.38. *A correct agent i with local history h_i in a synchronous agents context can infer from h_i the number of virtual rounds that have passed. Formally, for an agent context $\chi \in \mathscr{C}^S$, a χ-based interpreted system $I = (R^\chi, \pi)$, a run $r \in R^\chi$ and timestamp $t \in \mathbb{N}$*

$$(I, r, t) \models B_i nvr(r(t)). \tag{3.549}$$

Proof. By definition of the belief operator (2.132)

$$(I, r, t) \models B_i nvr(r(t)) \iff (I, r, t) \models K_i\Big(correct_i \to nvr(r(t))\Big).$$

By semantics of K_i (from Definition 2.4.8)

$$(\forall t' \in \mathbb{N})(\forall r' \in R^\chi)\left((r(t) \sim_i r'(t')) \to ((I, r', t') \models correct_i \to nvr(r(t)))\right)$$

Suppose by contradiction that there exists such a run r' and timestamp t' such that $r(t) \sim_i r'(t')$, i is correct in both runs and $NVR\left(r(t)\right) \neq NVR\left(r'(t')\right)$. By Definition 2.4.4 of the possible

worlds relation, it follows that $r_i(t) = r'_i(t')$. Since agent i is assumed to be correct in both runs, it follows

$$\begin{aligned}(\forall t''' < t')\ \beta_{f_i}^{t'''}(r') = \varnothing \\ (\forall t'' < t)\ \beta_{f_i}^{t''}(r) = \varnothing.\end{aligned} \tag{3.550}$$

Since $r_i(t) = r'_i(t')$ and 3.550 it follows that all updates of the agent histories in runs r and r' were due to correct actions and events. By Definition 3.5.22 of the synchronous joint protocol, Lemma 3.5.27 (an agent executes its protocol only during virtual rounds) and Lemma 3.5.29, it follows that the number of virtual rounds in both runs has to be the same, hence contradicting that $NVR(r(t)) \neq NVR(r'(t'))$. □

Global Introspection

Lemma 3.5.39. *For some agent $i \neq j$ $(i, j \in \mathcal{A})$, for a non-excluding agent context $\chi = \Big((P_\epsilon, \mathscr{G}(0), \tau^{S,B}, R), P^S\Big)$, where $\chi = \mathscr{E}^S$, such that P_ϵ makes agent i gullible and all other agents delayable, for a χ-based interpreted system $I = (R^\chi, \pi)$, for a run $r \in R^\chi$ and for a timestamp $t \in \mathbb{N}$,*

$$(I, r, t) \not\models K_i faulty_j. \tag{3.551}$$

Proof. Similarly to the asynchronous Byzantine extension for $t = 0$, the statement holds, as initially no agent can be faulty. For $t > 0$, the statement follows directly from the synchronous Brain-in-the-Vat Lemma 3.5.32, specifically from Properties 2 and 6 applied to interventions, where for all $m \leq t' - 1$ (where $t' - 1$ is the extent of the adjustment used in Lemma 3.5.32)

$$\rho_j^m = CFreeze. \tag{3.552}$$

□

Lemma 3.5.40. *For agent $i \neq j$, for a non-excluding agent context $\chi = \Big((P_\epsilon, \mathscr{G}(0), \tau^{S,B}, R), P^S\Big)$, where $\chi \in \mathscr{E}^S$, such that P_ϵ makes agent i gullible, agent j delayable and fallible and all other agents delayable, for a χ-based interpreted system $I = (R^\chi, \pi)$, for a run $r \in R^\chi$ and for a timestamp $t \in \mathbb{N}$,*

$$(I, r, t) \not\models K_i correct_j. \tag{3.553}$$

Proof. This proof is again analogous to the one for the asynchronous Byzantine extension. This Lemma simply follows from the synchronous Brain-in-the-Vat Lemma 3.5.32, specifically Properties 2 and 5 applied to the following interventions from the adjustment in Lemma 3.5.32, where for all $m \leq t' - 1$ (where $t' - 1$ is the extent of the adjustment used in Lemma 3.5.32):

$$\rho_k^m = \begin{cases} CFreeze & \text{if } k \notin \{i, j\} \\ BFreeze_j & \text{if } k = j. \end{cases} \tag{3.554}$$

□

3.5.7 Physical Multicast Communication

The **physical multicast communication** extension is quite similar to the multicast communication extension from Section 3.5.5. However, there is one major difference: Since the physical multicast communication extension is supposed to model hardware multicast communication systems, Byzantine agents cannot violate the multicast paradigm in such systems. Therefore, in addition to restricting the set of joint protocols, we also introduce an event filter, which removes fake send events that do not comply with the underlying multicast problem.

Since filter functions operate in the global context, we cannot use $Rec_X(\mu_k)$ from (3.513) in our filter definition.

Definition 3.5.41. For a set $X \subseteq \overline{GActions} \sqcup GEvents$, we denote the set of recipients for $\mu_k \in Msgs$ by

$$\begin{aligned} Rec^g_X(\mu_k) := \{j \mid (\exists i' \in \mathcal{A})(\exists t' \in \mathbb{N})\ \ gsend(i', j, \mu, id(i', j, \mu, k, t')) \in X \vee \\ (\exists A \in \overline{GActions} \sqcup \{\text{📅}\})(\exists i \in \mathcal{A})(\exists t \in \mathbb{N})\ fake\,(i, gsend(i, j, \mu, id(i, j, \mu, k, t)) \mapsto A) \in X\Big\}. \end{aligned} \tag{3.555}$$

Definition 3.5.42 (Physical Multicast Communication Event Filter)**.** For some global history $h \in \mathscr{G}$, sets $X_\epsilon \subseteq GEvents$, $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$ and a multicast problem Ch, we define the physical multicast communication event filter as follows (note that we use the abbreviation $X_{\epsilon_i} = X_\epsilon \cap GEvents_i$ from Definition 2.2.31):

$$\begin{aligned} &filter^{PMC_{Ch}}_\epsilon\,(h, X_\epsilon, X_1, \ldots, X_n) := X_\epsilon \setminus \\ &\{fake\,(i, gsend(i, j, \mu, id(i, j, \mu, k, t)) \mapsto A) \in X_\epsilon \mid \\ &\Big(active(i, X_\epsilon)\ \wedge\ Rec^g_{X_{\epsilon_i} \sqcup X_i}(\mu_k) \notin Mc_i\Big)\ \vee \\ &\Big(passive(i, X_\epsilon)\ \wedge\ Rec^g_{X_{\epsilon_i}}(\mu_k) \notin Mc_i\Big)\Big\}. \end{aligned} \tag{3.556}$$

Definition 3.5.43. For a multicast communication problem Ch, we denote by $\tau^{PMC_{Ch},B}$ the transition template, where the Byzantine action filters and the physical multicast communication event filter (3.556) is used.

Definition 3.5.44. For a multicast communication problem Ch, we denote by

$$\mathscr{E}^{PMC_{Ch}} := \Big(\mathscr{C}_\epsilon \times \mathscr{C}^{MC_{Ch}}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^{PMC_{Ch},B}, R\Big) \tag{3.557}$$

the **physical multicast communication** extension, where in $\tau^{PMC_{Ch},B}$ the physical multicast communication event filter and (for all $i \in \mathcal{A}$) the Byzantine action filter is used.

Definition 3.5.45. The **physical broadcast communication** extension $\mathscr{E}^{PBC}$ is a multicast communication extension $\mathscr{E}^{PMC_{BCh}}$ for the special broadcast communication problem

$$BCh := (\underbrace{\{\mathcal{A}\}, \ldots, \{\mathcal{A}\}}_{n}). \tag{3.558}$$

Lemma 3.5.46. *The physical multicast communication event filter is generally not monotonic for the domain PD^{t-coh}_ϵ, $2^{\overline{GActions}_1}$, ..., $2^{\overline{GActions}_n}$.*

Proof. Counterexample:
Given a multicast communication problem, where agent 1 can only send messages to agent 2 and 3, but all other agents are unrestricted

$$Ch = (\{\{2\},\{3\}\}, 2^{\mathcal{A}} \setminus \{\varnothing\}, \ldots, 2^{\mathcal{A}} \setminus \{\varnothing\}), \tag{3.559}$$

given sets $X_1 \subseteq \overline{GActions}_1$, ..., $X_n \subseteq \overline{GActions}_n$, actions $A, A' \in \overline{GActions}_1 \sqcup \{\boxminus\}$, agents $1,2,3 \in \mathcal{A}$, messages $\mu, \mu' \in Msgs$, global history $h \in \mathscr{G}$ and timestamp $t = |h|$ s.t.

$$\begin{aligned} X_\epsilon =& \{fake\,(1, gsend(1,2,\mu, id(1,2,\mu,1,t)) \mapsto A), \\ & fake\,(1, gsend(1,3,\mu, id(1,3,\mu,1,t)) \mapsto A')\} \\ X'_\epsilon =& \{fake\,(1, gsend(1,2,\mu, id(1,2,\mu,1,t)) \mapsto A)\} \end{aligned} \tag{3.560}$$

it obviously holds that $X'_\epsilon \subseteq X_\epsilon$, however we get that

$$filter_\epsilon^{PMC_{Ch}}(h, X_\epsilon, X_1, \ldots, X_n) = \varnothing \tag{3.561}$$

$$filter_\epsilon^{PMC_{Ch}}(h, X'_\epsilon, X_1, \ldots, X_n) = \{fake\,(1, gsend(1,2,\mu, id(1,2,\mu,1,t)) \mapsto A)\} \tag{3.562}$$

and therefore

$$filter_\epsilon^{PMC_{Ch}}(h, X_\epsilon, X_1, \ldots, X_n) \subset filter_\epsilon^{PMC_{Ch}}(h, X'_\epsilon, X_1, \ldots, X_n). \tag{3.563}$$

Since by (3.560) X_ϵ (and X'_ϵ) is t-coherent, the Lemma follows. □

Lemma 3.5.47. *The physical broadcast communication event filter is monotonic for the domain* PD_ϵ^{t-coh}, $2^{\overline{GActions}_1}$, ..., $2^{\overline{GActions}_n}$.

Proof. We conduct the proof by induction.
Induction Hypothesis: Suppose for some global history $h \in \mathscr{G}$, sets $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \in 2^{\overline{GActions}_1}$, ..., $X_n \in 2^{\overline{GActions}_n}$, $X'_\epsilon \subseteq X_\epsilon$, $X'_1 \subseteq X_1$, ..., $X'_n \subseteq X_n$ it holds that

$$filter_\epsilon^{PBC}(h, X'_\epsilon, X'_1, \ldots, X'_n) \subseteq filter_\epsilon^{PBC}(h, X_\epsilon, X_1, \ldots, X_n). \tag{3.564}$$

Base Case: For $X'_\epsilon = X_\epsilon$, $X'_1 = X_1$, ..., $X'_n = X_n$ the statement is trivially satisfied.
Induction Step: We split this into two parts. First we examine the filter w.r.t. removing an element from the event set and then we will do the same for some (arbitrary) action set. Suppose the induction hypothesis holds for X'_ϵ. We examine whether the induction hypothesis also holds for $X'_\epsilon \setminus \{E\}$ (where $E \in X'_\epsilon$). There are two cases:

1. $(\forall i,j \in \mathcal{A})(\forall \mu \in Msgs)(\forall k \in \mathbb{N})(\forall t \in \mathbb{N})(\forall A \in \overline{GActions}_i \sqcup \{\boxminus\})$
   $E \neq fake\,(i, gsend(i,j,\mu, id(i,j,\mu,k,t)) \mapsto A)$

   a) If $E = go(i)$ for some $i \in \mathcal{A}$
   It follows that

   $$Rec^g_{X'_{\epsilon_i} \setminus \{E\}}(\mu_k) \subseteq Rec^g_{X'_{\epsilon_i} \sqcup X'_i}(\mu_k). \tag{3.565}$$

   Therefore, by Definition 3.5.41 and 3.5.42

   $$\begin{aligned} & filter_\epsilon^{PBC}(h, X'_\epsilon \setminus \{E\}, X'_1, \ldots, X'_n) \subseteq filter_\epsilon^{PBC}(h, X'_\epsilon, X'_1, \ldots, X'_n) \setminus \{E\} \subseteq \\ & filter_\epsilon^{PBC}(h, X'_\epsilon, X'_1, \ldots, X'_n) \subseteq filter_\epsilon^{PBC}(h, X_\epsilon, X_1, \ldots, X_n). \end{aligned} \tag{3.566}$$

b) If $E \neq go(i)$ for some $i \in \mathcal{A}$
Note that since X'_ϵ is t-coherent, there can only ever be one of the system events present for each agent. Thus, it cannot happen that after removing some $sleep\,(i)$ or $hibernate\,(i)$ from X'_ϵ, there is a $go(i)$ left. By (3.555) and (3.556), for any $\mu \in Msgs$ and $k \in \mathbb{N}$ this leads to

$$Rec^g_{X'_{\epsilon_i}\setminus\{E\}}(\mu_k) = Rec^g_{X'_{\epsilon_i}}(\mu_k) \tag{3.567}$$

from which also

$$Rec^g_{(X'_{\epsilon_i}\setminus\{E\})\sqcup X_j}(\mu_k) = Rec^g_{X'_{\epsilon_i}\sqcup X_j}(\mu_k) \tag{3.568}$$

follows for any $j \in \mathcal{A}$. Thus, by (3.568) and the induction hypothesis (3.564)

$$\begin{aligned} &filter^{PBC}_\epsilon\,(h, X'_\epsilon \setminus \{E\}, X'_1, \ldots, X'_n) \;=\; filter^{PBC}_\epsilon\,(h, X'_\epsilon, X'_1, \ldots, X'_n) \setminus \{E\} \;\subseteq \\ &filter^{PBC}_\epsilon\,(h, X'_\epsilon, X'_1, \ldots, X'_n) \;\subseteq\; filter^{PBC}_\epsilon\,(h, X_\epsilon, X_1, \ldots, X_n)\,. \end{aligned} \tag{3.569}$$

2. $(\exists i', j \in \mathcal{A})(\exists \mu \in Msgs)(\exists k \in \mathbb{N})(\exists t \in \mathbb{N})(\exists A \in \overline{GActions_i} \sqcup \{\text{📅}\})$
$E = fake\,(i', gsend(i', j, \mu, id(i, j, \mu, k, t)) \mapsto A)$
By existential instantiation, we get that

$$E = fake\,(i, gsend(i, j', \mu', id(i', j', \mu', k', t')) \mapsto A'), \tag{3.570}$$

which implies that $Rec^g_{X'_{\epsilon_i}\setminus\{E\}}(\mu'_{k'}) \subset Rec^g_{X'_{\epsilon_i}}(\mu'_{k'})$. Hence we further distinguish between the cases:

a) $Rec^g_{(X'_{\epsilon_i}\setminus\{E\})\sqcup X_i}(\mu'_{k'}) = Rec^g_{X'_{\epsilon_i}\sqcup X_i}(\mu'_{k'})$
If $active(i', X'_\epsilon \setminus \{E\})$ is true, then by (3.555), since $E \in X'_\epsilon$ and by the induction hypothesis (3.564) we get that

$$\begin{aligned} &filter^{PBC}_\epsilon\,(h, X'_\epsilon \setminus \{E\}, X'_1, \ldots, X'_n) = filter^{PBC}_\epsilon\,(h, X'_\epsilon, X'_1, \ldots, X'_n) \setminus \{E\} \subset \\ &filter^{PBC}_\epsilon\,(h, X'_\epsilon, X'_1, \ldots, X'_n) \subseteq filter^{PBC}_\epsilon\,(h, X_\epsilon, X_1, \ldots, X_n)\,. \end{aligned} \tag{3.571}$$

Otherwise if $active(i, X'_\epsilon \setminus \{E\})$ is false or equivalently $passive(i, X'_\epsilon \setminus \{E\})$ is true, by definition of the broadcast communication problem (3.558) and the induction hypothesis (3.564) we get

$$\begin{aligned} &filter^{PBC}_\epsilon\,(h, X'_\epsilon \setminus \{E\}, X'_1, \ldots, X'_n) = (filter^{PBC}_\epsilon\,(h, X'_\epsilon, X'_1, \ldots, X'_n) \setminus \{E\}) \setminus \\ &\{fake\,(i, gsend(i, j, \mu', id(i, j, \mu', k', t)) \mapsto A) \in X'_\epsilon \setminus \{E\} \mid \\ &j \in \mathcal{A} \wedge t \in \mathbb{N} \wedge A \in \{\text{📅}\} \sqcup \overline{GActions_i}\} \subseteq \\ &filter^{PBC}_\epsilon\,(h, X'_\epsilon, X'_1, \ldots, X'_n) \subseteq filter^{PBC}_\epsilon\,(h, X_\epsilon, X_1, \ldots, X_n)\,. \end{aligned} \tag{3.572}$$

b) $Rec^g_{(X'_{\epsilon_i}\setminus\{E\})\sqcup X_i}(\mu'_{k'}) \subset Rec^g_{X'_{\epsilon_i}\sqcup X_i}(\mu'_{k'})$
By definition of the broadcast communication problem (3.558) and the induction

hypothesis (3.564) we get

$$\begin{aligned}&filter_\epsilon^{PBC}\left(h, X'_\epsilon \setminus \{E\}, X'_1, \ldots, X'_n\right) = \left(filter_\epsilon^{PBC}\left(h, X'_\epsilon, X'_1, \ldots, X'_n\right) \setminus \{E\}\right) \setminus \\ &\{fake\left(i, gsend(i, j, \mu', id(i, j, \mu', k', t)\right) \mapsto A) \in X'_\epsilon \setminus \{E\} \mid \\ &j \in \mathcal{A} \wedge t \in \mathbb{N} \wedge A \in \{\text{📅}\} \sqcup \overline{GActions_i}\Big\} \subseteq \\ &filter_\epsilon^{PBC}\left(h, X'_\epsilon, X'_1, \ldots, X'_n\right) \subseteq filter_\epsilon^{PBC}\left(h, X_\epsilon, X_1, \ldots, X_n\right).\end{aligned} \tag{3.573}$$

Lastly we will examine whether the induction hypothesis still holds after removing an action A from some action set X'_i for some $i \in \mathcal{A}$. We distinguish between two cases:

1. $(\forall j \in \mathcal{A})(\forall \mu \in Msgs)(\forall k \in \mathbb{N})(\forall t \in \mathbb{N})\ A \neq gsend(i, j, \mu, id(i, j, \mu, k, t))$
   By (3.555)
   $$Rec^g_{X'_{\epsilon_i} \sqcup (X'_i \setminus \{A\})}(\mu_k) = Rec^g_{X'_{\epsilon_i} \sqcup X'_i}(\mu_k), \tag{3.574}$$
   hence by (3.556) and the induction hypothesis (3.564) it follows that
   $$\begin{aligned}&filter_\epsilon^{PBC}\left(h, X'_\epsilon, X'_1, \ldots, X'_{i-1}, X'_i \setminus \{A\}, X'_{i+1}, \ldots, X'_n\right) = \\ &filter_\epsilon^{PBC}\left(h, X'_\epsilon, X'_1, \ldots, X'_n\right) \subseteq filter_\epsilon^{PBC}\left(h, X_\epsilon, X_1, \ldots, X_n\right).\end{aligned} \tag{3.575}$$

2. $(\exists j \in \mathcal{A})(\exists \mu \in Msgs)(\exists k \in \mathbb{N})(\exists t \in \mathbb{N})\ A = gsend(i, j, \mu, id(i, j, \mu, k, t))$
   By existential instantiation we get that $A = gsend(i, j', \mu', id(i, j', \mu', k', t'))$. We distinguish between the cases

   a) $Rec^g_{X'_{\epsilon_i} \sqcup (X'_i \setminus \{A\})}(\mu'_{k'}) = Rec^g_{X'_{\epsilon_i} \sqcup X'_i}(\mu'_{k'})$
   Similarly as in the case above by (3.556) and the induction hypothesis (3.564) it follows that
   $$\begin{aligned}&filter_\epsilon^{PBC}\left(h, X'_\epsilon, X'_1, \ldots, X'_{i-1}, X'_i \setminus \{A\}, X'_{i+1}, \ldots, X'_n\right) = \\ &filter_\epsilon^{PBC}\left(h, X'_\epsilon, X'_1, \ldots, X'_n\right) \subseteq filter_\epsilon^{PBC}\left(h, X_\epsilon, X_1, \ldots, X_n\right).\end{aligned} \tag{3.576}$$

   b) $Rec^g_{X'_{\epsilon_i} \sqcup (X'_i \setminus \{A\})}(\mu'_{k'}) \subset Rec^g_{X'_{\epsilon_i} \sqcup X'_i}(\mu'_{k'})$
   Since by (3.555)
   $$Rec^g_{X'_{\epsilon_i}}(\mu'_{k'}) \subseteq Rec^g_{X'_{\epsilon_i} \sqcup (X'_i \setminus \{A\})}(\mu'_{k'}) \tag{3.577}$$
   it follows that no matter whether $active(i, X'_\epsilon)$ or $passive(i, X'_\epsilon)$ is true, it holds that (by (3.556) and (3.564))
   $$\begin{aligned}&filter_\epsilon^{PBC}\left(h, X'_\epsilon, X'_1, \ldots, X'_{i-1}, X'_i \setminus \{A\}, X'_{i+1}, \ldots, X'_n\right) \subseteq \\ &filter_\epsilon^{PBC}\left(h, X'_\epsilon, X'_1, \ldots, X'_n\right) \subseteq filter_\epsilon^{PBC}\left(h, X_\epsilon, X_1, \ldots, X_n\right).\end{aligned} \tag{3.578}$$

□

Lemma 3.5.48. *The Physical Multicast event filter is idempotent for the domain* PD_ϵ^{t-coh}, $2^{\overline{GActions_1}}$, ..., $2^{\overline{GActions_n}}$.

Proof. For some history $h \in \mathscr{G}$, $X_\epsilon \in PD_\epsilon^{t-coh}$, $X_1 \in 2^{\overline{GActions_1}}$, ..., $X_n \in 2^{\overline{GActions_n}}$, let

$$filter_\epsilon^{PMC_{Ch}}(h, X_\epsilon, X_1, \ldots, X_n) = Y_\epsilon. \tag{3.579}$$

Suppose by contradiction that

$$filter_\epsilon^{PBC}(h, Y_\epsilon, X_1, \ldots, X_n) \subset Y_\epsilon. \tag{3.580}$$

This implies that for some agents $i, j \in \mathcal{A}$, some message $\mu \in Msgs$, some $k, t \in \mathbb{N}$, some action $A \in \overline{GActions_i} \sqcup \{\boxminus\}$

$$fake(i, gsend(i, j, \mu, id(i, j, \mu, k, t)) \mapsto A) \in Y_\epsilon \tag{3.581}$$

such that

$$Rec_{Y_\epsilon}^g(\mu_k) \notin Mc_i. \tag{3.582}$$

Since $Y_\epsilon \subseteq X_\epsilon$, it follows that also

$$fake(i, gsend(i, j, \mu, id(i, j, \mu, k, t)) \mapsto A) \in X_\epsilon. \tag{3.583}$$

If $Rec_{X_\epsilon}^g(\mu_k) \in Mc_i$, then by (3.556)

$$Rec_{Y_\epsilon}^g(\mu_k) \in Mc_i, \tag{3.584}$$

which however would contradict (3.582). Otherwise if $Rec_{X_\epsilon}^g(\mu_k) \notin Mc_i$ by (3.582), we would get that

$$fake(i, gsend(i, j, \mu, id(i, j, \mu, k, t)) \mapsto A) \notin Y_\epsilon, \tag{3.585}$$

which contradicts (3.583). Thus the Lemma follows. □

3.5.8 Coordinated Agents

Coordinated agents describes a system, where actions of certain groups of agents must be coordinated, i.e., the group of agents woken up by the environment at some round must satisfy certain criteria. These criteria can generally depend on time, so that the coordination could be dynamically adjusted. However, coordination is not changed based on the local or global state.

Definition 3.5.49. A **coordinated agents problem** is described as a *FailGo* function FG that, to each pair of a timestamp and a set of agents scheduled to act at this timestamp, assigns a collection of possible Byzantine culprits

$$FG\colon \mathbb{N} \times 2^{\mathcal{A}} \to 2^{2^{\mathcal{A}}}. \tag{3.586}$$

Let us define

$$Go_X = \{i \mid go(i) \in X\} \qquad \text{and} \qquad Fail_X = \{i \mid fail(i) \in X\}. \tag{3.587}$$

Since the coordinated agents problem FG could be such that it actually disallows rounds in which nothing happens, we cannot implement the coordinated agents property just with a specialized event filter. We therefore introduce the Coordinated Agents environment protocol.

Definition 3.5.50. For a coordinated agents problem FG, we define the set of Coordinated Agents environment protocols as

$$\mathscr{C}_\epsilon^{CA_{FG}} := \{P \in \mathscr{C}_\epsilon \mid (\forall t \in \mathbb{N})(\forall X_\epsilon \in P(t))(\exists Y \in FG(t, Go_{X_\epsilon}))\, Fail_{X_\epsilon} \supseteq Y\}\,. \tag{3.588}$$

Without loss of generality, we can require $FG(t, G)$, which is a subset of $2^{\mathcal{A}}$, to consist of pairwise incomparable subsets of $\mathcal{A}$ for each $t \in \mathbb{N}$ and each $G \subseteq \mathcal{A}$.

This means that for each timestamp t and each set of agents scheduled to act by the environment, the function FG gives a range of possibilities for assigning "*fall guys*" in case the scheduling is faulty. If it happens that when the scheduling is indeed faulty, the environment protocol does not supply a superset of the necessary *fall guys*, the environment protocol removes all go events, thus preventing anyone from acting.

Remark 3.5.51. The fact that a group G of agents is allowed to act together at timestamp t is represented by $FG(t, G) = \{\varnothing\}$, because $\varnothing \subseteq Fail_X$, whichever set $Fail_X$ of agents the adversary chooses to fail. W.l.o.g. if $\varnothing \in FG(t, G)$, then $FG(t, G) = \{\varnothing\}$.

Remark 3.5.52. All cases where $\varnothing \notin FG(t, G)$ correspond to violations of the coordination conditions. They demand $Fail_X$ to be non-empty and specify who should be considered at fault. For instance, suppose $\mathcal{A} = \{1, 2, 3\}$ and agents 1 and 2 are supposed to act either together or not at all. If

$$FG(t, \{1\}) = FG(t, \{1, 3\}) = \{\{1\}\} \qquad \text{and} \qquad FG(t, \{2\}) = FG(t, \{2, 3\}) = \{\{2\}\}$$

for all $t \in \mathbb{N}$, then the agent that acts unilaterally is at fault. If

$$FG(t, \{1\}) = FG(t, \{1, 3\}) = \{\{2\}\} \qquad \text{and} \qquad FG(t, \{2\}) = FG(t, \{2, 3\}) = \{\{1\}\}$$

for all $t \in \mathbb{N}$, then the agent that fails to join the action is at fault. If

$$FG(t, \{1\}) = FG(t, \{1, 3\}) = \{\{1\}, \{2\}\} \qquad \text{and} \qquad FG(t, \{2\}) = FG(t, \{2, 3\}) = \{\{1\}, \{2\}\}$$

for all $t \in \mathbb{N}$, then the environment has a choice whether to blame 1 or 2 but must blame at least one of them. If

$$FG(t, \{1\}) = FG(t, \{1, 3\}) = \{\{3\}\} \qquad \text{and} \qquad FG(t, \{2\}) = FG(t, \{2, 3\}) = \{\{3\}\}$$

for all $t \in \mathbb{N}$, then agent 3 is considered at fault for the failure of 1 and 2 to coordinate.

Definition 3.5.53. For a coordinated agents problem FG, the **coordinated agents extension** is

$$\mathscr{E}^{CA_{FG}} := \left(\mathscr{C}_\epsilon^{CA_{FG}} \times \mathscr{C}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^B, R\right). \tag{3.589}$$

Remark 3.5.54. The coordinated agents extension is more expressive than the basic framework. In particular, the basic framework is an instance of the coordinated agents problem for

$$FG(t, G) = \{\varnothing\}$$

for each $t \in \mathbb{N}$ and each $G \subseteq \mathcal{A}$, in which case the condition $Fail_X \supseteq \varnothing$ is trivially satisfied by any set X of events.

Remark 3.5.55. While the primary purpose of this extension is coordination among agents, it can also be used to prevent simultaneous actions. For instance, setting for all $t \in \mathbb{N}$

$$FG(t, G) = \{G\}$$

for all $G \subseteq \mathcal{A}$ such that $|G| > 1$ restricts correct action to one agent acting per round. If more than one agent acts, then all acting agents are considered to be at fault. Alternatively, one can set for all $t \in \mathbb{N}$

$$FG(t, G) = \{G \setminus \{i\} \mid i \in G\}$$

for all $G \subseteq \mathcal{A}$ such that $|G| > 1$, which sets the same restriction of one agent per round but allows to consider one of the acting agents be correct.

Remark 3.5.56. Synchronous agents can also be modeled as a particular coordinated agents problem with

$$FG_{sync}(t, G) = \{\mathcal{A} \setminus G\}$$

for each $t \in \mathbb{N}$ and each $G \subseteq \mathcal{A}$.

3.6 Combined Extensions

In this section we present extensions that were constructed by combining some previously defined extensions.

3.6.1 Rendezvous Communication

Rendezvous communication refers to a system, where sender and receiver must synchronize themselves. This implies that a message sent by a correct agent must be received in the same round. We only require the rendezvous property to hold for certain channels $C \subseteq \mathcal{A}^2$. Therefore we define the rendezvous communication extension as combination of $\mathscr{E}^{SC_C}$ and $\mathscr{E}^{RC_C}$ (the synchronous communication and reliable communication extension).

Lemma 3.6.1. *The extensions $\mathscr{E}^{RC_C}$ and $\mathscr{E}^{SC_C}$ are compatible w.r.t. composition $RC_C \circ SC_C$, thus $\mathscr{E}^{RC_C \circ SC_C}$ is a valid extension.*

Proof. Since obviously

$$\mathscr{C}_\epsilon \cap \mathscr{C}_\epsilon^{SC_C} \neq \varnothing \tag{3.590}$$

$$\mathscr{C} \cap \mathscr{C} \neq \varnothing \tag{3.591}$$

$$(2^{\mathscr{G}(0)} \setminus \varnothing) \cap (2^{\mathscr{G}(0)} \setminus \varnothing) \neq \varnothing, \tag{3.592}$$

we only need to examine, whether for

$$\mathscr{E}^{RC_C \circ SC_C} = \left(\mathscr{C}_\epsilon^{SC_C} \times \mathscr{C}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^{N,N}, EDel_C\right) \tag{3.593}$$

there exists an agent context $\chi \in \mathscr{E}^{RC_C \circ SC_C}$ such that $R^\chi \neq \varnothing$. Let $\chi = \left((P'_\epsilon, \mathscr{G}(0), \tau^{N,N}, EDel_C), P'\right)$, where P'_ϵ only produces the set containing the empty set and P' produces the set containing the empty set for every agent. By (3.509) and (3.498) (the synchronous communication environment protocol is defined as a special case of the time-bounded communication environment

protocol, which only restricts environment protocols w.r.t. allowing only correct receive events, when their corresponding send event has happened 'recent enough') it follows that $P'_\epsilon \in \mathscr{C}_\epsilon^{SC_C}$. Therefore

$$(\forall t \in \mathbb{N})\, P'_\epsilon(t) = \{\varnothing\} \tag{3.594}$$

$$(\forall h \in \mathscr{G})\, P'(h) = (\{\varnothing\}, \ldots, \{\varnothing\}). \tag{3.595}$$

Since this agent context can produce the empty run $r \in R$, where

$$(\forall t \in \mathbb{N})\; r_\epsilon(t) = [\varnothing, \underbrace{\ldots}_{t-2}, \varnothing], \tag{3.596}$$

$\mathscr{E}^{RC_C \circ SC_C}$ is indeed a valid extension. □

Lemma 3.6.2. *The extension $\mathscr{E}^{RC_C \circ SC_C}$ satisfies the liveness property of $\mathscr{E}^{RC_C}$.*

Proof. This follows from Lemma 3.3.10. □

Lemma 3.6.3. *The extension $\mathscr{E}^{RC_C \circ SC_C}$ satisfies the safety property S^{SC_C}.*

Proof. This follows from Corollary 3.3.29. □

Definition 3.6.4. For a set of channels $C \subseteq \mathcal{A}^2$, we denote by

$$\mathscr{E}^{dRdV_C} = \left(\mathscr{C}_\epsilon^{SC_C} \times \mathscr{C}, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^{N,N}, EDel_C\right) \tag{3.597}$$

the **rendezvous communication** extension.

3.6.2 Lock-step Synchronous Agents

In the **lock-step synchronous agents** extension agents are synchronous and communication is synchronous and reliable. Moreover communication is broadcast (physical or not). To implement this extension, we only have to combine extensions that we have already introduced (*synchronous agents, synchronous communication, reliable communication, broadcast communication, asynchronous Byzantine agents*). Hence our mix of extensions looks like this: $1 \times$ **Adm**, $1 \times$ **EnvJP$_{\mathbf{DC}}$**, $1 \times$ **JP − AFB**, $2 \times$ **EvFJP − AFB**.

Following our established extension combination guide from Section 3.3.6, the first (non-trivial) step is drawing a dependence graph of the event filters from $\mathscr{E}^S$ (Definition 3.5.23) and $\mathscr{E}^B$ (Definition 3.5.1). In order to help us with that, we construct a small table based on the definitions of the involved filters (2.24), (3.519) showing what both filters depend on and which events they remove.

Table 3.2: filter dependencies

filter	**dependency**	**removal**
$filter_\epsilon^S$	$go(i), sleep(i), hibernate(i)$	$go(i)$
$filter_\epsilon^B$	$go(i), gsend(i,j,\mu,id), fake\,(i, gsend(i,j,\mu,id) \mapsto A)$	$grecv(j,i,\mu,id)$

Table 3.2 reveals that $filter_\epsilon^B$ depends on $go(i)$ events, which $filter_\epsilon^S$ removes. Therefore we have a dependence relation from $filter_\epsilon^B$ to $filter_\epsilon^S$. $filter_\epsilon^B$ removes only correct receive

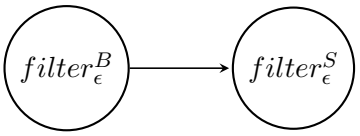

Figure 3.1: dependence graph for $filter_\epsilon^B$ and $filter_\epsilon^S$

events $grecv(j, i, \mu, id)$. $filter_\epsilon^S$ however, is independent of such events, hence there is no dependence relation from $filter_\epsilon^S$ to $filter_\epsilon^B$.

Figure 3.1 shows the final dependence graph. Since there is no circular dependence, we can directly use the composition order given by the graph. This gives us

$$\mathscr{E}^{B\circ S} = (\mathscr{C}_\epsilon \times \mathscr{C}^S, 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^{B\circ S,B}, R), \tag{3.598}$$

where in $\tau^{B\circ S,B}$ the event filter is $filter_\epsilon^{B\circ S}$ and the action filters result in $filter_i^B$ for all $i \in \mathcal{A}$ (by idempotence by Lemma 3.1.22 of the Byzantine action filter function). Following the rest of the extension combination guide finally leads to

$$\mathscr{E}^{B\circ S\circ BC\circ SC_{\mathcal{A}^2}\circ RC_{\mathcal{A}^2}} = \left(\mathscr{C}_\epsilon^{SC_{\mathcal{A}^2}} \times (\mathscr{C}^{MC_{BCh}} \cap \mathscr{C}^S), 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^{B\circ S,B}, EDel_{\mathcal{A}^2}\right). \tag{3.599}$$

Lemma 3.6.5. *The extensions $\mathscr{E}^B$, $\mathscr{E}^S$, $\mathscr{E}^{SC_{\mathcal{A}^2}}$, $\mathscr{E}^{RC_{\mathcal{A}^2}}$ and $\mathscr{E}^{BC}$ are compatible (w.r.t. the composition $B \circ S \circ BC \circ SC_{\mathcal{A}^2} \circ RC_{\mathcal{A}^2}$).*

Proof. The only condition from Definition 3.2.2 that does not trivially follow from the definition of the extensions in question is whether there exists an agent context χ, such that $\chi \in \mathscr{E}^{B\circ S\circ BC\circ SC_{\mathcal{A}^2}\circ RC_{\mathcal{A}^2}}$. Such a χ however can be easily constructed. Let $\chi = \left((P'_\epsilon, \mathscr{G}(0), \tau^{B\circ S,B}, EDel_{\mathcal{A}^2}), P'\right)$, where P'_ϵ only produces the set containing the empty set and P' for every agent produces the set containing the set that only contains the action ⏱.

$$(\forall t \in \mathbb{N})\, P'_\epsilon(t) = \{\varnothing\} \tag{3.600}$$

$$(\forall h \in \mathscr{G})\, P'(h) = (\{\{⏱\}\}, \ldots, \{\{⏱\}\}). \tag{3.601}$$

It is easy to see that this agent context is part of the extension $\mathscr{E}^{B\circ S\circ BC\circ SC_{\mathcal{A}^2}\circ RC_{\mathcal{A}^2}}$, as

$$P'_\epsilon \in \mathscr{C}_\epsilon^{SC_{\mathcal{A}^2}} \quad \text{and} \quad P' \in (\mathscr{C}^{MC_{BCh}} \cap \mathscr{C}^S). \tag{3.602}$$

□

Lemma 3.6.6. *The extension $\mathscr{E}^{B\circ S\circ BC\circ SC_{\mathcal{A}^2}\circ RC_{\mathcal{A}^2}}$ satisfies all safety properties of its constituent extensions.*

Proof. This results from its construction, as we were strictly following the extension combination guide from Section 3.3.6. □

Finally, after having proved that the resulting extension $\mathscr{E}^{B\circ S\circ BC\circ SC_{\mathcal{A}^2}\circ RC_{\mathcal{A}^2}}$ satisfies all desired properties, we can define it as $\mathscr{E}^{LSS}$.

Definition 3.6.7. We denote by $\mathscr{E}^{LSS} = \left(\mathscr{C}_\epsilon^{SC_{\mathcal{A}^2}} \times (\mathscr{C}^{MC_{BCh}} \cap \mathscr{C}^S), 2^{\mathscr{G}(0)} \setminus \{\varnothing\}, \tau^{B\circ S,B}, EDel_{\mathcal{A}^2}\right)$ the **lock-step synchronous agents** extension.

We will now add a few Lemmas about properties, which the lock-step synchronous agents extension inherits from the synchronous agents extension.

Lemma 3.6.8. *An agent i in a lock-step synchronous agents context executes its protocol only during virtual rounds, i.e., $go(i) \in \beta_{g_i}^t(r)$ iff $t.5$ is a virtual round.*

Proof. Follows from Lemma 3.5.27 for the synchronous agents extension, as Lemma 3.5.27 describes a property of S^S that by Lemma 3.6.6, $\mathscr{E}^{LSS}$ satisfies. □

Lemma 3.6.9. *For a correct agent i, a $\tau^{B\circ S,B}_{P_\epsilon^{SC_{\mathcal{A}^2}},P^{SMC_{BCh}}}$-transitional run r (where $P_\epsilon^{SC_{\mathcal{A}^2}} \in \mathscr{C}_\epsilon^{SC_{\mathcal{A}^2}}$ and $P^{SMC_{BCh}} \in \mathscr{C}^S \cap \mathscr{C}^{MC_{BCh}}$), some timestamp $t' \geq 1$, agent i's local history $r_i(t') = h_i = [\lambda_m, \ldots, \lambda_1, \lambda_0]$ (given the global history $h = r(t') \in \mathscr{G}$) and some round $(t-1).5$ ($t' \geq t \geq 1$), there exists some $a \in \overline{Actions_i}$ such that $a \in \lambda_{k_t}$ where $\lambda_{k_t} = \sigma(\beta_{\epsilon_i}^{t-1}(r) \sqcup \beta_i^{t-1}(r))$ if and only if $(t-1).5$ is a virtual round.*

Proof. This again follows from Lemma 3.5.29 for the synchronous agents extension, as the statement of this lemma is a safety property of $\mathscr{E}^S$ and by Lemma 3.6.6, $\mathscr{E}^{LSS}$ satisfies S^S. □

Lemma 3.6.10. *For any agent $i \in \mathcal{A}$, any run $r \in R^\chi$, where $\chi \in \mathscr{E}^{LSS}$ and any timestamp $t \in \mathbb{N}$ it holds that*

$$go(i) \in \beta_{g_i}^t(r) \iff (\exists A \in \overline{GActions_i})\, A \in \beta_i^t(r). \tag{3.603}$$

Proof. Analogous to the proof of Lemma 3.5.30 for the synchronous agents extension. □

Unlike for the synchronous agents extension, it is not possible to formulate a lock-step synchronous version of the Brain-in-the-Vat Lemma. We will now show why.

Lemma 3.6.11 (No Lock-step Synchronous Brain-in-the-Vat Lemma)**.** *Let $\mathcal{A} = [\![1;n]\!]$ be a set of agents with joint protocol $P^{SMC_{BCh}} = (P_1, \ldots, P_n) \in (\mathscr{C}^{MC_{BCh}} \cap \mathscr{C}^S)$, let $P_\epsilon^{SC_{\mathcal{A}^2}} \in \mathscr{C}_\epsilon^{SC_{\mathcal{A}^2}}$ be the protocol of the environment, for $\chi \in \mathscr{E}^{LSS}$, where $\chi = ((P_\epsilon^{SC_{\mathcal{A}^2}}, \mathscr{G}(0), \tau^{B\circ S,B}, EDel_C), P^{SMC_{BCh}})$, let $r \in R^\chi$, let $i \in \mathcal{A}$ be an agent, let $t > 0$ be a timestamp and let $adj = [B_{t-1}; \ldots; B_0]$ be an adjustment of extent $t-1$ satisfying*

$$B_m = (\rho_1^m, \ldots, \rho_n^m)$$

for all $0 \leq m \leq t-1$ with

$$\rho_i^m = PFake_i^m \qquad \text{and} \qquad \text{for all } j \neq i \quad \rho_j^m \in \{CFreeze, BFreeze_j\}.$$

If the protocol $P_\epsilon^{SC_{\mathcal{A}^2}}$ makes

- *agent i gullible,*

- *every agent $j \neq i$ delayable and fallible if $\rho_j^m = BFreeze_j$ for some m,*
- *all remaining agents delayable,*

then not for every run $r' \in R\left(\tau^{B\circ S,B}_{P_\epsilon^{SC_{\mathcal{A}^2}},P^{SMC_{BCh}}}, r, adj\right)$ it holds that $r' \in R^\chi$.

Proof. We will provide an appropriate counterexample. Suppose for some run $r \in R^\chi$, timestamp $t' \leq t-1$, agent $j' \neq i$, message $\mu \in Msgs$, copy number $k \in \mathbb{N}$ and action $A \in \{\boxminus\} \sqcup \overline{GActions}$

$$fake\left(i, gsend(i, j', \mu, id(i, j', \mu, k, t')) \mapsto A\right) \in \beta_\epsilon^{t'}(r). \tag{3.604}$$

From (3.475) (definition of $EDel_C$), it follows that for some timestamp $t'' \in \mathbb{N}$

$$grecv(j', i, \mu, id(i, j', \mu, k, t')) \in \beta_\epsilon^{t''}(r). \tag{3.605}$$

From (3.509) (definition of the synchronous communication environment protocol), it follows that $t'' = t'$. However by Definitions 2.7.8, 2.7.11, 2.7.12 of the interventions $PFake_i^t$, $CFreeze$ and $BFreeze_i$, it holds that for all $r' \in R\left(\tau^{B\circ S,B}_{P_\epsilon^{SC_{\mathcal{A}^2}},P^{SMC_{BCh}}}, r, adj\right)$

$$grecv(i, j', \mu, id(i, j', \mu, k, t')) \notin \beta_\epsilon^{t'}(r') \wedge fake\left(i, gsend(i, j', \mu, id(i, j', \mu, k, t')) \mapsto A\right) \in \beta_\epsilon^{t'}(r'). \tag{3.606}$$

Hence we conclude that $r' \notin R^\chi$. □

What follows are some new properties unique to the lock-step synchronous extension.

Lemma 3.6.12. *Whenever a correct agent $i \in \mathcal{A}$ in an agent context $\chi \in \mathscr{E}^{LSS}$ sends a message μ in round t, it sends μ to all agents and μ is received by all agents in the same round t.*

Proof. When a correct agent i sends a message, this is done by executing its protocol (as a fake send initiated by the environment protocol would immediately make this agent faulty). From the definition of the joint protocol (consisting of (3.521), (3.514) with (3.516)), an agent can only send a message to all agents or no one. From the admissibility condition $EDel_{\mathcal{A}^2}$ (3.475) and the synchronous communication environment protocol (3.509), it follows that a sent message has to be delivered to the receiving agent during the same round t it was sent. Suppose by contradiction that a message, sent in round t, is not received by some agent in round t. By (3.475), it follows that this message has to be correctly received at some later point in time $t' > t$. However by (3.509), a correct receive event can only happen during the same round of its corresponding send event, thus leading to a contradiction. □

Local Introspection

As for the synchronous agents extension, it is in principle possible for a lock-step synchronous agent to learn of its own defectiveness given the right circumstances.

Lemma 3.6.13. *For some agent $i \in \mathcal{A}$, an agent-context $\chi = \left((P_\epsilon^{SC_{\mathcal{A}^2}}, \mathscr{G}(0), \tau^{B \circ S}, EDel_{\mathcal{A}^2}), P^{SMC_B}\right)$, it is possible that for some (weakly) χ-based interpreted system I, some (weakly) χ-consistent run r and some timestamp $t > 0$*

$$(I, r, t) \models K_i faulty_i. \tag{3.607}$$

Proof. Analogous to the proof of Lemma 3.5.36. □

Lemma 3.6.14. *A correct agent i with local history h_i in a lock-step synchronous agents context can infer from h_i the number of virtual rounds that have passed. Formally, for an agent context $\chi \in \mathscr{E}^{LSS}$, a χ-based interpreted system $I = (R^\chi, \pi)$, a run $r \in R^\chi$ and timestamp $t \in \mathbb{N}$,*

$$(I, r, t) \models B_i nvr(r(t)) \tag{3.608}$$

Proof. Analogous to the proof of Lemma 3.5.38 from the synchronous agents extension as $\mathscr{C}^{MC_{BCh}} \cap \mathscr{C}^S \subseteq \mathscr{C}^S$. □

Global Introspection

Lemma 3.6.15. *There exists a non-excluding agent context $\chi = \left((P_\epsilon^{SC_{\mathcal{A}^2}}, \mathscr{G}(0), \tau^{B \circ S}, EDel_{\mathcal{A}^2}), \widetilde{P}^{SMC_{BCh}}\right)$, where $\chi \in \mathscr{E}^{LSS}$, such that for a χ-based interpreted system $I = (R^\chi, \pi)$, there exists a run $r \in R^\chi$, for agents $i, j \in \mathcal{A}$, where $i \neq j$ and some timestamp $t \in \mathbb{N}$, such that*

$$(I, r, t) \models B_i faulty_j. \tag{3.609}$$

Proof. Suppose the joint protocol is such that for all global histories $h \in \mathscr{G}$

$$\begin{aligned} \widetilde{P}^{SMC_{BCh}}(h) = \{(S_1, \ldots, S_n) \mid \\ (\forall i \in \mathcal{A})(\exists \mu \in Msgs)(\forall D \in S_i)\ \{send(j, \mu) \mid (\forall j \in \mathcal{A})\} \cup \{\textcircled{\scriptsize t}\} \subseteq D\}. \end{aligned} \tag{3.610}$$

meaning that every agent has to perform at least one broadcast in case it gets the opportunity to act. By Lemma 3.6.8 (agents execute their protocols only during virtual rounds), Lemma 3.6.12 (whenever a message is sent by a correct agent, all agents receive it during the same round) and (3.610), it follows that every agent receives at least one message from every correct agent during a virtual round. *Thus in all states, where i is correct, it received a message from itself, but not from some agent j, j has to be faulty.* Suppose by contradiction that this is not the case, meaning there exists a run $\overline{r}$ and timestamp $\overline{t}$, where i is correct, received a message from i (itself) but not from j, i.e.,

$$(\exists \mu \in Msgs)\ recv(i, \mu) \in \pi_1 \overline{r_i}(\overline{t}) \tag{3.611}$$

but

$$(\forall \mu \in Msgs)\ recv(j, \mu) \notin \pi_1 \overline{r_i}(\overline{t}) \tag{3.612}$$

despite the fact that j is correct.

$$(I, \overline{r}, \overline{t}) \models correct_j \tag{3.613}$$

Since agents execute their protocols only during virtual rounds (by Lemma 3.6.8) and all agents according to our defined protocol (3.610) send messages to all agents, when they get the opportunity to act and since any message that is sent, is correctly received in the same round (by Lemma 3.6.12), it is impossible that j is correct in $\overline{r}(\overline{t})$ by (2.78) (definition of $correct_j$) and Definition 2.3.3 (of $Failed\,(\overline{r},\overline{t})$). Hence, by Definition 2.132, of the belief operator the Lemma follows. □

CHAPTER 4

Conclusions and Directions of Future Research

In this thesis, we developed

(1) a generic extension framework for the existing epistemic reasoning framework [KPS+19a] for asynchronous multi-agent systems with byzantine-faulty agents,

(2) extensions for the most common distributed computing models, including reliable communication, time-bounded communication, multicasting, synchronous and lock-step synchronous agents, and even agents with coordinated actions,

(3) analyzed basic properties of the most important extensions, namely, synchronous and lock-step synchronous byzantine agents.

Our generic extension framework rests on five different handles for controlling the behavior of the environment and the agents, namely, the environment protocol, the agent protocols, the event filter, the action filters, and the admissibility conditions. Whereas liveness properties are primarily enforced by the admissibility conditions, safety properties could be enforced by different ways, some of which turned out to be favorable over others in terms of composability of extensions. This gave rise to the definition of a number of different implementation classes. A distinguishing feature of our extension framework is its explicit support for composability, in the sense that all extensions in certain implementation classes can (or cannot) be composed with others, as laid down in an explicit extension creation and combination guide.

The suitability of our extension framework has been demonstrated by providing explicit specifications of (composable) extensions that correspond to the most important distributed computing models. It is therefore possible to combine extensions such as reliable communication, broadcast communication, and synchronous agents to model classic lock-step synchronous distributed systems with byzantine agents, for example. For two such synchronous models, we also provided a set of generic results that prove the general inability of an agent to assert its own and some other agent's correctness. We also answer the question in which of those models a "brain-in-a-vat" scenario is possible.

Whereas our extension framework has been developed with genericity and completeness in mind, there are still open problems left for further research. Apart from adding extensions that cover distributed computing models currently not yet supported, we list the following two major issues:

- Develop additional generic results for pivotal implementation classes, for example, a characterization of the reliable causal cone in synchronous byzantine systems as done for asynchronous systems in [KPSF19a].
- Demonstrate the suitability of the specific extensions provided in this thesis by analyzing a concrete distributed algorithm that is known to work correctly in the corresponding model.

List of Figures

List of Tables

Index

Bibliography

[bM14] Ido Ben-Zvi and Yoram Moses. Beyond Lamport's happened-before: On time bounds and the ordering of events in distributed systems. *Journal of the ACM*, 61(2), 2014. doi:10.1145/2542181.

[bM18] Ido Ben-Zvi and Yoram Moses. Known unknowns: Time bounds and knowledge of ignorance. In *Jaakko Hintikka on Knowledge and Game-Theoretical Semantics*, pages 187–206. Springer, 2018. doi:10.1007/978-3-319-62864-6_7.

[BZM10] Ido Ben-Zvi and Yoram Moses. Beyond Lamport's happened-before: On the role of time bounds in synchronous systems. In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13–15, 2010, Proceedings*, volume 6343 of *Lecture Notes in Computer Science*, pages 421–436. Springer, 2010. doi:10.1007/978-3-642-15763-9_42.

[BZM13] Ido Ben-Zvi and Yoram Moses. Agent-time epistemics and coordination. In Kamal Lodaya, editor, *Logic and Its Applications, 5th Indian Conference, ICLA 2013, Chennai, India, January 10–12, 2013, Proceedings*, volume 7750 of *Lecture Notes in Computer Science*, pages 97–108. Springer, 2013. doi:10.1007/978-3-642-36039-8_9.

[CGM14] Armando Castañeda, Yannai A. Gonczarowski, and Yoram Moses. Unbeatable consensus. In Fabian Kuhn, editor, *Distributed Computing, 28th International Symposium, DISC 2014, Austin, TX, USA, October 12–15, 2014, Proceedings*, volume 8784 of *Lecture Notes in Computer Science*, pages 91–106. Springer, 2014. doi:10.1007/978-3-662-45174-8_7.

[dD14a] Hans van Ditmarsch. The Ditmarsch tale of wonders. In *KI 2014: Advances in Artificial Intelligence*, pages 1–12. Springer, 2014. doi:10.1007/978-3-319-11206-0_1.

[dD14b] Hans van Ditmarsch. Dynamics of lying. *Synthese*, 191(5):745–777, 2014. doi:10.1007/s11229-013-0275-3.

[DHJ+16] Danny Dolev, Keijo Heljanko, Matti Järvisalo, Janne H. Korhonen, Christoph Lenzen, Joel Rybicki, Jukka Suomela, and Siert Wieringa. Synchronous counting and computational algorithm design. *Journal of Computer and System Sciences*, 82(2):310–332, 2016. doi:10.1016/j.jcss.2015.09.002.

[DM90] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a Byzantine environment: Crash failures. *Information and Computation*, 88(2):156–186, October 1990. doi:10.1016/0890-5401(90)90014-9.

[EL17] J. Ezekiel and A. Lomuscio. Combining fault injection and model checking to verify fault tolerance, recoverability, and diagnosability in multi-agent systems. *Information and Computation*, 254(2):167–194, 2017. doi:10.1016/j.ic.2016.10.007.

[FD02] Alan Fedoruk and Ralph Deters. Improving fault-tolerance by replicating agents. In *AAMAS 2002: Autonomous Agents and Multiagent Systems (Part 2)*, pages 737–744. ACM, 2002. doi:10.1145/544862.544917.

[FHMV95a] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[FHMV95b] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[FHMV99] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. Common knowledge revisited. *Annals of Pure and Applied Logic*, 96(1–3):89–105, March 1999. doi:10.1016/S0168-0072(98)00033-5.

[Fim18] Patrik Fimml. Temporal-epistemic logic in Byzantine message-passing contexts. Master's thesis, TU Wien, Institute of Computer Engineering, 2018. URL: https://publik.tuwien.ac.at/files/publik_273448.pdf.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

[Fru19] Krisztina Fruzsa. Hope for epistemic reasoning with faulty agents! In *ESSLLI 2019 Student Session*. FOLLI, 2019. URL: http://esslli2019.folli.info/wp-content/uploads/2019/08/tentative_proceedings.pdf.

[GM13] Yannai A. Gonczarowski and Yoram Moses. Timely common knowledge: Characterising asymmetric distributed coordination via vectorial fixed points. In Burkhard C. Schipper, editor, *Proceedings of TARK XIV (2013)*, pages 79–93, Chennai, India, January 7–9, 2013. URL: http://www.tark.org/proceedings/tark_jan7_13/p79-gonczarowski.pdf.

[GM18] Guy Goren and Yoram Moses. Silence. In *PODC '18, Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 285–294, Egham, United Kingdom, July 23–27, 2018. ACM. doi:10.1145/3212734.3212768.

[GM19] Guy Goren and Yoram Moses. Byzantine consensus in the common case. Technical report, arXiv, 2019. arXiv:1905.06087.

[Gra78] J. N. Gray. Notes on data base operating systems. In *Operating Systems*. Springer, 1978. doi:10.1007/3-540-08755-9_9.

[Hin62] Jaakko Hintikka. *Knowledge and Belief: An Introduction to the Logic of the Two Notions*. Cornell University Press, 1962.

[HM90] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, July 1990. doi:10.1145/79147.79161.

[HMW01] Joseph Y. Halpern, Yoram Moses, and Orli Waarts. A characterization of eventual Byzantine agreement. *SIAM Journal on Computing*, 31(3):838–865, 2001. doi:10.1137/S0097539798340217.

[Iye95] Ravishankar K. Iyer. Experimental evaluation. In *FTCS 1995: Fault-tolerant Computing*, pages 115–132. IEEE, 1995.

[KK07] Meir Kalech and Gal A. Kaminka. On the design of coordination diagnosis algorithms for teams of situated agents. *Artificial Intelligence*, 171(8–9):491–513, 2007. doi:10.1016/j.artint.2007.03.005.

[KPS+19a] Roman Kuznets, Laurent Prosperi, Ulrich Schmid, Krisztina Fruzsa, and Lucas Gréaux. Knowledge in Byzantine message-passing systems I: Framework and the causal cone. Technical Report TUW-260549, TU Wien, 2019. URL: https://publik.tuwien.ac.at/files/publik_260549.pdf.

[KPS+19b] Roman Kuznets, Laurent Prosperi, Ulrich Schmid, Krisztina Fruzsa, and Lucas Gréaux. Knowledge in byzantine message-passing systems I: Framework and the causal cone. Technical Report TUW-260549, TU Wien, Institute of Computer Engineering, 2019. URL: https://publik.tuwien.ac.at/files/publik_260549.pdf.

[KPSF19a] Roman Kuznets, Laurent Prosperi, Ulrich Schmid, and Krisztina Fruzsa. Causality and epistemic reasoning in byzantine multi-agent systems. In *TARK 2019: Theoretical Aspects of Rationality and Knowledge*, pages 293–312. Open Publishing Association, 2019. doi:10.4204/EPTCS.297.19.

[KPSF19b] Roman Kuznets, Laurent Prosperi, Ulrich Schmid, and Krisztina Fruzsa. Epistemic reasoning with byzantine-faulty agents. In *FroCoS 2019: Frontiers of Combining Systems*, pages 259–276. Springer, 2019. doi:10.1007/978-3-030-29007-8_15.

[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.

[LKWB18] Marijana Lazić, Igor Konnov, Josef Widder, and Roderick Bloem. Synthesis of distributed algorithms with parameterized threshold guards. In *OPODIS 2017: Principles of Distributed Systems*. Schloss Dagstuhl–LZI, 2018. doi:10.4230/LIPIcs.OPODIS.2017.32.

[LQR09] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *CAV 2009: Computer Aided Verification*, pages 682–688. Springer, 2009. doi:10.1007/978-3-642-02658-4_55.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

[Mic89] Ruben Michel. A categorical approach to distributed systems, expressibility and knowledge. In Piotr Rudnicki, editor, *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 129–143. ACM, 1989. doi:10.1145/72981.72990.

[MS93] Yoram Moses and Yoav Shoham. Belief as defeasible knowledge. *Artificial Intelligence*, 64:299–321, 1993.

[MT88] Yoram Moses and Mark R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3(1–4):121–169, November 1988. doi:10.1007/BF01762112.

[ST87] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, July 1987. doi:10.1145/28869.28876.

[WL88] Jennifer Lundelius Welch and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, 1988. doi:10.1016/0890-5401(88)90043-0.