Assuming Just Enough Fairness to make Session Types Complete for Lock-freedom

Rob van Glabbeek
Data61, CSIRO and UNSW, Australia
Sydney, Australia
Email: rvg@cs.stanford.edu

Peter Höfner
Australian National University
Canberra, Australia
Email: peter.hoefner@anu.edu.au

Ross Horne
Computer Science, University of Luxembourg
Esch-sur-Alzette, Luxembourg
Email: ross.horne@uni.lu

***Abstract*—We investigate how different fairness assumptions affect results concerning *lock-freedom*, a typical liveness property targeted by session type systems. We fix a minimal session calculus and systematically take into account all known fairness assumptions, thereby identifying precisely three interesting and semantically distinct notions of lock-freedom, all of which having a sound session type system. We then show that, by using a general merge operator in an otherwise standard approach to global session types, we obtain a session type system complete for the strongest amongst those notions of lock-freedom, which assumes only *justness* of execution paths, a minimal fairness assumption for concurrent systems.**

I. Introduction

It has long been known that there is an intimate relationship between liveness properties and fairness assumptions. Seminal work by Owicki and Lamport [1] draws attention to the fact that liveness properties, such as "each request will eventually be answered" are indispensable to create correct concurrent programs.

Typically, a liveness property does not hold for all execution paths of a concurrent system: imagine two sellers and two buyers: *buyer1* repeatedly requests product A from *seller1*, who is able to sell the product. Similarly, *buyer2* and *seller2* are able to exchange product B. Assuming that both buyers try to request infinitely many products, there is an infinite execution path where product A is always requested and bought, and B is never sold. When taking all infinite execution paths into consideration, the rudimentary liveness property mentioned by Owicki and Lamport does not hold. Ranging over all infinite or completed executions – the default assumption for many model checkers – essentially assumes only that the system as a whole progresses if there is some work to do and there is no deadlock.

When reasoning about starvation-sensitive liveness properties, i.e, properties that avoid situations where a component wants to do something but is denied forever, Owicki and Lamport state explicitly that such liveness properties depend on a fairness assumption.

Assuming that the parties in our example act independently, claiming that the aforementioned liveness property fails is unrealistic, for both sellers should be able to react on any request. It is reasonable to make some fairness assumption that ensures that the parties requesting and selling A do not impair the parties involved with B. This simple example can be used as a litmus test that any realistic fairness assumption for a concurrent system should pass.

978-1-6654-4895-6/21/$31.00 ©2021 IEEE

Thus, liveness properties have to be parametrised with a fairness assumption that rules out potential executions of a system. As the fairness assumption becomes weaker (permitting more executions), the liveness property becomes stronger (systems can do more, so the liveness property is more likely to be rejected).

A reason why there exist different notions of fairness is that some notions are not realistic for some applications. For example, an implication of making the strongest of all fairness assumptions might be that you will phone everyone in your phone book repeatedly, which is unlikely. The minimal assumption *justness* [2] does not entail this, but it does imply that you will not be prevented from having a phone conversation due to unrelated calls between others. A recent survey [2] of fairness assumptions classifies dozens of semantically distinct notions by their strength in ruling out potential executions. Thus, for every liveness property, there are dozens of incarnations of that property obtained simply by varying the underlying fairness assumption.

Not all liveness properties obtained by varying fairness assumptions are semantically distinct. We identify two key reasons why liveness properties coincide: the (fixed) choice of process model and the choice of liveness property.

In this paper, we fix the process model to be a core synchronous session calculus featuring an internal and external choice [3], [4], which is frequently studied in the context of session types. We also fix the liveness properties to follow a scheme for *lock-freedom* [5], [6], which has emerged as one of the most important liveness properties for multiparty session calculi and related calculi, such as the linear π-calculus. Lock-freedom is essentially the absence of starvation, as described above. Clearly, the choice of the fairness assumption will influence whether a system is lock-free.

The restriction to session calculi, for which *session type systems* exist, allows us to answer the following question:

> For a given fairness assumption, does there exist a session type system that is sound and/or complete, in the sense that a network is lock-free if and/or only if it is well-typed?

$$(\mathbb{N}_1 \parallel \mathbb{N}_2) \parallel \mathbb{N}_3 \equiv \mathbb{N}_1 \parallel (\mathbb{N}_2 \parallel \mathbb{N}_3) \qquad \mathbb{M} \parallel \mathbb{N} \equiv \mathbb{N} \parallel \mathbb{M} \qquad \mathbb{M} \parallel 0 \equiv \mathbb{M}$$

$$\frac{\mathbb{N} \equiv \mathbb{N}' \quad \mathbb{N}' \xrightarrow{\alpha} \mathbb{M}' \quad \mathbb{M}' \equiv \mathbb{M}}{\mathbb{N} \xrightarrow{\alpha} \mathbb{M}} \qquad \frac{k \in I}{p[\![\bigoplus_{i\in I} p_i!\lambda_i; \mathrm{T}_i]\!] \parallel \mathbb{N} \xrightarrow{\tau} p[\![\ulcorner p_i!\lambda_i; \mathrm{T}_k]\!] \parallel \mathbb{N}}$$

$$\frac{}{p[\![\mu X.\mathrm{T}]\!] \parallel \mathbb{N} \xrightarrow{\tau} p[\![\mathrm{T}\{^{\mu X.\mathrm{T}}/_X\}]\!] \parallel \mathbb{N}} \qquad \frac{k \in I}{p_k[\![\ulcorner q!\lambda_k; \mathrm{U}]\!] \parallel q[\![\sum_{i\in I} p_i?\lambda_i; \mathrm{T}_i]\!] \parallel \mathbb{N} \xrightarrow{p_k \to q : \lambda_k} p_k[\![\mathrm{U}]\!] \parallel q[\![\mathrm{T}_k]\!] \parallel \mathbb{N}}$$

Fig. 1. The default semantics for networks that we fix for this study.

Our systematic study yields the following main contributions.

1) We classify the notions of lock-freedom that arise by taking every notion of fairness in the survey [2] and using them to instantiate a parameter in a general scheme for lock-freedom. The resulting classification includes classic notions of lock-freedom of session calculi found in the literature. Hence it relates these notions as well. However, we discover that the notion of lock-freedom which arises from *justness* is new to the literature.
2) We introduce a generalisation of the projection mechanism of global types onto threads, which uses the most general possible merge operator. This solves the problem that session type systems employing global types without an explicit parallel composition operator are incomplete, in the sense that there are lock-free networks that cannot be typed. This leads to the following main result.
3) We prove that our session type system is complete for lock-freedom, when assuming justness. To the best of our knowledge, this is the first completeness result of this kind. We delineate the scope of our completeness result by showing that completeness does not hold for weaker notions of lock-freedom.
4) We prove that more notions of lock-freedom coincide when restricting to race-free networks. Furthermore, race-free networks are sound for all notions of lock-freedom, whenever we assume at least *justness*.

Following [7], [8], we employ session types that abstract from the concrete types (e.g. `Bool` or `Nat`) of messages, using labels λ instead. As a result, systems and types have a fairly similar syntax. It is fairly trivial to move from our session type system with labels to one with data and data types.

Structure of the paper: Section II introduces our session calculus and a spectrum of fairness assumptions, and then systematically classifies the resulting spectrum of lock-freedom properties. Section III presents our session type system featuring a general merge operator and guarded types, which we prove to be complete with respect to $\mathcal{L}(\mathrm{J})$ – the notion of lock-freedom arising from the assumption of justness – for all networks. Section IV considers race-free networks in order to explore the scope of soundness results. Section V situates our results with respect to notions of lock-freedom from the literature.

II. The scope: A session calculus, its key fairness notions and liveness properties

In this section, we define the session calculus and a scheme for lock freedom. We also explain various fairness assumptions and illustrate their differences through separating examples.

A. Syntax and semantics for threads and networks

Our session calculus features finitely many recursive *threads* that send and receive messages. Threads, uniquely identified by location names, feature an internal choice $\bigoplus p_i!\lambda_i$ between messages labelled λ_i sent to locations p_i (a choice made at run-time entirely by the sending thread), and an external choice $\sum p_i?\lambda_i$ amongst messages received (meaning that the thread is ready to receive different messages λ_i from p_i, but cannot influence which of them will eventually come through).

$$\begin{array}{rl} \mathrm{T} \coloneqq & \mathtt{OK} \\ \mid & \bigoplus_{i\in I} p_i!\lambda_i; \mathrm{T}_i \\ \mid & \sum_{i\in I} p_i?\lambda_i; \mathrm{T}_i \\ \mid & X \\ \mid & \mu X.\mathrm{T} \end{array} \qquad \begin{array}{rl} \mathbb{N} \coloneqq & p[\![\mathrm{T}]\!] \\ \mid & 0 \\ \mid & \mathbb{N} \parallel \mathbb{N} \end{array}$$

The index sets I are finite, and in the case of $\bigoplus_{i\in I}$ also non-empty. We enforce guarded recursion by excluding threads of the form $\mu X.X$ or $\mu X.\mu Y.\mathrm{T}$. If $p[\![\mathrm{T}]\!]$ is a sub-expression of a network $\mathbb{N}$, then p is called a *location* of $\mathbb{N}$. In a network $\mathbb{N}$, all locations are required to be distinct and all threads closed, meaning that each occurrence of a variable X is in the scope of a recursion $\mu X.\mathrm{T}$. Moreover, in each sub-expression $p_k!\lambda_k$ or $p_k?\lambda_k$, the p_k must be a location of $\mathbb{N}$. We may elide $\mathtt{OK}$; we write $p_1!\lambda_1; \mathrm{T}_1 \oplus \cdots \oplus p_n!\lambda_n; \mathrm{T}_n$ for $\bigoplus_{i\in\{1,\ldots,n\}} p_i!\lambda_i; \mathrm{T}_i$, and $p_1?\lambda_1; \mathrm{T}_1 + \cdots + p_n?\lambda_n; \mathrm{T}_n$ for $\sum_{i\in\{1,\ldots,n\}} p_i!\lambda_i; \mathrm{T}_i$. In particular, we write $p?\lambda; \mathrm{T}$ in case I is a singleton set. We follow a recent trend allowing inputs in an external choice to listen to different locations [7], [9], which allows us to broaden the scope of our investigation.

A reduction semantics for our session calculus: The rules for our session calculus, presented in Figure 1, are fairly standard. In this semantics, an output that a thread has committed to can interact synchronously with some input in an external choice. Also, recursion is unfolded by a τ-transition and the standard associativity and commutativity of parallel composition can be applied to enable any transition.

A design decision, we will demonstrate to be significant, is that there is a τ-transition for resolving all internal choices.

To ensure that singleton internal choices perform only one τ-transition (and not a diverging sequence of τ–transitions), the transition ends in a *network state* that is not a syntactically valid network. *Network states* are comprised of located *thread states*, which due to the annotation $\ulcorner$, are not necessarily threads themselves.

B. Fairness notions for session calculi

We now discuss three fairness assumptions for our session calculus. A fairness assumption restricts the set of complete execution paths, here simply referred to as *paths*.

Definition 1: A path consists of a network state $\mathbb{N}_0$ and a maximal list of transitions $\mathbb{N}_i \xrightarrow{\alpha_i} \mathbb{N}_{i+1}$, permitted by Figure 1.

Maximality ensures that either the list is infinite or the final network state has no outgoing transition, that is, we restrict ourselves to *complete* execution paths.

A *fairness notion* $\mathcal{F}$ characterises a subset of all paths as the *fair* ones, modelling executions that we assume can actually occur; we refer to such paths as *$\mathcal{F}$-fair paths*. It is required to satisfy the condition of *feasibility* [10], saying that each finite prefix of a path is also a prefix of a fair path. One notion of fairness $\mathcal{G}$ is *stronger* than another one $\mathcal{F}$ – in symbols $\mathcal{F} \preceq \mathcal{G}$ – if it rules out more paths as unfair.

A network $\mathbb{N}$ *successfully terminates* under a fairness notion $\mathcal{F}$ iff all fair paths successfully terminate, i.e., all components of $\mathbb{N}$ eventually take the form $p[\![\,\mathtt{OK}\,]\!]$.

A liveness property, or more generally a *linear-time property*, is formalised as a property φ of paths. It holds for network state $\mathbb{N}_0$ under a certain fairness assumption iff all fair paths starting in $\mathbb{N}_0$ satisfy φ.

B.1 Strong and weak fairness: In [2], the concepts of *strong and weak fairness* are parametrised by the notion of a *task*. What a task is may differ from one notion of fairness to another, but for each task it should be clear when it is *enabled* in a network state, and when a path *engages* in a task. A task T is said to be *relentlessly* enabled on a path π if each suffix of π contains a network state in which T is enabled; it is *perpetually* enabled if it is enabled in all network states of π. A path π is *strongly fair* if, for each suffix π' of π, each task that is relentlessly enabled on π' is engaged in by π'. It is *weakly fair* if, for each suffix π' of π, each task that is perpetually enabled on π' is engaged in by π'.

Given a notion of a task $\mathfrak{t}$, the concept of strong fairness $\mathsf{S}\mathfrak{t}$ is always stronger than its weak counterpart $\mathsf{W}\mathfrak{t}$, i.e., $\mathsf{W}\mathfrak{t} \preceq \mathsf{S}\mathfrak{t}$.

In [2], several notions of fairness found in the literature are characterised through formalising what constitutes a task. *Fairness of transitions* is obtained by taking the tasks to be the transitions. Such a task is enabled in a network state $\mathbb{N}$ if $\mathbb{N}$ is the source state of that transition. A path π engages in a transition if that transition occurs in π.

Fact 1: Strong fairness of transitions (ST) characterises exactly those paths π with the property that whenever a transition is relentlessly enabled on π then the transition must be taken infinitely often on π; it rules out all other paths.

In [2], it is shown that for finite-state systems strong fairness of transitions (ST) is the strongest feasible notion of fairness.

Example 1: Consider the following network where a *buyer* chooses to talk to or to buy a product from a *seller*, after which the order is shipped.

$$\begin{array}{ll} & \mathit{buyer}[\![\,\mu X.(\mathit{seller}!\mathtt{talk}; X \oplus \mathit{seller}!\mathtt{buy})\,]\!] \\ \parallel & \mathit{seller}[\![\,\mu Y.(\mathit{buyer}?\mathtt{talk}; Y \\ & \qquad + \mathit{buyer}?\mathtt{buy}; \mathit{shipper}!\mathtt{order})\,]\!] \\ \parallel & \mathit{shipper}[\![\,\mathit{seller}?\mathtt{order}\,]\!] \end{array}$$

The network successfully terminates when assuming ST, for in the only infinite execution the τ-transition belonging to instruction *seller*!buy is relentlessly enabled but never taken.

A notion of task that figures prominently in the literature is that of a *component*. A component is one of the prime elements in a parallel composition – in a network expression it is completely determined by its location. Each transition involves either one or two components. A component is *enabled* in a network state iff a transition involving that component is enabled; a path *engages* in a component iff it contains a transition that involves that component.

We define a function *comp* which returns for a transition the set of components participating in the transition. Each transition labelled τ involves exactly one component (location) evident from the rule; each transition labelled $p \to q : \lambda$ involves exactly two components, p and q. This defines strong and weak fairness of components.

Fact 2: Strong fairness of components (SC) characterises the paths π such that, for any location p, if there are transitions involving p relentlessly enabled on π, then a transition that involves p must be taken infinitely often on π.

Fact 3: A path π satisfies *weak fairness of components* (WC) whenever, for every location p, if some transition involving p is, from some state onwards, perpetually enabled, then a transition that involves p occurs infinitely often in π.

Under the fairness assumption SC, Example 1 does not successfully terminate, for there is an infinite path where, alternately, the buyer performs a τ-transition to select the left branch of its choice and then the *buyer* and *seller* talk to each other. Along this path there is never a transition enabled that involves the *shipper*; hence that branch need never be taken. This illustrates that SC allows strictly more paths than ST, i.e., SC $\precneqq$ ST.

Example 2: To see that SC excludes some paths, consider the following network.

$$\begin{array}{ll} & \mathit{seller}[\![\,\mu X.(\mathit{buyer}1?\mathtt{order1}; X + \mathit{buyer}2?\mathtt{order2})\,]\!] \\ \parallel & \mathit{buyer}1[\![\,\mu Y.\mathit{seller}!\mathtt{order1}; Y\,]\!] \\ \parallel & \mathit{buyer}2[\![\,\mathit{seller}!\mathtt{order2}\,]\!] \end{array}$$

The above network terminates under SC (albeit in a state where *buyer*1 has not successfully terminated), for, in any infinite execution, a transition from *buyer*2 is relentlessly enabled but never taken. It does not need to terminate under weak fairness of components, for no transition is enabled perpetually due to the τ-transitions that unfold the recursion after each communication.

Guaranteeing termination in this example seems wrong as the fairness assumption constrains the 'free will' of the *seller* in the sense that they have to sell items to *buyer*2. Therefore we will introduce a weaker fairness assumption.

B.2 Justness: We consider a minimal notion of fairness that guarantees only that concurrent transitions cannot prevent each other from happening. Informally, two transitions are concurrent if no component is involved in both transitions.

Definition 2: Two transitions t and u are *concurrent*, notation $t \smile u$, if $\mathit{comp}(t) \cap \mathit{comp}(u) = \emptyset$.

Justness guarantees that once a transition is enabled that stems from a set of parallel components, one (or more) of these components will eventually partake in a transition.

Definition 3: A path π is *just* whenever, for every suffix of π beginning with state s and for every transition t enabled in state s, some transition u occurs in that suffix such that $t \not\smile u$. Equivalently, one might say that no enabled transition is denied forever only by concurrent transitions. The corresponding fairness assumption, which only allows just paths, is called *justness* (J).

Example 2 illustrates that J is strictly weaker than SC, i.e., J rules out fewer paths. While this system terminates under SC, it does not necessarily terminate under J, for it allows infinite communication between the *seller* and *buyer*1. Although the transition involving *buyer*2 is relentlessly enabled, it is not ruled out by justness since the *seller* is involved in both communications.

Justness is however enough to assume that in our leading example at the top of the introduction, the two concurrent interactions cannot prevent each other from occurring.

Example 3: More formally, we can model the scenario described at the top of the introduction as follows.

$$\begin{array}{rl} & \mathit{seller}1[\![\mu X.\mathit{buyer}1?\mathtt{order}; X]\!] \\ \| & \mathit{buyer}1[\![\mu Y.\mathit{seller}1!\mathtt{order}; Y]\!] \\ \| & \mathit{seller}2[\![\mu Z.\mathit{buyer}2?\mathtt{order}; Z]\!] \\ \| & \mathit{buyer}2[\![\mu W.\mathit{seller}2!\mathtt{order}; W]\!] \end{array}$$

There is no just path where *seller*2 and *buyer*2 never act. Indeed, for any just path all components act infinitely often.

In general, J $\preceq$ WC holds [2]. In addition, for our session calculus, justness coincides with weak fairness of components.

Proposition 1: WC coincides with J.

Proof: Let π be an infinite path in our network that is not WC-fair. So, on a suffix of π, a component p is perpetually enabled, but never taken. In case p is stuck in a state where its next transition is a τ, then π is not just.

In case p is stuck in a state $\ulcorner q!\lambda; \mathrm{T}$, then, for component p to be perpetually enabled, q must always be in a state $\sum_{i\in I} p_i?\lambda_i; \mathrm{T}_i$ with $p = p_k$ and $\lambda = \lambda_k$ for some $k \in I$. Location q must get stuck in such a state, for if q keeps moving, it will at some point reach a state $\mu X.\mathrm{U}$, which is not of the above form. Consequently, π is not just.

The remaining case is that p is stuck in a state of the form $\sum_{i\in I} p_i?\lambda_i; \mathrm{T}_i$. For component p to be enabled, a component p_k with $k \in I$ must be in a state $p!\lambda_k; \mathrm{T}$. Again it follows that π is not just. $\square$

As we will observe later, under a different choice of semantics of our session calculus, J and WC do not coincide.

$$\begin{tikzcd} & \textit{deadlock-freedom} \\ \mathrm{ST} \arrow[d] & \mathcal{L}(\mathrm{ST}) \arrow[u, Rightarrow] \\ \mathrm{SC} \arrow[d] & \mathcal{L}(\mathrm{SC}) \arrow[u, Rightarrow] \\ \mathrm{J} = \mathrm{WC} \arrow[d] & \mathcal{L}(\mathrm{J}) \arrow[u, Rightarrow] \\ \mathrm{P} = \mathrm{WT} & \mathcal{L}(\mathrm{P}) \arrow[u, Rightarrow] \end{tikzcd}$$

Fig. 2. A classification for our session calculus of fairness assumptions and liveness properties.

B.3 Further notions of fairness: If we define *weak fairness of transitions* (WT), where, as for ST, the tasks are the individual transitions, then WT imposes no restrictions on the completed traces for our languages. To see why, observe that in any infinite path, no transition is enabled perpetually due to the τ-transitions for unfolding recursion. This most liberal fairness assumption, which we denote P,[1] only guarantees that the system as a whole will progress if some transition is enabled.

The survey [2] classifies 21 different notions of fairness, covering all common notions found in the literature. In our session calculus, many of these notions coincide, so that only 7 different notions of fairness remain; see Appendix A of [17].

Here we have presented those that we found to be the most important notions for session calculi – summarised in Figure 2. Notably, there are strong fairness assumptions strictly between SC and ST. However, every fairness assumption from [2] leads to a notion of lock-freedom that coincides with one based on a fairness assumption defined in this section [17].

C. A scheme for lock-freedom

As discussed, a fairness assumption rules out certain paths for given systems. As lock-freedom considers only the paths of a system that can actually be taken, it depends on the

[1] *On terminology.* In related work [2], P stands for "progress", the assumption that a system cannot spontaneously halt as long as it is neither deadlocked nor successfully terminated. However, the word "progress" is heavily overloaded, meaning anything from deadlock-freedom [11], [12] and lock-freedom [5], [13], [14] to other liveness properties [15], such as weak and strong normalisation. That means, it refers to some desired property rather than an assumption on paths. Furthermore, there are related liveness properties such as *global progress* that concerns delegation [16].

underlying fairness assumption. Hence, a *scheme for lock-freedom* reads as follows:

Along any $\mathcal{F}$-fair path, if a component has not successfully terminated, then it must eventually do something. (1)

We can now formally define our scheme for lock-freedom with respect to a fairness assumption $\mathcal{F}$.

Definition 4: Let $\mathcal{F}$ be a fairness assumption. A network $\mathbb{N}$ satisfies liveness property $\mathcal{L}(\mathcal{F})$ (for short $\mathbb{N} \models \mathcal{L}(\mathcal{F})$) if, for each $\mathcal{F}$-fair path π starting in $\mathbb{N}$ and each location p of $\mathbb{N}$,

- either p successfully terminates on π, or
- π contains infinitely many transitions involving p.

Remember that a location p successfully terminates when it is of the form $p[\![\mathtt{OK}]\!]$. The letter $\mathcal{L}$ indicates "liveness" or "lock-freedom".

We say $\mathcal{L}(\mathcal{F})$ is *stronger* than $\mathcal{L}(\mathcal{G})$, denoted by $\mathcal{L}(\mathcal{F}) \Rightarrow \mathcal{L}(\mathcal{G})$, if $\mathbb{N} \models \mathcal{L}(\mathcal{F}) \Rightarrow \mathbb{N} \models \mathcal{L}(\mathcal{G})$, for all $\mathbb{N}$. It is *strictly stronger* if moreover $\mathcal{L}(\mathcal{G}) \not\Rightarrow \mathcal{L}(\mathcal{F})$. In case a fairness assumption $\mathcal{G}$ is stronger than $\mathcal{F}$, then $\mathcal{L}(\mathcal{G})$ is a weaker property than $\mathcal{L}(\mathcal{F})$.

Proposition 2: $\mathcal{F} \preceq \mathcal{G}$ implies $\mathcal{L}(\mathcal{F}) \Rightarrow \mathcal{L}(\mathcal{G})$, for fairness assumptions $\mathcal{F}$ and $\mathcal{G}$.

Intuitively, any path of $\mathbb{N}$ that is lock-free under $\mathcal{F}$ will also be lock-free under $\mathcal{G}$. Since $\mathcal{G}$ rules out more paths than $\mathcal{F}$ and since $\mathcal{L}$ is defined over paths, the proof is obvious.

A network has a *deadlock* (state) if there exists a reachable network state without outgoing transitions that is not successfully terminated; a network is *deadlock-free* if it does not have a deadlock.

Clearly, $\mathcal{L}(\mathrm{ST})$ implies deadlock-freedom, since every finite execution can be extended to some ST-fair path, using feasibility. In networks consisting of one or two parties only, deadlock-freedom coincides with all notions of lock-freedom. Deadlock-freedom, however, is considered to be insufficient for networks with three or more locations, as those networks may experience starvation: *starvation* occurs when there is an execution path along which some component wants to perform a task but no task involving that component occurs.

Using the relationship between $\mathcal{L}(\mathrm{ST})$ and deadlock-freedom, as well as Proposition 2, yields the classification of liveness properties on the right-hand side of Figure 2. Since $\mathcal{F} \precneqq \mathcal{G}$ does not imply that $\mathcal{L}(\mathcal{F})$ is strictly stronger than $\mathcal{L}(\mathcal{G})$, we provide separating examples to prove that the presented notions of lock-freedom are different.

C.1 $\mathcal{L}(ST)$ is strictly stronger than deadlock-freedom:

Example 4: Consider the following network, where a *buyer* purchases goods repeatedly from a *seller*, while a *shipper* is awaiting an order that is never placed.

$$\begin{aligned} & \mathit{buyer}[\![\mu X.\mathit{seller}!\mathtt{buy}; X]\!] \\ \parallel\ & \mathit{seller}[\![\mu Y.\mathit{buyer}?\mathtt{buy}; Y]\!] \\ \parallel\ & \mathit{shipper}[\![\mathit{seller}?\mathtt{order}]\!] \end{aligned}$$

This network is deadlock-free, for the buyer and seller can always interact; it does not satisfy $\mathcal{L}(\mathrm{ST})$ as *shipper* is not in state $\mathtt{OK}$ and is never involved in a transition.

C.2 $\mathcal{L}(SC)$ is strictly stronger than $\mathcal{L}(ST)$: Consider Example 1. We have seen that all ST-fair paths successfully terminate. In particular, along all fair paths the *shipper* performs a transition. In contrast, there is an infinite SC-fair path where the *shipper* neither makes a transition nor successfully terminates.

The following example separates ST from SC without considering termination.

Example 5: Consider the following network, where a *buyer* talks alternatingly to two *seller*s, but talks to each seller for as long as they desire.

$$\begin{aligned} & \mathit{buyer}[\![\mu X.(\mathit{seller}1?\mathtt{talk}; X \\ & \qquad + \mathit{seller}1?\mathtt{wait}; \\ & \qquad\quad \mu Z.(\mathit{seller}2?\mathtt{talk}; Z \\ & \qquad\qquad + \mathit{seller}2?\mathtt{wait}; X))\ ]\!] \\ \parallel\ & \mathit{seller}1[\![\mu V.(\mathit{buyer}!\mathtt{talk}; V \oplus \mathit{buyer}!\mathtt{wait}; V\ )]\!] \\ \parallel\ & \mathit{seller}2[\![\mu W.(\mathit{buyer}!\mathtt{talk}; W \oplus \mathit{buyer}!\mathtt{wait}; W\ )]\!] \end{aligned}$$

The above network satisfies $\mathcal{L}(\mathrm{ST})$ but not $\mathcal{L}(\mathrm{SC})$, since no location terminates and there are SC-fair paths on which one of *seller*1 or *seller*2 ceases to act, violating the condition that there must be infinitely many transitions stemming from them.

C.3 $\mathcal{L}(J)$ is strictly stronger than $\mathcal{L}(SC)$: We consider a variant of Example 2.

Example 6:

$$\begin{aligned} & \mathit{seller}[\![\mu X.(\mathit{buyer}1?\mathtt{order1}; X + \mathit{buyer}2?\mathtt{order2}; X)\ ]\!] \\ \parallel\ & \mathit{buyer}1[\![\mu Y.\mathit{seller}!\mathtt{order1}; Y]\!] \\ \parallel\ & \mathit{buyer}2[\![\mu Z.\mathit{seller}!\mathtt{order2}; Z]\!] \end{aligned}$$

The above network satisfies $\mathcal{L}(\mathrm{SC})$, since each location p has a relentlessly enabled communication transition. Hence, p will engage in a communication transition infinitely often. However, the system does not satisfy $\mathcal{L}(\mathrm{J})$, since there is a J-path where *buyer*1 never acts. Namely, every communication of *buyer*1 may be preempted by a communication of *buyer*2, as both buyers communicate with the same *seller*.

Although Example 2 separates J from SC, we cannot use it as separating example for $\mathcal{L}(\mathrm{J})$ and $\mathcal{L}(\mathrm{SC})$. It does not even satisfy $\mathcal{L}(\mathrm{ST})$, since if *buyer*2 ever acts, then *buyer*1 never successfully terminates nor engages in infinitely many transitions.

C.4 $\mathcal{L}(P)$ is strictly stronger than $\mathcal{L}(J)$: The network of Example 3 – the example from the introduction – satisfies $\mathcal{L}(\mathrm{J})$, since on a just path there are infinitely many transitions stemming from each location. However, it does not satisfy $\mathcal{L}(\mathrm{P})$, since there exists a path where two components talk forever, to the exclusion of the other two. This example indicates (again) that J is the minimal realistic fairness assumption.

In [17], we analyse further notions of lock-freedom, based on other fairness assumptions.

D. Lock-freedom in the literature

There are two prevalent notions of lock-freedom in the literature, which we call Kobayashi lock-freedom and Padovani

$$\frac{\mathbb{N} \equiv \mathbb{N}' \quad \mathbb{N}' \xrightarrow{\alpha} \mathbb{M}' \quad \mathbb{M}' \equiv \mathbb{M}}{\mathbb{N} \xrightarrow{\alpha} \mathbb{M}} \qquad \frac{p\llbracket \mathrm{T}\{^{\mu X.\mathrm{T}}/_{X}\} \rrbracket \mid\mid \mathbb{N} \xrightarrow{\alpha} p\llbracket \mathrm{U} \rrbracket \mid\mid \mathbb{N}}{p\llbracket \mu X.\mathrm{T} \rrbracket \mid\mid \mathbb{N} \xrightarrow{\alpha} p\llbracket \mathrm{U} \rrbracket \mid\mid \mathbb{N}}$$

$$\frac{j \in H \quad k \in I \quad \lambda_k = \lambda_j}{p_k\llbracket \bigoplus_{h\in H} q_h!\lambda_h; \mathrm{U}_h \rrbracket \mid\mid q_j\llbracket \sum_{i\in I} p_i?\lambda_i; \mathrm{T}_i \rrbracket \mid\mid \mathbb{N} \xrightarrow{p_k \to q_j : \lambda_k} p_k\llbracket \mathrm{U}_j \rrbracket \mid\mid q_j\llbracket \mathrm{T}_k \rrbracket \mid\mid \mathbb{N}}$$

Fig. 3. A reactive semantics without τ-transitions for internal choice or recursion. The definition of $\equiv$ is unchanged.

lock-freedom, acknowledging the authors of key papers where these properties are investigated. We prove that these two notions relate to $\mathcal{L}(\mathrm{SC})$ and $\mathcal{L}(\mathrm{ST})$, respectively. We believe, however, that $\mathcal{L}(\mathrm{J})$ is a novel notion of lock-freedom. In Section V we discuss further notions.

D.1 Kobayashi lock-freedom: Our scheme (1) for lock-freedom is inspired by a scheme proposed by Kobayashi [6] in the setting of the linear π-calculus, which does not feature operators for choice. Our scheme is more general, making it applicable to several calculi.

Although Kobayashi argues that lock-freedom is parametrised by a fairness assumption, he settles for exactly one, called *strong fairness* and attributed to [18], [19], with the stated intention that: "every process that is able to participate in a communication infinitely often can eventually participate in a communication." The intended fairness assumption in [6] coincides with SC. Almost the same can be said for the formalisation of strong fairness in [6], although literally speaking the latter is slightly weaker.[2]

D.2 Padovani lock-freedom coincides with $\mathcal{L}(ST)$: Padovani [5] presents a notion of lock-freedom that does not refer explicitly to a fairness assumption. Below we use the abbreviation $\textsc{proc}(p, \mathbb{N})$ that denotes the unique thread state T such that $\mathbb{N} \equiv p\llbracket \mathrm{T} \rrbracket \mid\mid \mathbb{N}'$, if p is a location of a network state $\mathbb{N}$.

Definition 5: $\mathbb{N}$ is *Padovani lock-free* if for each reachable state $\mathbb{M}$ of $\mathbb{N}$, and for each location p of $\mathbb{M}$ such that $\textsc{proc}(p, \mathbb{M}) \neq \textsc{ok}$, network $\mathbb{M}$ has an execution path that contains a transition involving p.

Theorem 1: A network is Padovani lock-free iff it satisfies $\mathcal{L}(\mathrm{ST})$. [See Appendix C of [17] for the proof.]

E. Lock-freedom for a reactive semantics

This section demonstrates that differences between session calculi, which may appear to be merely stylistic, in fact impact the resulting notions of liveness. An alternative semantics, (e.g. [7], [8]), which we call *reactive semantics*, is given in Figure 3. Here, neither unfolding recursion nor making a choice between various send actions induces a τ-transition.

[2]The reason is that Kobayashi's intended requirement that a component must act is formalised by describing the states right before and right after that component acts, and stipulating that one must go from the former to the latter. However, in [6] there is no way to unambiguously project global states on individual components, and one can make the prescribed transition without actually involving that component.

In Definition 4, we formally introduced liveness properties for a network, parametrised by a fairness assumption. In fact, the definition also depends on the given semantics. In the remainder, we denote by $\mathcal{L}(\mathcal{F})$ a liveness property with regard to the semantics of Figure 1, and by $\mathcal{R}(\mathcal{F})$ a liveness property with regard to the reactive semantics.

Example 7: The following network has a deadlock by the default semantics of Figure 1. Consequently, it satisfies none of the properties $\mathcal{L}(\mathcal{F})$. Yet, it satisfies $\mathcal{R}(\mathcal{F})$, for each $\mathcal{F}$.

$$\begin{aligned} & \textit{buyer}\llbracket \textit{seller}!\mathtt{buy} \oplus \textit{seller}!\mathtt{order} \rrbracket \\ \mid\mid\ & \textit{seller}\llbracket \textit{buyer}?\mathtt{buy} \rrbracket \end{aligned}$$

A similar result to Proposition 2 shows that the strength of a fairness assumption partially determines the strength of the corresponding liveness property.

Proposition 3: $\mathcal{F} \preceq \mathcal{G}$ implies $\mathcal{R}(\mathcal{F}) \Rightarrow \mathcal{R}(\mathcal{G})$, for fairness assumptions $\mathcal{F}$ and $\mathcal{G}$.

Consequently, a classification of the liveness properties $\mathcal{R}(\mathcal{F})$, for $\mathcal{F}$ any of the fairness assumptions from [2], can be obtained from the classification of these fairness properties (Figure 6 in [17]) by collapsing certain entries, just as for the classification of liveness properties $\mathcal{L}(\mathcal{F})$ from Figure 2. Since the separating examples given for $\mathcal{L}(\mathcal{F})$ apply also to $\mathcal{R}(\mathcal{F})$, we end up with at least four different notions $\mathcal{R}(\mathcal{F})$. However, we expect a lattice that is quite a bit larger, with fewer notions coinciding.

As an instance of this, $\mathcal{R}(\mathrm{J})$ is strictly stronger than $\mathcal{R}(\mathrm{WC})$. Strictness is shown by the following example.

Example 8: The following network presents a *buyer* who negotiates with *seller*1 up to a point and then decides to order a product with *seller*2 and inform *seller*1 about their decision.

$$\begin{aligned} & \textit{buyer}\llbracket \mu X.(\textit{seller}1!\mathtt{negotiate}; X \\ & \qquad \oplus\ \textit{seller}2!\mathtt{order}; \textit{seller}1!\mathtt{done}) \rrbracket \\ \mid\mid\ & \textit{seller}1\llbracket \mu Y.(\textit{buyer}?\mathtt{negotiate}; Y + \textit{buyer}?\mathtt{done}) \rrbracket \\ \mid\mid\ & \textit{seller}2\llbracket \textit{buyer}?\mathtt{order} \rrbracket \end{aligned}$$

The network successfully terminates under $\mathcal{R}(\mathrm{WC})$, for a transition involving *seller*2 is perpetually enabled, when appealing to Figure 3. It does not need to terminate under justness as the *buyer* is involved in all transitions.

Similar to Example 2, guaranteeing termination in this example seems wrong as the fairness assumption constrains the *buyer*'s 'free will'. Therefore, the presented results suggest that J is a more realistic notion than WC.

III. Session types and completeness

We now focus on session type systems. A suitably crafted session type system guarantees liveness properties for a network, if the network is well-typed. We devise a session type system that is complete for $\mathcal{L}(\mathrm{J})$, meaning that all lock-free networks can be typed.

A. Global session types, projections and type judgements

We build on a widely-adopted approach for multiparty session types. It first defines a global type, describing the interacting behaviour of all parties involved. In our syntax for global types, communications of the form $p \to q{:}\lambda$ describe the sending of a message labelled λ from location p to q, and $\boxplus$ indicates a choice over a finite, non-empty index set I.

$$\begin{array}{rlr}\mathcal{G} \coloneqq & \mathrm{OK} & \text{(successful termination)}\\ \mid & \boxplus_{i\in I}\ p \to q_i{:}\lambda_i\,;\mathcal{G}_i & \text{(choice of communication)}\\ \mid & X & \text{(recursion variable)}\\ \mid & \mu X.\mathcal{G} & \text{(recursion)}\end{array}$$

As for our session type calculus we exclude types of the form $\mu X.X$ or $\mu X.\mu Y.\mathcal{G}$ to enforce guarded recursion. Moreover, for $\boxplus_{i\in I}\ p \to q_i{:}\lambda_i\,;\mathcal{G}_i$, we assume $p \neq q_i$ for all $i \in I$. That means locations cannot send messages to themselves. A global type is *closed* whenever it contains no free recursion variables. The fact that p is the same in every branch of a choice means there is a distinguished choice leader p, who makes that choice, but there may be different recipients, as in related work on flexible choices [7].

A global session type can be projected to a local view for each location. We call local types stemming from projections *projection types*. They are defined almost in the same way as threads of Section II: instead of the construct $\sum_{i\in I} p_i?\lambda_i;\mathrm{T}_i$ they feature merely its unary case $p?\lambda;\mathrm{T}$, as well as the *merge* operators $\bigsqcap_{i\in I}\mathrm{T}_i$.

We define the set of *participants* of a global type $\mathcal{G}$ recursively:

$$\begin{aligned}\mathrm{parties}(\mathrm{OK}) &= \mathrm{parties}(X) = \emptyset\\ \mathrm{parties}(\mu X.\mathcal{G}) &= \mathrm{parties}(\mathcal{G})\\ \mathrm{parties}\big(\boxplus_{i\in I} p \to q_i{:}\lambda_i\,;\mathcal{G}_i\big) &= \textstyle\bigcup_{i\in I}\{p, q_i\} \cup \mathrm{parties}(\mathcal{G}_i)\end{aligned}$$

Given a global session type $\mathcal{G}$ and location p, we define the projection $\mathcal{G}\!\restriction_p$ of $\mathcal{G}$ on p as follows.

$$\mathrm{OK}\!\restriction_p = \mathrm{OK} \qquad X\!\restriction_p = X$$

$$(\mu X.\mathcal{G})\!\restriction_p = \begin{cases}\mathrm{OK} & \text{if } p \notin \mathrm{parties}(\mathcal{G})\\ & \text{and } \mu X.\mathcal{G} \text{ is closed}\\ \mu X.(\mathcal{G}\!\restriction_p) & \text{otherwise}\end{cases}$$

$$(\boxplus_{i\in I}\ p \to q_i{:}\lambda_i\,;\mathcal{G}_i)\!\restriction_r = \begin{cases}\bigoplus_{i\in I}(p \to q_i{:}\lambda_i\,;\mathcal{G}_i)\!\restriction_r & p = r\\ \bigsqcap_{i\in I}(p \to q_i{:}\lambda_i\,;\mathcal{G}_i)\!\restriction_r & p \neq r\end{cases}$$

$$(p \to q{:}\lambda\,;\mathcal{G})\!\restriction_r = \begin{cases}q!\lambda;(\mathcal{G}\!\restriction_r) & p = r\\ p?\lambda;(\mathcal{G}\!\restriction_r) & q = r\\ \mathcal{G}\!\restriction_r & r \notin \{p, q\}\end{cases}$$

The merge operator is interpreted directly through the judgement relation $\vdash$ between threads and projection types, coinductively defined in Figure 4. See [20] for a formal definition of what it means to interpret such rules coinductively. Usually, the merge is defined independently from the type judgements; it is simply an operation that builds a single type from several types, without using an explicit merge primitive. In the standard approach [21], the work of our judgement relation $\vdash$ is split between (a) the aforementioned merge operation, (b) a subtyping relation $\leq$ between types [21, Definition 6], and (c) a relation $\vdash$ between threads and local session types [21, Figure 5]. Our use of merge as a primitive construct for generating projection types, interpreted through $\vdash$, makes merging as general as possible.

$$\frac{\mathrm{T}\{^{\mu X.\mathrm{T}}/_X\} \vdash \mathrm{U}}{\mu X.\mathrm{T} \vdash \mathrm{U}} \qquad \frac{\mathrm{T} \vdash \mathrm{U}\{^{\mu X.\mathrm{U}}/_X\}}{\mathrm{T} \vdash \mu X.\mathrm{U}}$$

$$\frac{}{\mathrm{OK} \vdash \mathrm{OK}} \qquad \frac{i \in I \quad \mathrm{T}_i \vdash \mathrm{U}_i}{\sum_{i\in I} p_i?\lambda_i;\mathrm{T}_i \vdash p_i?\lambda_i;\mathrm{U}_i}$$

$$\frac{I \subseteq J \quad \forall i \in I \quad \mathrm{T}_i \vdash \mathrm{U}_i}{\bigoplus_{i\in I} p_i!\lambda_i;\mathrm{T}_i \vdash \bigoplus_{i\in J} p_i!\lambda_i;\mathrm{U}_i} \qquad \frac{\forall i \in I \quad \mathrm{T} \vdash \mathrm{U}_i}{\mathrm{T} \vdash \bigsqcap_{i\in I}\mathrm{U}_i}$$

Fig. 4. Typing judgements relating threads to projection types.

B. Well-typed networks

The following definition plays the role of a type rule assigning a global type to a network in related systems, e.g., [7], [8], [21], [22].

Definition 6: A network $\mathbb{N} = p_1[\![\mathrm{T}_1]\!] \parallel p_2[\![\mathrm{T}_2]\!] \parallel \ldots \parallel p_n[\![\mathrm{T}_n]\!]$ is well-typed with respect to a global type $\mathcal{G}$, denoted $\mathbb{N} \vdash \mathcal{G}$, if $\mathcal{G}$ is closed, $\mathrm{parties}(\mathcal{G}) \subseteq \{p_1, p_2, \ldots, p_n\}$, and $\mathrm{T}_i \vdash \mathcal{G}\!\restriction_{p_i}$ for all i.

A network $\mathbb{N}$ is *well-typed* if $\mathbb{N} \vdash \mathcal{G}$ for some global type $\mathcal{G}$.

Example 9: A global type for the network of Example 1 is

$$\begin{aligned}\mathcal{G} = \mu X.(&\mathit{buyer} \to \mathit{seller}{:}\mathtt{talk}\,;X\\ &\boxplus\ \mathit{buyer} \to \mathit{seller}{:}\mathtt{buy}\,;\\ &\mathit{seller} \to \mathit{shipper}{:}\mathtt{order}\,;\mathrm{OK}).\end{aligned}$$

We have

$$\begin{aligned}\mathcal{G}\!\restriction_{\mathit{buyer}} &= \mu X.(\mathit{seller}!\mathtt{talk};X \oplus \mathit{seller}!\mathtt{buy};\mathrm{OK})\\ \mathcal{G}\!\restriction_{\mathit{seller}} &= \mu X.(\mathit{buyer}?\mathtt{talk};X\\ &\qquad \sqcap\ \mathit{buyer}?\mathtt{buy};\mathit{shipper}!\mathtt{order};\mathrm{OK})\\ \mathcal{G}\!\restriction_{\mathit{shipper}} &= \mu X.(X \sqcap \mathit{seller}?\mathtt{order};\mathrm{OK}).\end{aligned}$$

With the help of the rules of Figure 4, we can derive the following facts, using proofs that are not well-founded.

$$\begin{aligned}&\mu X.(\mathit{seller}!\mathtt{talk};X \oplus \mathit{seller}!\mathtt{buy};\mathrm{OK}) &&\vdash \mathcal{G}\!\restriction_{\mathit{buyer}}\\ &\mu Y.(\mathit{buyer}?\mathtt{talk};Y\\ &\quad + \mathit{buyer}?\mathtt{buy};\mathit{shipper}!\mathtt{order};\mathrm{OK}) &&\vdash \mathcal{G}\!\restriction_{\mathit{seller}}\\ &\mathit{seller}?\mathtt{order};\mathrm{OK} &&\vdash \mathcal{G}\!\restriction_{\mathit{shipper}}\end{aligned}$$

This network is well-typed. However, in the literature, it is commonly regarded as not well-typed, which may be due to the unguarded recursion in $\mathcal{G}\!\restriction_{\mathit{shipper}}$

Example 10: This network is a restriction of the previous example, where no message is sent to the *shipper* on any path.

$$\begin{array}{ll} & \mathit{buyer}[\![\mu X.(\mathit{seller}!\texttt{talk};X)\,]\!] \\ \| & \mathit{seller}[\![\mu Y.(\mathit{buyer}?\texttt{talk};Y \\ & \qquad + \mathit{buyer}?\texttt{buy};\mathit{shipper}!\texttt{order};\texttt{OK})]\!] \\ \| & \mathit{shipper}[\![\mathit{seller}?\texttt{order};\texttt{OK}]\!] \end{array}$$

By using the same global type $\mathcal{G}$, we can type the network. The following judgement makes use of the rule for internal choice in Figure 4, which permits deleting branches, as for most session subtype relations in the literature [23], [24].

$$\mu X.\mathit{seller}!\texttt{talk};X \vdash \mathcal{G}\!\upharpoonright_{\mathit{buyer}}$$

The projections to the other locations are the same as in Example 9.

This network is well-typed and deadlock-free, but is not lock-free under any fairness assumption. That is, this type system is unsound for any notion of lock-freedom we have discussed.

Any type system targeting some notion of lock-freedom presented must reject Example 10. In this paper, our design decision to ensure soundness is to require that recursion has to be guarded for projections.[3] We thereby disallow the projection $\mathcal{G}\!\upharpoonright_{\mathit{shipper}}$ in the above examples, thereby rejecting the networks in Examples 9 and 10. Since Example 9 satisfies $\mathcal{L}(\mathrm{ST})$, we have to aim for a stronger notion of lock-freedom. It will be lock-freedom under justness and we will prove that our session type system is complete for that type of lock-freedom.

C. Guarded type judgements

We define a variant of well-typedness (Definition 6) that enforces each projection type to be guarded. A projection type T is *guarded* iff each occurrence of a variable X within a subexpression $\mu X.\mathrm{U}$ of T occurs within a subexpression $p!\lambda;\mathrm{T}$ or $p?\lambda;\mathrm{T}$.

Definition 7: A network $\mathbb{N}$ is *guardedly well-typed* with respect to a global type $\mathcal{G}$, denoted $\mathbb{N} \vdash^g \mathcal{G}$, if $\mathbb{N} \vdash \mathcal{G}$ and all projections $\mathcal{G}\!\upharpoonright_p$ are guarded.

Note that $\mathcal{G}$ is guarded by definition, but this is not sufficient to ensure that $\mathcal{G}\!\upharpoonright_p$ is guarded. Examples 9 and 10 are well-typed, but not guardedly well-typed.

Example 11: Example 6, which features a competition between two buyers, is guardedly well-typed with respect to the following global type.

$$\begin{array}{l}\mathcal{G} = \mu X.(\mathit{buyer}1 \rightarrow \mathit{seller}{:}\texttt{order1}\,; \\ \qquad\qquad \mathit{buyer}2 \rightarrow \mathit{seller}{:}\texttt{order2}\,;X)\end{array}$$

All projections are guarded. Indeed, any global type without a choice will lead to guarded projections. The interesting projection relates the thread for the *seller* to the projection of the *seller*.

$$\begin{array}{l}\mathcal{G}\!\upharpoonright_{\mathit{seller}} = \mu X.(\mathit{buyer}1?\texttt{order1};\mathit{buyer}2?\texttt{order2};X) \\ \mu X.(\mathit{buyer}1?\texttt{order1};X{+}\mathit{buyer}2?\texttt{order2};X) \vdash \mathcal{G}\!\upharpoonright_{\mathit{seller}}\end{array}$$

The above judgement holds by unfolding the recursions so as to appeal twice to the rule for $\sum$ in Figure 4.

The following example illustrates that our type system cannot be complete for $\mathcal{L}(\mathrm{SC})$.

Example 12: The next network satisfies $\mathcal{L}(\mathrm{SC})$, but not $\mathcal{L}(\mathrm{J})$.

$$\begin{array}{ll} & p[\![\mu X.(q!a;X \oplus q!b;X)\,]\!] \\ \| & q[\![\mu Y.\big(p?a;Y + r?c;(r?d;Y + p?b;r?d;Y)\,\big)]\!] \\ \| & r[\![\mu Z.q!c;q!d;Z]\!] \end{array}$$

The network does not satisfies $\mathcal{L}(\mathrm{J})$ for there is an infinite just path in which locations p and q constantly communicate via $p \rightarrow q{:}a$ and r never engages in a communication. That just path is not a SC-fair, since the communication $r \rightarrow q{:}c$ involving location r is relentlessly enabled yet never taken.

The network is not well-typed, let alone guardedly well-typed, for each global type must have a subexpression $p \rightarrow q{:}a\,;\mathcal{G}_1 \boxplus p \rightarrow q{:}b\,;\mathcal{G}_2$, and hence must have a reachable state $\mathbb{M}$ in which both transitions $\mathbb{M} \xrightarrow{p \rightarrow q:a}$ and $\mathbb{M} \xrightarrow{p \rightarrow q:b}$ are enabled. Yet there is no such reachable state.

Example 12 shows that the strongest completeness result possible is completeness with respect to $\mathcal{L}(\mathrm{J})$. Before turning to our completeness proof in the next section, we demonstrate the power of our general merge operator.

Example 13: This network consists of two independent pairs of threads, both of which make a choice repeatedly.

$$\begin{array}{ll} & \mathit{buyer}1[\![\mu X.(\mathit{seller}1!\texttt{wait};X \oplus \mathit{seller}1!\texttt{order})\,]\!] \\ \| & \mathit{seller}1[\![\mu Y.(\mathit{buyer}1?\texttt{wait};Y + \mathit{buyer}1?\texttt{order})\,]\!] \\ \| & \mathit{buyer}2[\![\mu X.(\mathit{seller}2!\texttt{wait};X \oplus \mathit{seller}2!\texttt{order})\,]\!] \\ \| & \mathit{seller}2[\![\mu Y.(\mathit{buyer}2?\texttt{wait};Y + \mathit{buyer}2?\texttt{order})\,]\!] \end{array}$$

The following is a global type for this example.

$$\begin{array}{l}\mathcal{G} = \mu X.\Big(\big(\mathit{buyer}1 \rightarrow \mathit{seller}1{:}\texttt{wait}\,; \\ \qquad\qquad (\,\mathit{buyer}2 \rightarrow \mathit{seller}2{:}\texttt{wait}\,;X \\ \qquad\qquad\quad \boxplus\ \mathit{buyer}2 \rightarrow \mathit{seller}2{:}\texttt{order}\,;\mathcal{G}_Y)\big) \\ \qquad\quad \boxplus\ \mathit{buyer}1 \rightarrow \mathit{seller}1{:}\texttt{order}\,;\mathcal{G}_Z\Big)\end{array}$$

$$\begin{array}{lrcl}\text{with} & \mathcal{G}_Y & = & \mu Y.(\ \mathit{buyer}1 \rightarrow \mathit{seller}1{:}\texttt{wait}\,;Y \\ & & & \qquad \boxplus\ \mathit{buyer}1 \rightarrow \mathit{seller}1{:}\texttt{order}\,;\texttt{OK}\ ) \\ & \mathcal{G}_Z & = & \mu Z.(\ \mathit{buyer}2 \rightarrow \mathit{seller}2{:}\texttt{wait}\,;Z \\ & & & \qquad \boxplus\ \mathit{buyer}2 \rightarrow \mathit{seller}2{:}\texttt{order}\,;\texttt{OK}\ ).\end{array}$$

We can show that $\mathbb{N} \vdash^g \mathcal{G}$. For example, we have

$$\begin{array}{rcl}\mathcal{G}\!\upharpoonright_{\mathit{buyer}1} & = & \mu X.(\ \mathit{seller}1!\texttt{wait}; \\ & & \quad \big(X \sqcap \mu Y.(\ \mathit{seller}1!\texttt{wait};Y \\ & & \qquad\qquad \oplus\ \mathit{seller}1!\texttt{order};\texttt{OK})\big) \\ & & \quad \oplus\ \mathit{seller}1!\texttt{order};\texttt{OK}\ )\ \text{and} \\ \multicolumn{3}{l}{\mu X.(\mathit{seller}1!\texttt{wait};X \oplus \mathit{seller}1!\texttt{order}) \vdash \mathcal{G}\!\upharpoonright_{\mathit{buyer}1}}\end{array}$$

where the projection $\mathcal{G}\!\upharpoonright_{\mathit{buyer}1}$ on *buyer*1 is guarded.

[3]An alternative design decision for strengthening the type system, that we do not pursue here, could be to restrict the type rule for internal choice (Figure 4) to prevent branches from being deleted (c.f. [7]), which, combined with our general merge, would allow Example 9 to stay in the fold for $\mathcal{L}(\mathrm{ST})$.

$$\textsc{gt}(h,\mathbb{M}) = \begin{cases} \textsc{gt}(h,\mathbb{M}') & \text{if } \mathbb{M} \xrightarrow{\tau} \mathbb{M}' \text{ for a network } \mathbb{M}', \\ \textsc{ok} & \text{if } \textsc{proc}(p,\mathbb{M}) = \textsc{ok} \text{ for each location } p \text{ of } \mathbb{M}, \\ \textsc{deadlock} & \text{if no location is ready in } \mathbb{M}, \\ X_{\mathbb{M}} & \text{if } \mathbb{M} \text{ occurs in } h \text{ and } h \upharpoonright \mathbb{M} \text{ is complete for } \mathbb{M}, \\ \boxplus_{i\in I}\, p \to q_i : \lambda_i\,;\textsc{gt}(h_i,\mathbb{M}_i^p) & \text{if } \mathbb{M} \text{ occurs in } h,\ p = \textsc{ch}(h,\mathbb{M}) \text{ and } \textsc{proc}(p,\mathbb{M}) = \bigoplus_{i\in I} q_i!\lambda_i;\mathrm{T}_i, \\ \mu X_{\mathbb{M}}.\boxplus_{i\in I}\, p \to q_i : \lambda_i\,;\textsc{gt}(h_i,\mathbb{M}_i^p) & \text{if } p = \textsc{ch}(\varepsilon,\mathbb{M}) \text{ and } \textsc{proc}(p,\mathbb{M}) = \bigoplus_{i\in I} q_i!\lambda_i;\mathrm{T}_i. \end{cases}$$

where $h_i := h(\mathbb{M},p,q_i)$, i.e., the sequence obtained from h by appending the triple $(\mathbb{M},p,q_i)$.

Fig. 5. Algorithm for synthesising a global type for a network.

The above example is out of scope of most systems for global session types that do not feature an explicit parallel composition operator. These systems are incomplete, in the sense that there are lock-free networks that cannot be typed. Our session type system overcomes the incompleteness, due our general treatment of merge. A similar example, which also can be typed by our methodology, is given in [25]. It is used there to demonstrate that there are networks that cannot be typed using established notions of global type without parallel composition [26]. An alternative approach using coinductive projections has been proposed in [27].

D. Completeness for lock-freedom under justness

We now show one of our main results, namely that, for our session type calculus, all lock-free networks can be typed, when assuming justness. To the best of our knowledge, this is the first completeness result of this kind.

Theorem 2: If $\mathbb{N} \models \mathcal{L}(\mathrm{J})$, then $\mathbb{N}$ is guardedly well-typed.

The proof [17, Appendix D] makes use of an algorithm for synthesising a global type from a network, along with a proof that the algorithm terminates with the correct guarded type.

To express the algorithm we require the following concepts. A *reachable network*, from a given network $\mathbb{N}$, is a reachable network state $\mathbb{M}$ that happens to be a network, in the sense that $\textsc{proc}(p,\mathbb{N}) \neq {}^{\ulcorner}\mathrm{T}$ for all locations p of $\mathbb{N}$. A network $\mathbb{M}$ is *unfolded* if there is no network $\mathbb{M}'$ with $\mathbb{M} \xrightarrow{\tau} \mathbb{M}'$ (although there may be network states $\mathbb{M}'$ with $\mathbb{M} \xrightarrow{\tau} \mathbb{M}'$). The unfolding of a network $\mathbb{M}$ is the unique network $\mathbb{M}'$ such that $\mathbb{M}\ (\xrightarrow{\tau})^*\ \mathbb{M}'$ and $\mathbb{M}'$ is unfolded. A location p is *ready* in a network $\mathbb{N}$ if $\textsc{proc}(p,\mathbb{N}) = \bigoplus_{i\in I} q_i!\lambda_i;\mathrm{T}_i$, and for each $i \in I$ there exists a transition $\mathbb{N} \xrightarrow{p \to q_i : \lambda_i}\!\!\!\rightarrow \mathbb{N}_i$, using the transition relation of Figure 3. Define a *history* as a sequence of triples $(\mathbb{N},p,q)$ with $\mathbb{N}$ a network and p,q locations of $\mathbb{N}$. A history h is *complete* for a network $\mathbb{M}$ if each location that is ready in $\mathbb{M}$ occurs in h. If h is a history and $\mathbb{M}$ a network expression that occurs in h, then $h \upharpoonright \mathbb{M}$ denotes the suffix of h that starts with the first occurrence of $\mathbb{M}$ in h. Moreover, $h \upharpoonleft \mathbb{M}$ denotes the prefix of h prior to the first occurrence of $\mathbb{M}$ in h, so that $h = (h \upharpoonleft \mathbb{M})(h \upharpoonright \mathbb{M})$. Call a location p *eligible* in a network state $\mathbb{M}$ w.r.t. a history h if (a) p is ready in $\mathbb{M}$, and (b) either $\mathbb{M}$ does not occur in h, or p does not occur in $h \upharpoonright \mathbb{M}$. Finally, DEADLOCK is a constant, temporarily added to the syntax of session types.

Our algorithm requires several choices. First, we select a fresh variable $X_{\mathbb{M}}$ for each unfolded network $\mathbb{M}$ that is reachable from $\mathbb{N}$. We then pick a total order on the finite set of locations of $\mathbb{N}$, referred to as *age*, so that each nonempty set of locations has an oldest element. Finally, for each reachable network $\mathbb{M}$ and each location p that is ready in $\mathbb{M}$, say with $\textsc{proc}(p,\mathbb{M}) = \bigoplus_{i\in I} q_i!\lambda_i;\mathrm{T}_i$, and for each $i \in I$, pick a network $\mathbb{M}_i^p$ such that $\mathbb{M} \xrightarrow{p \to q_i : \lambda_i}\!\!\!\rightarrow \mathbb{M}_i^p$ and $\textsc{proc}(p,\mathbb{M}_i^p) = \mathrm{T}_i$.

Our algorithm employs the routine $\textsc{gt}(h,\mathbb{M})$, parametrised by the choice of a network $\mathbb{M}$ and a history h, as defined in Figure 5. Here, $\textsc{ch}(h,\mathbb{M})$ is a partial function that selects, for a given history h and reachable network $\mathbb{M}$, the oldest location that is eligible in $\mathbb{M}$ w.r.t. h. It is defined only when such a location exists. The case distinction in the figure is meant to be prioritised, in the sense that a later-listed option is taken only if none of the higher-listed options apply. Our algorithm is then defined to yield the global type $\textsc{gt}(\varepsilon,\mathbb{N})$, with $\mathbb{N}$ the given network and ε the empty history (sequence).

The intuition for our algorithm, which attempts to construct a global session type $\mathcal{G}$ out of a given network $\mathbb{N}$, is as follows. Since it is essential that $\mathcal{G}$ induces ongoing progress of all unterminated locations in the network, we keep track of the history h of communications incorporated in $\mathcal{G}$ until the "construction front" at network state $\mathbb{M}$. Here the routine $\textsc{gt}(h,\mathbb{M})$ specifies the next communication-choice that will be incorporated in $\mathcal{G}$. The first clause in Figure 5, where the τ-transition must unfold recursion, says that all recursions should be unfolded before attempting the remaining case distinctions. The second clause says that we can safely terminate upon reaching a state in which the threads of all locations have terminated. The last two clauses specify a choice leader p and extend $\mathcal{G}$ with the send actions of p in state $\mathbb{M}$. Here p must be a location that is ready in $\mathbb{M}$; if such a p does not exist the failed attempt is reported by including the constant DEADLOCK in the attempted session type $\mathcal{G}$ (Clause 3). Since the syntactic expression $\mathcal{G}$ must be finite, each branch that does not reach OK needs to loop back to a previous stage in the construction of $\mathcal{G}$, at some point. To facilitate looping back, we attach a recursion variable $X_{\mathbb{M}}$ to each stage we might want to loop back to, namely to each first occurrence of a network state $\mathbb{M}$ in our history. This explains the difference between Clauses 5 and 6. Clause 4 says that we can safely loop back to a previous stage if it involves the same current network state $\mathbb{M}$, and between that previous stage and the present all locations that are ready in $\mathbb{M}$ already had a turn. If Clause 4 does not apply, then $\mathbb{M}$ must have eligible locations w.r.t. h, and Clause 5 or 6 picks the oldest such location, to make sure that in the end all eligible locations get a turn.

Example 14: Applying this algorithm to the network of Example 1 yields the type of Example 9, but with a spurious recursive anchor μZ right before $\mathit{seller} \to \mathit{shipper}{:}\texttt{order}$. Applied to Example 2 it fails with possible output

$$\mu X.\mathit{buyer}1 \to \mathit{seller}{:}\texttt{order1}\,;\\ \mathit{buyer}2 \to \mathit{seller}{:}\texttt{order2}\,;\,\text{DEADLOCK}.$$

For Example 3 it yields the following correct type.

$$\mu X.(\mathit{buyer}1 \to \mathit{seller}1{:}\texttt{order}\,;\mathit{buyer}2 \to \mathit{seller}2{:}\texttt{order}\,;X)$$

For Example 4 it yields the type $\mu X.\mathit{buyer} \to \mathit{seller}{:}\texttt{buy}\,;X$; this type is incorrect for that network $\mathbb{N}$. This does not contradict the proof of Theorem 2, since $\mathbb{N} \not\models \mathcal{L}(\mathrm{J})$.

For Example 6 the algorithm yields the type of Example 11. For Example 7 it fails with output DEADLOCK.
For Example 8 it yields the following correct type.

$$\mu X.(\mathit{buyer} \to \mathit{seller}1{:}\texttt{negotiate}\,;X\\ \boxplus\, \mathit{buyer} \to \mathit{seller}2{:}\texttt{order}\,;\mu Z.\mathit{buyer} \to \mathit{seller}1{:}\texttt{done}\,;\text{OK})$$

For Example 13 it yields the type of Example 13.

Observation 1: An immediate consequence of Proposition 1 and Theorem 2, is that guardedly well-typed networks are complete for $\mathcal{L}(\mathrm{WC})$, suggesting that a carefully selected notion of weak fairness is suitable for some session calculi.

Corollary 1: If $\mathbb{N} \models \mathcal{L}(\mathrm{WC})$ then $\mathbb{N}$ is guardedly well-typed.

In contrast, recall that Example 7 satisfies $\mathcal{R}(\mathrm{P})$; yet it is not (guardedly) well-typed. This shows that there is no corresponding completeness result for any notion of lock-freedom $\mathcal{R}(\mathcal{F})$, where $\mathcal{F}$ is some notion of fairness. This is an argument for why we emphasise the semantics in Figure 1 rather than the one in Figure 3.

IV. Race-freedom and soundness

We have established that completeness holds, with respect to $\mathcal{L}(\mathrm{J})$. Hence, if we can model-check $\mathcal{L}(\mathrm{J})$, we know we can always synthesise a global type for a network. In this section, we consider soundness, meaning that a network is lock-free if it is (guardedly) well-typed. To complement our completeness result, we target $\mathcal{L}(\mathrm{J})$ and prove soundness for guardedly well-typed networks that are additionally race-free. The insight of this section is that soundness can be achieved when making the minimal fairness assumption justness.

Definition 8: A network state $\mathbb{N}$ has a *race* whenever $\mathbb{N} \xrightarrow{p \to r : \lambda} \mathbb{N}'$ and $\mathbb{N} \xrightarrow{q \to r : \mu} \mathbb{N}''$ with either $p \neq q$ or $\mathbb{N}' \neq \mathbb{N}''$. A network is *race-free* if it has no reachable network state with a race.

Figure 2 implies that $\mathcal{L}(\mathrm{J})$ is the strongest lock-freedom property we can get, with the exception of $\mathcal{L}(\mathrm{P})$. Our guarded type system cannot be sound for the latter notion of lock-freedom, for Example 3 is guardedly well-typed and race-free, but does not satisfy $\mathcal{L}(\mathrm{P})$.

Our example to distinguish J and SC features races. Indeed, there is no race-free network separating J from SC, as confirmed by the following proposition.

Proposition 4: On race-free networks, J coincides with SC, for our session calculus in Figure 1.

Proof: Let π be an infinite path in a network that is not SC-fair. So, on π, a component p is infinitely often enabled, but never taken.

In case p is stuck in a state where its next transition is a τ, then π is not just.

In case p is stuck in a state $\ulcorner q!\lambda; P$, then, in the first state on π on which p is enabled, q must be in a state $\sum_{i\in I} p_i?\lambda_i; \mathrm{T}_i$ with $p = p_k$ and $\lambda = \lambda_k$ for some $k \in I$. If q remains in this state throughout π, then π is not just. If q leaves this state via a transition of π that does not involve p, then the state where this happens must be a race, and the network is not race-free.

In the remaining case, p is stuck in a state $\sum_{i\in I} p_i?\lambda_i; \mathrm{T}_i$. For p to be enabled, a component p_k with $k \in I$ must be in a state $p!\lambda_k; P$, which reduces this case to the previous one. □

Observation 2: In contrast to Proposition 4, for the session calculus in Figure 3, J and SC do not coincide, even for race-free networks (race-freedom also needs to be reformulated for that semantics). Example 8 illustrates this fact.

Since Examples 1, 3 and 4 are race-free, the remaining four notions $\mathcal{L}(\mathrm{P})$, $\mathcal{L}(\mathrm{SC})$, $\mathcal{L}(\mathrm{ST})$ and deadlock-freedom from Figure 2 are all different for race-free networks. Consequently, for race-free networks, the only collapse of Figure 2 is $\mathcal{L}(\mathrm{J}) \Leftrightarrow \mathcal{L}(\mathrm{SC})$.

Example 15: The following network is race-free, guardedly well-typed and satisfies $\mathcal{L}(\mathrm{J})$. It is a variant of Example 8, which is also race-free, but does not satisfy $\mathcal{L}(\mathrm{J})$.

$$\begin{array}{rl} & \mathit{buyer}[\![\mu X.\big((\mathit{seller}1!\texttt{order1}; \mathit{seller}2!\texttt{wait}; X)\\ & \qquad \oplus\ (\mathit{seller}2!\texttt{order2}; \mathit{seller}1!\texttt{done})\big)\ ]\!]\\ \| & \mathit{seller}1[\![\mu Y.(\mathit{buyer}?\texttt{order1}; Y + \mathit{buyer}?\texttt{done})\ ]\!]\\ \| & \mathit{seller}2[\![\mu Z.(\mathit{buyer}?\texttt{wait}; Z + \mathit{buyer}?\texttt{order2})\ ]\!] \end{array}$$

This network is well-typed, for we can use the following global type.

$$\begin{array}{rl} \mathcal{G} = \mu X.\big(& (\mathit{buyer} \to \mathit{seller}1{:}\texttt{order1}\,;\\ & \ \mathit{buyer} \to \mathit{seller}2{:}\texttt{wait}\,;X)\\ & \boxplus\ (\mathit{buyer} \to \mathit{seller}2{:}\texttt{order2}\,;\\ & \ \mathit{buyer} \to \mathit{seller}1{:}\texttt{done}\,)\big) \end{array}$$

This example illustrates that it is possible to send messages to several locations via an internal choice in a race-free way. Many session type systems, e.g. [21], [22], [28], ensure that, for every sub-expression of the form $\sum_{i\in I} p_i?\lambda_i; \mathrm{T}_i$, we have $p_i = p_j$ for all $i, j \in I$, and $\lambda_i \neq \lambda_j$ for all $i, j \in I$ with $i \neq j$. This syntactically guarantees race-freedom, but excludes the above example.

The reverse of Theorem 2 does not hold. Hence we cannot expect a soundness result for all networks. It is not even the case that guardedly well-typed networks are deadlock-free.

Example 16: The following network is guardedly well-typed, but neither race-free nor deadlock-free, and hence certainly not $\mathcal{L}(\mathrm{J})$.

$$\begin{array}{rl} & \mathit{buyer}1[\![\mathit{seller}!\mathtt{buy1};\mathrm{OK}]\!] \\ \| & \mathit{buyer}2[\![\mathit{seller}!\mathtt{buy2};\mathrm{OK}]\!] \\ \| & \mathit{seller}[\![(\mathit{buyer}1?\mathtt{buy1};\mathit{buyer}2?\mathtt{buy2};\mathrm{OK}) \\ & \quad + (\mathit{buyer}2?\mathtt{buy2};\mathit{buyer}1?\mathtt{buy1}; \\ & \quad\quad \mathit{buyer}1!\mathtt{order};\mathrm{OK})]\!] \end{array}$$

The global type for this example is the following.

$$\mathcal{G} = \mathit{buyer}1 \to \mathit{seller}{:}\mathtt{buy1}\,;\mathit{buyer}2 \to \mathit{seller}{:}\mathtt{buy2}\,;\mathrm{OK}$$

In particular, $\mathcal{G}\restriction_{\mathit{seller}} = \mathit{buyer}1?\mathtt{buy1};\mathit{buyer}2?\mathtt{buy2}$ and

$$\begin{array}{l} \mathit{buyer}1?\mathtt{buy1};\mathit{buyer}2?\mathtt{buy2} \\ +\ \mathit{buyer}2?\mathtt{buy2};\mathit{buyer}1?\mathtt{buy1};\mathit{buyer}1!\mathtt{order} \vdash^g \mathcal{G}\restriction_{\mathit{seller}} \end{array}$$

which holds due to the rule for $\sum$ in Figure 4, permitting branches of an external choice to be removed.

This example may suggest that the culprit preventing soundness is the flexible external choice, i.e., the subtype relation $\vdash^g$. However, even if $\vdash^g$ would be almost the identity relation, with each merge on types corresponding to an external choice of the corresponding threads, there would be guardedly well-typed networks that are not deadlock-free.

Example 17: Consider the following network.

$$\begin{array}{rl} & p[\![(s!a;t!a;r!d) \oplus (s!b;t!b)]\!] \\ \| & r[\![(s?c;t?e;p?d) + (t?e;s?c)]\!] \\ \| & s[\![p?a;r!c + p?b;r!c]\!] \\ \| & t[\![p?a;r!e + p?b;r!e]\!] \end{array}$$

Using the global type

$$\begin{array}{rl} \mathcal{G} = & (p \to s{:}a\,;p \to t{:}a\,;s \to r{:}c\,;t \to r{:}e\,;p \to r{:}d\,;\mathrm{OK}) \\ & \boxplus\ p \to s{:}b\,;p \to t{:}b\,;t \to r{:}e\,;s \to r{:}c\,;\mathrm{OK} \end{array}$$

yields projections that are identical to the network threads, e.g.

$$\mathcal{G}\restriction_p = (s!a;t!a;r!d) \oplus s!b;t!b.$$

So, $\mathbb{N} \vdash^g \mathcal{G}$ follows by using the identity as subtyping relation. Yet, this network is not deadlock-free, for the execution $p \to s{:}b\,;s \to r{:}c\,;p \to t{:}b\,;t \to r{:}e$ reaches a deadlock with hanging input $p?d$ in location r.[4]

Examples 16 and 17 are both excluded by race-freedom. We now prove another main result, namely that the converse of Theorem 2 holds for race-free networks.

Theorem 3: If $\mathbb{N}$ is guardedly well-typed and race-free, then $\mathbb{N} \models \mathcal{L}(\mathrm{J})$.

The proof [Appendix F of [17]] hinges on the following *session fidelity* result, for which we appeal to race-freedom:

For race-free network states $\mathbb{N}$, if $\mathbb{N} \xrightarrow{p \to q:\lambda} \mathbb{M}$ and $\mathbb{N} \vdash^g \mathcal{G}$ then there exists $\mathcal{G}'$ such that $\mathcal{G} \xrightarrow{p \to q:\lambda}\!\!\!\!\to \mathcal{G}'$ and $\mathbb{M} \vdash^g \mathcal{G}'$. Here, $\xrightarrow{\alpha}\!\!\!\!\to$ is a transition relation on global types, defined for this purpose. Session fidelity strengthens subject reduction, by insisting that the form of $\mathcal{G}'$ reflects the transition.

From this statement we conclude that each reachable network state $\mathbb{N}'$ along any just path π is guardedly well-typed. For every location p, either there are infinitely many transitions along π involving p, or there exists a suffix π' of π stemming from network state $\mathbb{N}'$, such that location p has no further transition involving p. Using justness and the fact that $\mathbb{N}'$ is guardedly well-typed one can show that in the latter case p has successfully terminated.

Using Theorems 2 and 3, and Proposition 4 leads to a soundness and a completeness result for our type system.

Corollary 2: For race-free network $\mathbb{N}$, $\mathbb{N}$ is guardedly well-typed iff $\mathbb{N}$ satisfies $\mathcal{L}(\mathrm{SC})$.

Observation 3: Projections are defined such that recursion maps to OK whenever $p \notin \mathrm{parties}(\mathcal{G})$ and $\mu X.\mathcal{G}$ is closed (see Page 7). The latter condition plays an essential role for soundness. To see why, consider the following global type.

$$\mu X.p \to q{:}a\,;\mu Y.r \to q{:}b\,;X$$

The anchor with variable Y should, intuitively, be useless. However, if we were to exclude condition "$\mu X.\mathcal{G}$ is closed", the above global type would type the following network.

$$\begin{array}{rl} & p[\![\mu X.q!a;\mathrm{OK}]\!] \\ \| & q[\![\mu X.p?a;\mu Y.r?b;X]\!] \\ \| & r[\![\mu X.\mu Y.q!b;X]\!] \end{array}$$

This race-free network reaches a deadlock right after communications $p \to q{:}a\,;r \to q{:}b$. The above closedness-condition resolves this issue, and can be used to correct papers on global types featuring recursion binders.

V. Related and future work on lock-freedom

While our completeness result is the first of its kind, there are several soundness results for type systems with respect to some notion of lock-freedom, e.g., [5], [8], [13], [30], [31], the most closely related of which we draw attention to in this section. We also situate related work on lock-freedom with regard to our classification and point to future challenges.

a) Strong lock-freedom: Severi and Dezani-Ciancaglini propose a notion of *strong lock-freedom* [8] that coincides with $\mathcal{R}(\mathrm{J})$ for race-free networks. They employ a reactive semantics. The authors impose a restriction on paths, ensuring that all concurrent transitions proceed in lockstep. Their assumption is not a fairness assumption, as defined here, as it does not satisfy feasibility. However, it does have the effect of assuming justness, up to permutations of transitions, for race-free networks. By Observation 1, a completeness result along the lines of Theorem 2 cannot hold for strong lock-freedom. Strong lock-freedom cannot be lifted directly to networks with races. A minimal change to their definitions requiring a maximal number of enabled locations to act in every step would extend their definition to networks with races; we did not analyse this extension. Their use of coinductive syntax,

[4] This is a counterexample to subject reduction in previous work that allows multiple recipients in an external choice [7], [29]. Those soundness results can be restored by restricting to race-free networks.

rather than binders, is an alternative for avoiding the soundness problem in Observation 3 that is common in the literature.

b) Further lock-freedom schemes: Carbone, Dardha and Montesi translate Kobayashi's scheme for lock-freedom to a session calculus where both internal and external choices are with a single location [31]. Their scheme is instantiated with SC[5] and coincides with $\mathcal{L}(\text{SC})$, restricted to their calculus. Their scheme inherits the ambiguity discussed in Section II-D1. It assumes a semantics intermediate to those we study in this work, where internal choice is like in Figure 1, but recursion and singleton internal choice are reactive as in Figure 3. This makes their approach weaker than ours, if lifted directly to our calculus with flexible choices: Example 6 is lock-free under their scheme instantiated with assumption J (or even P), but does not satisfy $\mathcal{L}(\text{J})$ (or even $\mathcal{R}(\text{J})$) in our scheme. Note that their work concerns binary session types with delegation, which we do not consider.

Scalas and Yoshida propose the notions of LIVE, LIVE+, and LIVE++ [25]. The first, LIVE, follows the scheme of Padovani, hence coincides with $\mathcal{L}(\text{ST})$. The second, LIVE+, is essentially another formulation of $\mathcal{L}(\text{SC})$. The third, LIVE++, coincides with $\mathcal{L}(\text{P})$, hence is unsound for session calculi since it rejects key examples such as Example 3. The definitions of [25] are arguably less portable than Definition 5, since their definitions refer to specific language features.

c) Asynchronous session calculi: An evaluation of lock-freedom for asynchronous calculi, where queues are inserted between communicating threads, requires separate attention. The asynchronous analogue to our session calculus is an infinite-state system. Therefore, ST is no longer the strongest fairness assumption; this is now *full fairness* (Fu) [2]. At the other end of the spectrum, there are also complications when defining concurrency of transitions (Definition 2). It is a design decision whether a thread is treated as a single component along with its queues, and whether enqueue and dequeue events for the same queue are dependent or concurrent. Consequently, synchrony/asynchrony and the spectrum of fairness assumptions are not entirely perpendicular dimensions when defining notions of lock-freedom. Fairness plays an essential role in related work on preciseness of subtyping for asynchronous calculi [32], which is further evidence that fairness assumptions require scrutiny here.

d) Synthesis and multiparty compatibility: The body of literature on synthesising global types from multiparty compatible local types [26], [33], [34] plays a complementary role to our synthesis results, used to establish completeness. Usually, it is immediate that networks inhabiting a global type are multiparty compatible. Hence, we expect that a corollary of Theorem 2 is that $\mathcal{L}(\text{J})$ implies multiparty compatibility, for some notion of multiparty compatibility. If we further assume a synthesis result showing that multiparty compatible networks are guardedly well-typed – under conditions such as race-freedom – then that notion of multiparty compatibility coincides with $\mathcal{L}(\text{J})$. Such a result for our global type system, does not quite follow immediately from synthesis results in the literature, since Example 13 would require parallel composition in related work. The formal development of multiparty compatibility is left as future work.

[5]The authors do not provide a definition of fairness, but cite Kobayashi [6] instead. Kobayashi's definition does not lift immediately to the calculus of [31]. However, the authors appear to intend SC.

e) Fair subtyping and weak normalisation: A fair subtyping relation has been defined for session types as the largest relation over threads that preserves weak normalisation [35]. Weak normalisation is the property that, at any point during an execution, it is not inevitable that a network will not successfully terminate. Weak normalisation is strictly stronger than Padovani's notion of lock-freedom (Definition 5) – since the possibility of all components to successfully terminate entails the possibility of all components performing some enabled action – but is incomparable to liveness properties stronger than $\mathcal{L}(\text{SC})$, including $\mathcal{L}(\text{J})$ – Example 1 is weakly normalising, but does not satisfy $\mathcal{L}(\text{SC})$. Consequently, the proposed notion of fair subtyping does not quite fit lock-freedom. Investigating a notion of fair subtyping that is adequate for $\mathcal{L}(\text{ST})$ rather than weak normalisation, and also identifying a session type system complete for $\mathcal{L}(\text{ST})$, as hinted at in the discussion surrounding Example 10, is future work. In particular, we do not claim that $\mathcal{L}(\text{J})$ is the only notion of lock-freedom that can be characterised by some session type system.

VI. CONCLUSION

In this paper, we have systematically classified the notions of lock-freedom that arise by taking every fairness assumption listed in a recent survey [2]. Based on our comprehensive analysis, we are compelled to put forward a notion of fairness suitable for session calculi: *justness* (Definition 3), and its resulting notion of lock-freedom $\mathcal{L}(\text{J})$, which we propose to call *"just lock-freedom"*. Through a generalisation of the classical merge operation on local session types, we have devised a session type system that is complete for just lock-freedom. Moreover, race-free networks are sound for just lock-freedom. Justness is always reasonable to assume, since it does not constrain the 'free will' of participants (c.f. Examples 2 and 8), while ensuring that concurrent transitions do not constrain each other (c.f. Examples 3 and 13).

A strength of our results is that completeness (Theorem 2) holds for networks with flexible choice, in which branches of the same choice operator may involve different locations. Completeness suggests a methodology for session calculi that allows us to pass straight from any network satisfying our realistic notion of lock-freedom $\mathcal{L}(\text{J})$ to a global session type. The methodology would be to directly model check that a network satisfies $\mathcal{L}(\text{J})$, and then use the algorithm in Figure 5 to synthesise a global type for that network. This methodology works even for networks featuring races.

Interestingly, there are no previous results synthesising global types directly from lock-freedom. Indeed, Example 13, which satisfies almost all notions of lock-freedom in the literature, is known to be out of scope of related session type systems based on global types without explicit parallel composition. While this incompleteness issue in related work is partly due to the less general merge operator employed in those systems, another reason that enables us to obtain the completeness result in Theorem 2 is our scrutiny of the role of fairness assumptions. In fact, Example 12 shows that completeness of our session type system cannot be attained when assuming strong fairness of components. Furthermore, even small variations in the choice of semantics for the transition system can affect fairness assumptions significantly, weakening corresponding notions of lock-freedom (see Observation 1). Indeed, amongst all notions of lock-freedom considered in this paper, only $\mathcal{L}(\mathrm{J})$ yields both completeness for all networks and soundness for race-free networks (Theorem 3).

Acknowledgement: The key question addressed in this paper arose in conversation with Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini, to whom we are grateful for their generous feedback on this work.

References

[1] S. S. Owicki and L. Lamport, "Proving liveness properties of concurrent programs," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 455–495, 1982. Available: https://doi.org/10.1145/357172.357178

[2] R. J. van Glabbeek and P. Höfner, "Progress, justness, and fairness," *ACM Computing Surveys*, vol. 52, no. 4, 2019. Available: https://doi.org/10.1145/3329125

[3] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe, "A theory of communicating sequential processes," *Journal of the ACM*, vol. 31, no. 3, pp. 560–599, 1984. Available: https://doi.org/10.1145/828.833

[4] R. De Nicola and M. Hennessy, "CCS without τ's," in *TAPSOFT '87*, H. Ehrig, R. Kowalski, G. Levi, and U. Montanari, Eds. Springer, 1987, pp. 138–152. Available: https://doi.org/10.1007/3-540-17660-8_53

[5] L. Padovani, "Deadlock and lock freedom in the linear π-calculus," in *CSL-LICS '14*, T. A. Henzinger and D. Miller, Eds. ACM, 2014. Available: https://doi.org/10.1145/2603088.2603116

[6] N. Kobayashi, "A type system for lock-free processes," *Inf. Comput.*, vol. 177, no. 2, pp. 122–159, 2002. Available: https://doi.org/10.1006/inco.2002.3171

[7] I. Castellani, M. Dezani-Ciancaglini, and P. Giannini, "Reversible sessions with flexible choices," *Acta Informatica*, vol. 56, no. 7-8, pp. 553–583, 2019. Available: https://doi.org/10.1007/s00236-019-00332-y

[8] P. Severi and M. Dezani-Ciancaglini, "Observational equivalence for multiparty sessions," *Fundam. Inform.*, vol. 170, no. 1-3, pp. 267–305, 2019. Available: https://doi.org/10.3233/FI-2019-1863

[9] S. Jongmans and N. Yoshida, "Exploring type-level bisimilarity towards more expressive multiparty session types," in *ESOP '20*, P. Müller, Ed. Springer, 2020, pp. 251–279. Available: https://doi.org/10.1007/978-3-030-44914-8_10

[10] K. R. Apt, N. Francez, and S. Katz, "Appraising fairness in languages for distributed programming," *Distributed Computing*, vol. 2, pp. 226–241, 1988. Available: https://doi.org/10.1007/BF01872848

[11] K. Honda, N. Yoshida, and M. Carbone, "Multiparty asynchronous session types," *Journal of the ACM*, vol. 63, no. 1, pp. 9:1–9:67, 2016. Available: https://doi.org/10.1145/2827695

[12] M. Dezani-Ciancaglini, D. Mostrous, N. Yoshida, and S. Drossopoulou, "Session types for object-oriented languages," in *ECOOP '06*, D. Thomas, Ed. Springer, 2006, pp. 328–352. Available: https://doi.org/10.1007/11785477_20

[13] M. Dezani-Ciancaglini, N. Yoshida, A. J. Ahern, and S. Drossopoulou, "A distributed object-oriented language with session types," in *Trustworthy Global Computing, International Symposium, TGC '05, Revised Selected Papers*, R. De Nicola and D. Sangiorgi, Eds. Springer, 2005, pp. 299–318. Available: https://doi.org/10.1007/11580850_16

[14] E. Najm, A. Nimour, and J. Stefani, "Guaranteeing liveness in an object calculus through behavioural typing," in *FORTE XII / PSTV XIX*, J. Wu, S. T. Chanson, and Q. Gao, Eds. Kluwer, 1999, pp. 203–221. Available: https://doi.org/10.1007/978-0-387-35578-8_12

[15] J. Misra, *A Discipline of Multiprogramming: Programming Theory for Distributed Applications*. Springer, 2001, ch. Progress Properties, pp. 155–213. Available: https://doi.org/10.1007/978-1-4419-8528-6_6

[16] M. Coppo, M. Dezani-Ciancaglini, N. Yoshida, and L. Padovani, "Global progress for dynamically interleaved multiparty sessions," *Mathematical Structures in Computer Science*, vol. 26, no. 2, pp. 238–302, 2016. Available: https://doi.org/10.1017/S0960129514000188

[17] R. Horne, R. J. van Glabbeek, and P. Höfner, "Assuming just enough fairness to make session types complete for lock-freedom." Available: https://arxiv.org/abs/2104.14226

[18] G. Costa and C. Stirling, "Weak and strong fairness in CCS," *Information and Computation*, vol. 73, no. 3, pp. 207–244, 1987. Available: https://doi.org/10.1016/0890-5401(87)90013-7

[19] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science (vol. B): Formal Models and Semantics*. MIT press, 1990, pp. 995–1072. Available: https://dl.acm.org/doi/10.5555/114891.114907

[20] R. J. van Glabbeek, "Coinductive validity." Available: http://arxiv.org/abs/2104.13021

[21] N. Yoshida and L. Gheri, "A very gentle introduction to multiparty session types," in *Distributed Computing and Internet Technology*, D. V. Hung and M. D´Souza, Eds. Springer, 2020, pp. 73–93. Available: https://doi.org/10.1007/978-3-030-36987-3_5

[22] P.-M. Denielou, N. Yoshida, A. Bejleri, and R. Hu, "Parameterised Multiparty Session Types," *Log. Meth. Comp. Sci.*, vol. Volume 8, Issue 4, 2012. Available: https://doi.org/10.2168/LMCS-8(4:6)2012

[23] S. J. Gay and M. Hole, "Subtyping for session types in the pi calculus," *Acta Informatica*, vol. 42, no. 2, pp. 191–225, 2005. Available: https://doi.org/10.1007/s00236-005-0177-z

[24] R. Demangeon and K. Honda, "Full abstraction in a subtyped pi-calculus with linear types," in *CONCUR '11*, J.-P. Katoen and B. König, Eds. Springer, 2011, pp. 280–296. Available: https://doi.org/10.1007/978-3-642-23217-6_19

[25] A. Scalas and N. Yoshida, "Less is more: multiparty session types revisited," *PACMPL*, vol. 3, no. POPL, pp. 30:1–30:29, 2019. Available: https://doi.org/10.1145/3290343

[26] J. Lange, E. Tuosto, and N. Yoshida, "From communicating machines to graphical choreographies," in *POPL '15*. ACM, 2015, pp. 221–232. Available: https://doi.org/10.1145/2676726.2676964

[27] F. Barbanera, M. Dezani-Ciancaglini, I. Lanese, and E. Tuosto, "Composition and decomposition of multiparty sessions," *Journal of Logical and Algebraic Methods in Programming*, vol. 119. 100620, 2021. Available: https://doi.org/10.1016/j.jlamp.2020.100620

[28] S. Ghilezan, S. Jakšić, J. Pantović, A. Scalas, and N. Yoshida, "Precise subtyping for synchronous multiparty sessions," *Journal of Logical and Algebraic Methods in Programming*, vol. 104, pp. 127–173, 2019. Available: https://doi.org/10.1016/j.jlamp.2018.12.002

[29] I. Castellani, M. Dezani-Ciancaglini, P. Giannini, and R. Horne, "Global types with internal delegation," *Theoretical Computer Science*, vol. 807, pp. 128–153, 2020. Available: https://doi.org/10.1016/j.tcs.2019.09.027

[30] L. Padovani, V. T. Vasconcelos, and H. T. Vieira, "Typing liveness in multiparty communicating systems," in *Coordination Models and Languages*, E. Kühn and R. Pugliese, Eds. Springer, 2014, pp. 147–162. Available: https://doi.org/10.1007/978-3-662-43376-8_10

[31] M. Carbone, O. Dardha, and F. Montesi, "Progress as compositional lock-freedom," in *Coordination Models and Languages*, E. Kühn and R. Pugliese, Eds. Springer, 2014, pp. 49–64. Available: https://doi.org/10.1007/978-3-662-43376-8_4

[32] S. Ghilezan, J. Pantović, I. Prokić, A. Scalas, and N. Yoshida, "Precise subtyping for asynchronous multiparty sessions," *Proc. ACM Program. Lang.*, vol. 5, 2021. Available: https://doi.org/10.1145/3434297

[33] J. Lange and E. Tuosto, "Synthesising choreographies from local session types," in *CONCUR 2012 – Concurrency Theory*, M. Koutny and I. Ulidowski, Eds. Springer, 2012, pp. 225–239. Available: https://doi.org/10.1007/978-3-642-32940-1_17

[34] P.-M. Deniélou and N. Yoshida, "Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types," in *ICALP '13*, F. V. Fomin, R. Freivalds, M. Kwiatkowska, and D. Peleg, Eds. Springer, 2013, pp. 174–186. Available: https://doi.org/10.1007/978-3-642-39212-2_18

[35] L. Padovani, "Fair subtyping for multi-party session types," *Mathematical Structures in Computer Science*, vol. 26, no. 3, pp. 424–464, 2016. Available: https://doi.org/10.1017/S096012951400022X