Coordination: Reo, Nets, and Logic*

Dave Clarke

CWI, Amsterdam, The Netherlands
dave@cwi.nl

Abstract. This article considers the coordination language *Reo*, a Petri net variant called *zero-safe nets*, and *intuitionistic temporal linear logic* (ITLL). The first part examines the semantics of the coordination language Reo in relation to zero-safe nets. Although the external presentations of the two models are quite different, the difference in underlying semantics is rather small. In fact, Reo connectors can be compositionally encoded into zero-safe nets. This means that the tools and techniques developed for Petri nets over the last 30 years, such as various extensions to the zero-safe nets model, such reconfigurable and dynamic nets, can be adapted to the Reo setting. The second part re-examines the idea of using linear logic as a basis for coordination languages. Specifically, we argue that *intuitionistic temporal linear logic* (ITLL) can encode the semantics of Reo and zero-safe nets, by encoding their notion of transaction. Moreover, by adapting the encoding and exploring the additional connectives of ITLL, it can form the basis of an expressive coordination language which goes beyond these models, by introducing means for explicitly reasoning about choices made by the environment and by providing more fine-grained control over the timing of interaction.

1 Introduction

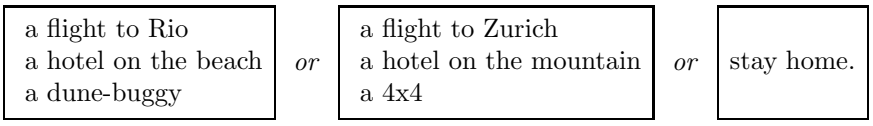
Pundits of coordination languages and models argue that concurrent, component-based software should be broken into two independent parts: the entities performing computation (components), and the entities coordinating the data flow and resource usage between the components. This was crystallised into the equation [23]:

Concurrent Programming = Computation + Coordination.

To date numerous coordination models have been proposed, each realising the philosophy in a different manner [4,37]. The ones we are interested in in this paper endeavour to schedule collections of actions together into multi-party transactions, in the sense that either some desired set of actions occur together or none of them do, possibly at the exclusion of other collections of actions. The following variant of the well-known *holiday booking example* illustrates well what we are after. Consider the following:

* This work is in the context of the EU project IST-33826 CREDO: Modeling and analysis of evolutionary structures for distributed services (http://credo.cwi.nl)

F.S. de Boer et al. (Eds.): FMCO 2007, LNCS 5382, pp. 226–256, 2008.



The *goal* is to book, with suitably matching dates, either all the entries in the first box and none of the entries in the second box, *or* all the entries in the second box and none of those in the first box, *or* nothing at all (the third box). Our assumption is that various components (or services) providing the basic functionality for booking flights, hotels, dune buggies, 4x4s, etc. exist, and the task of the coordinator is to plug these together in such a way to achieve the desired goal.

In order to discuss the kind of coordination we are interested in this paper, we first need to establish some terminology. We consider coordination not simply as a one-off event, but rather as an ongoing activity. Conceptually, time is partitioned into a number of contiguous *epochs*, which we consider as the *units of coordination*, meaning that the coordinator attempts to achieve something within each epoch, and that, from the perspective of an external observer, no finer granularity of time is observable. From the perspective of the coordinator, a lot may occur within an epoch.

We say that a set of actions A is *atomic* wrt to another set of actions B if and only if (1) either all of A occur or none of A within an epoch; and (2) the actions A are not interleaved with the actions in B within that epoch. The actions A are said to be *mutually exclusive* with the actions B. It is clear that this notion of atomic can be broken into two part. Indeed, this is how atomicity is conceived in the coordination language Reo [6]. In Reo, and in this article, the notion of *synchrony* is associated with (1), that is, a set of actions are considered *synchronous* if and only if they either all occur or none of them occurs within an epoch. The notion of *asynchrony* or *mutual exclusion* is associated with (2), that is, two sets of actions A and B are *asynchronous* if and only if if some $a \in A$ occurs within an epoch, then no element of B can occur within that epoch, and vice versa. Simply put, two events that *necessarily* occur within the same epoch are considered to be *synchronous*, and two events that *necessarily* occur within different epochs are considered to be *asynchronous*. Synchrony can be also be equated with a primitive notion of transaction.

Using this terminology, we say that the actions underlying the Rio-holiday are synchronous, the actions underlying the Zurich-holiday are synchronous, *and* the Rio-holiday is asynchronous with the Zurich-holiday.

We will work with the following definition of *coordination*:

> *coordination* is the task of (continuously/repeatedly) scheduling groups of dependent actions within temporal epochs.

In this definition, 'scheduling' captures both that actions may be scheduled to occur or to not occur—or perhaps even that they occur and are rolled-back. Furthermore, 'dependent' captures that one action may have temporal or data

dependencies on another action, so even though all actions are conceptually considered to happen at the same time (synchronously), there may be a sequential ordering among them, though only the fact that they occur within the same epoch is relevant for our purposes.

The two coordination models considered in this paper are Reo [6] and zero-safe nets [15]. These models, more or less, employ the notions defined above, coordinating by grouping together actions into synchronous epochs at the exclusion of other possibilities. In both cases, the possibilities in different epochs changes due to state changes of the primitive elements of the coordination model.

Reo. Reo is a channel-based coordination designed by Farhad Arbab, and developed extensively by his group at CWI [6]. In Reo component *connectors* are built by plugging together primitive channels.[1] In Reo, channels come in a variety of behavioural variants, but one characteristic persistent across the entire set is that some channels are synchronous, as defined above, whereas others are asynchronous; other channels admit variants of these possibilities or change their behaviour depending upon their state. Channels in Reo can be composed to express a wide range of possible behaviours. Composition occurs at *nodes*—a collection of coincident channel ends—, which consist of a number of channel ends writing data into the node, and a number of ends taking data from the node. The node selects one possible data item written to it *at the exclusion of all others*, and duplicates it to all the ends taking data *synchronously*—thus all ends must be capable of accepting the data. Nodes have the policy that they cannot buffer data, it must be passed onward. The power of Reo comes from the combination of synchronous and mutual exclusion constraints of channels and nodes, resulting in the *propagation of synchrony and exclusion* across a connector. This means, for example, that if two primitives require that their ports a, b, c and d, e, respectively, are synchronised, and they are plugged together, joining c and d, then the synchronisation propagates over the composite of the two primitives, meaning that $a, b, c(= d), e$ will all by synchronised. If action f is mutually exclusive with a, then it will be mutually exclusive with $b, c(= d)$, and e as well after the composition.

Synchronisation and mutual exclusion constraints restrict the possible ways that data can flow through a connector. As we shall see, this means that data cannot simply flow through a connector; rather, only some of the possible ways data can flow are valid. Consider the example in Figure 1(a). Primitives $1(Anpr)$, $3(moB)$, and $7(stC)$ are *replicators*. They move the data synchronously from A, m and s, respectively, to npr, oB, and tc. Primitives $2(nm)$ and $6(rs)$ are *lossy sync* channels. They move data synchronously from n and r to m and s, respectively, *or* simply lose the data at n and r respectively. Primitive $5(otq)$ is a merger. It non-deterministically chooses between moving data from o to q or from t to q. Finally, primitive $4(pq)$ is a *synchronous drain*. It will remove data from p and q synchronously. The semantics of this connector can be understood as the following 'token game': the goal is to move a token from boundary node

[1] We sometimes write *primitive* instead of *channel* to de-emphasise the requirement that they have two ends.

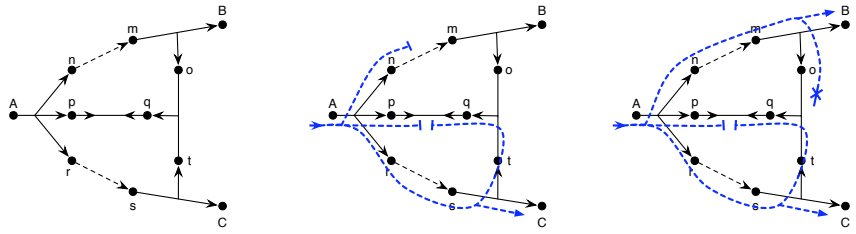


Fig. 1. (a) An exclusive router in Reo. (The numbering of primitives will be used later). (b) Valid data flow. (c) Stuck data flow.

A so that (1) moving it through the connector obeys all the above-mentioned rules imposed by the primitives, (2) no token gets stuck on an internal node $\{n, m, o, p, q, s, r, t\}$, and (3) no primitive is used more than once, so that (4) the token possibly ends up on B and/or C or is lost.

Two of the possible 'plays' are presented in Figures 1(b) and 1(c). The dashed (blue) line marks the path of the token(s) moved through the connector. In the first case, choices were made satisfying the rules of the game, and thus this corresponds to a correct behavioural possibility of the connector. In the second case, we were over-zealous and allowed the data to pass through both lossy syncs nm and rs, because the merger can choose only one item from o and t, hence the token will get stuck on, for example, o, and thus the behavioural possibility indicated (copying data from A to B and C) is invalid. After playing with the various possibilities for this connector, the conclusion should be that it is possible to move the token on A to either B or C, exclusively. This connector is in fact called an *exclusive router*.

Zero-safe Nets. Coordination in zero-safe nets [15] is achieved in a different way. Zero-safe nets are a variant of Petri nets which includes certain places called zero places. These places cannot be observed, meaning that any transition involving a marking of such a place is internal. In contrast, normal places are called *stable*. Only markings of stable places are observable. The basic firing of transitions of a zero-safe net is the same as for an ordinary Petri net. The main difference is that the semantics is expressed as two levels. The *micro*-level is more or less the same as in an ordinary Petri net, except that a micro-step may involve multiple firings. Precisely, it may involve the movement of any number of tokens into and out of zero places, but a token that is moved into a stable place may not be moved out of it again. This way a series of micro-steps makes up a so-called *macro-step*, which models a transaction between observable states. The internal transitions can be seen as synchronising the observable ones. As usual, choice in a Petri net is modelled by the competition between different transitions for the same token.

Figure 2(a) presents and example zero-safe net solving the dining philosophers problem (adapted from Bruni et. al. [15]). A token may be moved from

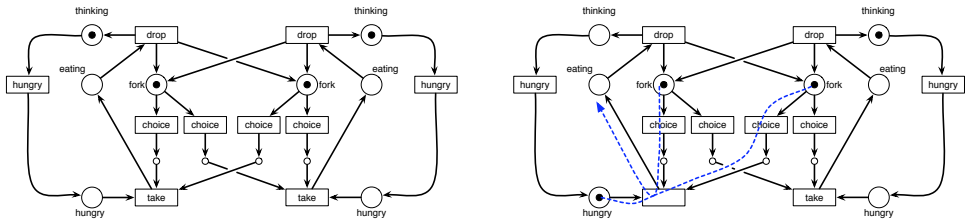


Fig. 2. (a) A Zero-safe Petri net solving the dining philosophers problem. Stable places are represented using large circles. Zero places are represented using small circles. (b) A successful transaction. Notice that it involves multiple internal steps. A stuck configuration would result, for example, if the far right hand *choice* had selected the *fork*.

fork (a stable place) to one of the zero places between the *choice* and the *take* transitions. A *take* transition will fire only if both connected zero places and the *hungry* place have tokens. The transition from having two *fork*s (on the table) to *eating* involves a number of internal transitions. Only combinations of internal transitions that do not get stuck are considered to be valid transactions. Thus, in this example, the two *choice* transitions need to *coordinate* in order to ensure that the transaction succeeds (Figure 2(b)).

This Work. On the surface, the coordination models Reo and zero-safe nets do not look particularly close. We show that they are surprisingly similar by casting their semantic models into a common, simple framework. The semantics of Reo and zero-safe nets are cast in terms of transitions between elements of a pair of monoids. The monoids respectively represent the stable parts of a configuration (states of buffers or stable places) and the internal parts of a configuration (the nodes or the zero places), and transitions represent the synchronous flow of data through Reo primitives or the firing of Petri net transitions. A common set of operational rules gives the semantics for the two models, and the difference is ultimately is in the details of the monoid used. After casting Reo's and zero-safe nets' semantics into a common semantic framework, we show that Reo connectors can be compositionally encoded into zero-safe nets. This result applies to causal Reo models which do not include any notion of context dependence, priority or maximal progress. This creates the opportunity for transferring tools, theoretical results, and implementation techniques for Petri nets to Reo.

The second part of the paper explores the idea of using intuitionistic temporal linear logic [26] as a meta-language for describing coordination models that have a transactional flavour. We then use the logic to encode both Reo and zero-safe nets. Transitions are encoded as linear formula describing the transfer of data between ports and state changes. In addition, the temporal modalities of the logic are used to control in which epochs events may, must and cannot occur. For example, $\Box m \otimes z \multimap \bigcirc \Box m' \otimes z'$, states that connector in state (or stable place) m in the presence of data on node z (or in zero place z) can by synchronously

transformed into state (or stable place) m', which cannot be used until the next step or thereafter, and data on node (or in zero place) z, which must be consumed in this epoch. The $\Box-$ modality on the m represents that connector m (or place) may or may not be used in the epoch, and the absence of such a modality on z states that it must be used within the epoch. The semantics of Reo and zero-safe nets are soundly encoded as proofs of judgements of a certain shape, which succeed in either consuming all formulæ that must be consumed within an epoch or by delaying the consumption of formulæ until another epoch. The encoding consists of encoding the common operational framework and showing how the firing of primitive elements can be encoded. A side benefit of using an intuitionistic logic is that our model of Reo does not suffer from *causality* problems which blight most other models of Reo. Finally, we describe how the additional connectors of ITLL can express coordination patterns beyond those expressible in Reo and zero-safe nets, by exploiting the game-like nature of the logic to express choices made by the environment, in contrast to the choices made by the coordinator, and by varying the use of the temporal modalities to change both the specification of the epoch in which an even occurs *and* who decides when it occurs.

Caveat. Many semantic models for Reo have been presented in the literature to address more complex classes of connectors possessing mechanisms such as context dependency [18]. In this article, we take the simplest published semantics of Reo as our basis [8], though the semantics we present does not suffer from any causality problems, so is somewhat closer in spirit to the operational semantics of Mousavi et. al. [35]. In any case, there is no natural semantics for Reo—Reo makes sense only when given a semantic model, and an encoding of channel semantics and composition in that model—so we pursue but a reasonable model of Reo.

Organization. The paper is organized as follows: Section 2 presents a unified semantic framework into which both Reo's and zero-safe nets' semantics are cast, and shows how Reo can be encoded into zero-safe nets. Section 3 describes intuitionistic temporal linear logic. Section 4 describes how both Reo and Zero-safe nets are encoded into ITLL. Section 5 explores ITLL as a coordination model. Section 6 describes related work. Section 7 concludes the paper.

2 Unified Semantic Framework

One of the goals of this paper is to compare the coordination language Reo and zero-safe nets. We do so by casting both formalisms into a common semantic framework. An appropriate, simple framework was developed by Bruni et. al. [13] for studying zero-safe nets, which extends the well-known '*Petri nets are monoids*' approach [34]. Typically, the monoids are so-called *place monoids* modelling the markings of places—essentially, the monoids are multi-sets, with multi-set union as the multiplication operation. The variant adapted for zero-safe nets considers the net as two monoids, one for stable places and one for zero places.

Here we model Reo simply by choosing different (partial) monoids as the basis of the semantics, while keeping the same operational rules as for zero-safe nets. By making the monoid partial, we are able to both place an upper bound on the placement of tokens, modelling the requirement in Reo that nodes (temporarily) have at most one token, and to track the fact that each Reo primitive can *fire* at most once per epoch.

The semantics is based on a two reduction relations, *macro-step reduction* $\Rightarrow_{\Delta} \subseteq M \times M$ and *mirco-step reduction* $\rightarrow_{\Delta} \subseteq (M \times Z) \times (M \times Z)$, between configurations capturing the state of a system, where the configurations are elements of M and $M \times Z$, respectively, and M and Z are monoids. Both relations are determined by a set of primitive firings $\Delta \subseteq (M \times Z) \times (M \times Z)$, described below. *External configurations* M represent externally observable states. *Internal configurations* in $M \times Z$ represent, in addition, the internal, unobservable states. M denotes the stable part of a configuration, which persists between epochs, and is externally observable. Z denotes the internal or unobservable part of a configuration. States in Z are used in internal steps to coordinate externally observable transitions, but do not themselves persist from one epoch to another. Thus, *macro-steps* capture what occurs from one epoch to another epoch as far as the outside world is concerned, whereas *micro-steps* capture what occurs within an epoch in order to make a macro-step happen—the coordination. One can think of the macro-step semantics as an extensional description of behaviour, and the micro-step semantics as an intensional description of behaviour.

In Reo, M records the states of the primitive connectors, whereas Z records the data flow on nodes, as the semantics of Reo do not permit data to be (observably) buffered on nodes. The chosen monoids (Section 2.2) will ensure that each primitive can be only in one state and can only fire once per epoch, and that only one data item is permitted on a node.

In zero safe nets, M records the markings of the stable places, whereas Z records the zero-safe places. Only stable places can form part of external configurations. The stable places may change at most once during each epoch. More precisely, each stable token in a zero-safe net may be moved out of a stable place and, eventually, into another stable place at most once per epoch. The zero places are used to coordinate the activities occurring within an epoch. These may be used an arbitrary number of times, and can be fired either sequentially or in parallel.

We now present a generic operational semantics which can be used to give the semantics to Reo and zero-safe nets. The semantic rules are parameterised both by two monoids and a set of transitions, $\Delta \subseteq (M \times Z) \times (M \times Z)$, describing the *firing* of primitives (such as channels and nodes in Reo and transitions in zero-safe nets). Each element of Δ is written as $(m, z)\ [\rangle (m', z')$, where $m, m' \in M$ and $z, z' \in Z$, stating that there is a transition taking internal configuration (m, z) to internal configuration (m', z'). Note that only transitions between internal configurations are provided (though z and z' may be 0, and hence not involve internal states). These form the basis for micro-steps; macro-steps are derived from them using the rules of the operational semantics.

$$\frac{(m,z)\,[\rangle(m',z') \in \Delta \quad m'' \in M \quad z'' \in Z}{(m \circ m'', z \cdot z'') \rightarrow_\Delta (m' \circ m'', z' \cdot z'')}\ (\text{FIRING}) \qquad \frac{(m_1,z_1) \rightarrow_\Delta (m_1',z_1') \quad (m_2,z_2) \rightarrow_\Delta (m_2',z_2')}{(m_1 \circ m_2, z_1 \cdot z_2) \rightarrow_\Delta (m_1' \circ m_2', z_1' \cdot z_2')}\ (\text{PARALLEL})$$

$$\frac{(m_1,z) \rightarrow_\Delta (m_1',z'') \quad (m_2,z'') \rightarrow_\Delta (m_2',z')}{(m_1 \circ m_2, z) \rightarrow_\Delta (m_1' \circ m_2', z')}\ (\text{CONCATENATION}) \qquad \frac{(m,0_Z) \rightarrow_\Delta (m',0_Z)}{m \Rightarrow_\Delta m'}\ (\text{COMMIT})$$

Fig. 3. Operational Semantics Rules. A transition is defined only when each of its constituents parts is defined. $(M, \circ, 0_M)$ and $(Z, \cdot, 0_Z)$ are monoids (Section 2.1).

Figure 3 presents the generic operational semantic rules for the micro- and macro-steps of a system represented by monoids M and Z and primitive transition relation Δ. The rule (FIRING) describes how a primitive firing embeds into the rule format. The rule also can capture parts of a configuration that do not change during a particular micro-step. The rule (PARALLEL) describes two independent micro-steps firing in parallel, in different parts of the connector/net. The rule (CONCATENATION) describes the sequential composition of two series of micro-steps. The two mirco-step sequences typically deal with different parts of the configuration. This rule acts as a coordinator for the various actions which may occur in parallel. The rule (COMMIT) describes macro-steps as a series of micro-steps starting from an external configuration and ending in an external configuration. At the beginning and end of an epoch, the Z component must equal the unit element of the monoid, meaning, in zero-safe nets, that there are no elements on zero-places, and, in Reo, that no data is buffered in nodes.

Some advantages of the monoid-based semantics are that it can define the semantics of a system in a piecemeal fashion and that it can express non-interleaved concurrency. For example, a part of a Reo connector (or zero-safe net) may be described by configuration (m_1, z_1) and another part by (m_2, z_2). The configuration $(m_1 \circ m_2, z_1 \cdot z_2)$ describes the combination of these two connector (net) parts, assuming that it is defined. If we have micro-step transitions $(m_1, z_1) \rightarrow_\Delta (m_1', z_1')$ and $(m_2, z_2) \rightarrow_\Delta (m_2', z_2')$, the transition $(m_1 \circ m_2, z_1 \cdot z_2) \rightarrow_\Delta (m_1' \circ m_2', z_1' \cdot z_2')$ may be possible in the larger connector, depending on certain conditions.

2.1 Partial Commutative Monoids

The generic semantic rules presented above are parameterized by two partial commutative monoids—we will typically just say *monoid* for *partial commutative monoid*. We now define these and present a number of example monoids used in this paper. But first, some notation.

Notation 1. *Recall that $A^B = \{f : B \rightarrow A\}$ describes the set of functions from B to A, and that $B \rightharpoonup A$ is the partial functions from B to A. If $f : B \rightharpoonup A$, we write* $\mathrm{dom}(f)$ *to be the part of B for which f is defined, i.e.,* $\mathrm{dom}(f) = \{b \in B \mid f(b)$ *is defined*$\}$.

Given a set C and a C-indexed collection of sets $Q_{c \in C}$, define the partial dependent function space $C \rightharpoonup\!\!\!\!\rightarrow Q_{c \in C}$ *to be the subset of the partial functions*

$C \rightharpoonup \biguplus_{c \in C} Q_c$ *such that if* $c \in \mathrm{dom}(f)$*, then* $f(c) \in Q_c$*. Such a function maps each defined* c *to a member of* Q_c*.*

Let $\mathcal{D}ata$ *be a non-empty set representing the data domain.*

Definition 1 (Partial Commutative Monoid). *A partial commutative monoid* $M^\circ = (M, \circ, 1)$ *consists of a set* M*, an unit element* $1 \in M$*, and a partial function* $\circ : M \times M \rightharpoonup M$*, obeying the following axioms:*

- $(a \circ b) \circ c = a \circ (b \circ c)$ *(associativity)*
- $a \circ b = b \circ a$ *(commutativity)*
- $a \circ 1 = 1 \circ a = a$ *(neutral element)*

Definition 2 (Product). *Given two monoids* $M^\circ = (M, \circ, 1_M)$ *and* $N^\cdot = (N, \cdot, 1_N)$*, their product,* $M^\circ \times N^\cdot$ *is defined as* $M^\circ \times N^\cdot = (M \times N, \bullet, 1_{M \times N})$ *where:*

- $(m, n) \bullet (m', n') = (m \circ m', n \cdot n')$*, whenever both components are defined, and undefined otherwise; and*
- $1_{M \times N} = (1_M, 1_N)$*.*

The following two monoids help define semantics for Petri nets. The first models the number of tokens on each place in the net, whereas the second models coloured tokens in coloured variants.

Definition 3 (Place Monoid). *Given a set of places* $\mathcal{P}$*, define the monoid* $PM(\mathcal{P}) = (\mathbb{N}^{\mathcal{P}}, \cdot, 0)$ *as:*

- $(p_1 \cdot p_2)(x) = p_1(x) + p_2(x)$*, and*
- $0(x) = 0$*.*

Definition 4 (Coloured Place Monoid). *Given a set of places* $\mathcal{P}$*, define the monoid* $CPM(\mathcal{P}) = (\mathcal{P} \to (\mathcal{D}ata \to \mathbb{N}), \cdot, 0)$ *with*

- $(p_1 \cdot p_2)(x) = \lambda d.(p_1(x)(d) + p_2(x)(d))$*, and*
- $0(x) = \lambda d.0$*.*

Typically, we would require that the function located at each place has finite support—meaning that each place holds a finite multiset of data items.

The following three monoids are used to define the semantic of Reo. The first two model data on nodes: the black-and-white node monoid models the case where data values have no significance (they are *signals*), whereas the coloured node monoid models the case where the data values are of interest. In both cases, elements of the monoid represent a partial mapping of nodes to data values representing the value stored on the node (or just whether it has data in the B/W case). The key point to note is that composition is undefined for two elements of the monoid that give a data value for the same node. The state monoid is similar to the coloured node monoid, in that it describes a partial function; the main difference is that each element of the domain of the function has its own codomain. The domain C is used to represent the primitives and each component Q_c of the codomain $\bigcup_{c \in C} Q_c$ represents the state set of components c. Again, only one state per connector is permitted, so the composition operation is also partial.

Definition 5 (B/W Node Monoid). *Given a set of nodes $\mathcal{N}$, define the monoid $NM(\mathcal{N}) = (2^{\mathcal{N}}, \cdot, \emptyset)$ with*

- $n_1 \cdot n_2 = \begin{cases} n_1 \cup n_2, & \text{if } n_1 \cap n_2 = \emptyset \\ \text{undefined}, & \text{otherwise} \end{cases}$

Definition 6 (Coloured Node Monoid). *Let $\mathcal{N}$ be a set of nodes. A* coloured *node monoid is a triple $CNM(\mathcal{N}) = (\mathcal{N} \rightharpoonup \mathcal{D}ata, \cdot, 0)$, where*

- $f_1 \cdot f_2 = \begin{cases} f_1 \cup f_2, & \text{if } \mathrm{dom}(f_1) \cap \mathrm{dom}(f_2) = \emptyset \\ \text{undefined}, & \text{otherwise} \end{cases}$
- 0 *is the nowhere defined function.*

Definition 7 (State Monoid). *Let C be a set, and for each $c \in C$, let Q_c be a non-empty set. Define the monoid $SM(C, Q_{c\in C}) = (C \rightharpoonup\!\!\!\!\!\rightharpoonup Q_{c\in C}, \cdot, 0)$ where:*

- $f_1 \cdot f_2 = \begin{cases} f_1 \cup f_2, & \text{if } \mathrm{dom}(f_1) \cap \mathrm{dom}(f_2) = \emptyset \\ \text{undefined}, & \text{otherwise} \end{cases}$
- 0 *is the nowhere defined function.*

2.2 Semantics of Reo

Much of the hard work has now been done to give an operational semantics to Reo connectors. We need simply to instantiate the monoids M and Z, and provide transitions for primitives. We present two variants, a 'black and white' one which ignores data and only sends signals through connectors, and a 'coloured' one which considers data as well.

Let $\mathcal{P}ort$ be a denumerable set of port names. Let $i, o, a, b, \ldots$ range over elements of $\mathcal{P}ort$ and $I, O, A, B, \ldots$ range over subsets of $\mathcal{P}ort$.

A *Reo primitive* P is a 5-tuple $P = (Q, q, I, O, \Delta)$, where Q is a set of states, $q \in Q$ is the initial state, $I, O \subseteq \mathcal{P}ort$ are the sets of input and output ports, respectively, and Δ defines the transition relation. The transitions depend upon whether the model we are interested in is black and white or coloured.

B/W. Initially, transitions will be of the form $\Delta \subseteq (Q \times 2^I) \times (Q \times 2^O)$, satisfying the requirement that if $(q, A)\,[\rangle\,(r, B) \in \Delta$, then $A \cap B = \emptyset$, for causality reasons. The transition $(q, A)\,[\rangle\,(r, B)$ states that in state q the connector can *synchronously* accept input on ports A (excluding ports $I \setminus A$), and output on ports B (excluding ports $O \setminus B$).
Now let $M = SM(\mathbf{1}, Q)$ and $Z = NM(I \cup O)$, where $\mathbf{1}$ is a one element set. It is clear that the following embeddings exists $Q \times 2^I \hookrightarrow Q \times 2^{I\cup O} \hookrightarrow M \times Z$, and similarly for $Q \times 2^O$. Thus we consider the transitions Δ in the desired format as $\Delta \subseteq (M \times Z) \times (M \times Z)$ using these embeddings.

Coloured. This time transitions have the form $\Delta \subseteq (Q \times (I \rightharpoonup \mathcal{D}ata)) \times (Q \times (O \rightharpoonup \mathcal{D}ata))$, where for $(q, f)\,[\rangle\,(r, g) \in \Delta$ we again require that $\mathrm{dom}(f) \cap \mathrm{dom}(g) = \emptyset$, for causality reasons. The transition $(q, f)\,[\rangle\,(r, g)$ states that in state q the connector can *synchronously* accept input on ports $\mathrm{dom}(f)$ (excluding ports $I \setminus \mathrm{dom}(f)$), and output on ports $\mathrm{dom}(g)$ (excluding ports

$O \setminus \mathrm{dom}(g)$). Partial functions f and g describe the actual data that flows. Now let $M = SM(\mathbf{1}, Q)$ and $Z = CNM(I \cup O)$. Given the embeddings $Q \times (I \rightharpoonup \mathcal{D}ata) \hookrightarrow Q \times (I \cup O \rightharpoonup \mathcal{D}ata) \hookrightarrow M \times Z$, and similarly for O, we can consider the transitions Δ in the desired format as $\Delta \subseteq (M \times Z) \times (M \times Z)$ using these embeddings.

Figure 4 presents the firing relations for a number primitives. These consist of Reo's selection of channels, mergers and replicators (to model Reo's n-m nodes), and takers and writers, to model boundary interaction. We assume that the channels are defined over port set $\{a, b\}$, and that mergers and replicators are defined over port set $\{a, b, c\}$.

Primitive	**Arity**	**States**	**Init.**	**B/W Trans.**	**Coloured Trans.**
Sync	$\{a\} \to \{b\}$	$\{\circ\}$	$\circ$	$(\circ, a)\,[\rangle\,(\circ, b)$	$(\circ, a \mapsto d)\,[\rangle\,(\circ, b \mapsto d)$
SyncDrain	$\{a, b\} \to \emptyset$	$\{\circ\}$	$\circ$	$(\circ, ab)\,[\rangle\,(\circ, \mathbf{1})$	$(\circ, a \mapsto d + b \mapsto d')\,[\rangle\,(\circ, \mathbf{1})$
SyncSpout	$\emptyset \to \{a, b\}$	$\{\circ\}$	$\circ$	$(\circ, \mathbf{1})\,[\rangle\,(\circ, ab)$	$(\circ, \mathbf{1})\,[\rangle\,(\circ, a \mapsto d + b \mapsto d')$
AsyncDrain	$\{a, b\} \to \emptyset$	$\{\circ\}$	$\circ$	$(\circ, a)\,[\rangle\,(\circ, \mathbf{1})$ $(\circ, b)\,[\rangle\,(\circ, \mathbf{1})$	$(\circ, a \mapsto d)\,[\rangle\,(\circ, \mathbf{1})$ $(\circ, b \mapsto d)\,[\rangle\,(\circ, \mathbf{1})$
AsyncSpout	$\emptyset \to \{a, b\}$	$\{\circ\}$	$\circ$	$(\circ, \mathbf{1})\,[\rangle\,(\circ, a)$ $(\circ, \mathbf{1})\,[\rangle\,(\circ, b)$	$(\circ, \mathbf{1})\,[\rangle\,(\circ, a \mapsto d)$ $(\circ, \mathbf{1})\,[\rangle\,(\circ, b \mapsto d)$
LossySync	$\{a\} \to \{b\}$	$\{\circ\}$	$\circ$	$(\circ, a)\,[\rangle\,(\circ, b)$ $(\circ, a)\,[\rangle\,(\circ, \mathbf{1})$	$(\circ, a \mapsto d)\,[\rangle\,(\circ, b \mapsto d)$ $(\circ, a \mapsto d)\,[\rangle\,(\circ, \mathbf{1})$
FIFO1	$\{a\} \to \{b\}$	$\{\circ, \bullet\}$	$\circ$	$(\circ, a)\,[\rangle\,(\bullet, \mathbf{1})$ $(\bullet, \mathbf{1})\,[\rangle\,(\circ, b)$	$(\circ, a \mapsto d)\,[\rangle\,(\bullet(d), \mathbf{1})$ $(\bullet(d), \mathbf{1})\,[\rangle\,(\circ, b \mapsto d)$
Merger	$\{a, b\} \to \{c\}$	$\{\circ\}$	$\circ$	$(\circ, a)\,[\rangle\,(\circ, c)$ $(\circ, b)\,[\rangle\,(\circ, c)$	$(\circ, a \mapsto d)\,[\rangle\,(\circ, c \mapsto d)$ $(\circ, b \mapsto d)\,[\rangle\,(\circ, c \mapsto d)$
Replicator	$\{a\} \to \{b, c\}$	$\{\circ\}$	$\circ$	$(\circ, a)\,[\rangle\,(\circ, bc)$	$(\circ, a \mapsto d)\,[\rangle\,(\circ, b \mapsto d + c \mapsto d)$

Fig. 4. Some Reo Primitives. ab denotes the set $\{a, b\}$. $a \mapsto d + b \mapsto d'$ is a function mapping a to d and b to d'.

A *Reo connector* is simply a set of primitives $P = (Q_n, q_n, I_n, O_n, \Delta_n)_{n \in N}$, where N is used to name the primitives, subject to the conditions (1) for all $a \in \mathcal{P}ort$, if $a \in I_n$ and $a \in I_m$, for $n, m \in N$, then $n = m$, and, similarly, if $a \in O_n$ and $a \in O_m$, for $n, m \in N$, then $n = m$; and (2) for all $a \in \mathcal{P}ort$, if $a \in I_n$ then there is at most one $m \neq n$ such that $a \in O_m$, and if $a \in O_n$, then there is at most one $m \neq n$ such that $a \in I_m$. Condition (1) ensures that the input/output ports are unique for each primitive, and condition (2) ensures that the ports are primitive ports are plugged together 1:1, output to input—that is, if $a \in O_n \cap I_m$, then output port a of n is plugged into input port a of m.

The semantics of a Reo connector is given by taking the two monoids $M = SM(N, Q_{n \in N})$ and $Z = NM(\bigcup_{n \in N}(I_n \cup O_n))$ (or $Z = CNM(\bigcup_{n \in N}(I_n \cup O_n))$ for coloured models) and the transition relation $\Delta = \bigcup_{n \in N} \Delta_n$, where each primitive firing $(q, A)\,[\rangle\,(r, B)$ (or $(q, f)\,[\rangle\,(r, g)$) is embedded in the obvious fashion.

Figure 5 gives an example derivation for the connector presented in Figure 1.

$$\dfrac{\dfrac{(1,A)\,[\rangle\,(1,npr)}{(13,A)\to(13,npr)}\,\textsc{(Firing)} \quad \dfrac{\dfrac{\dfrac{(2,n)\,[\rangle\,(2,0)}{(2,npr)\to(2,pr)}\,\textsc{(Firing)} \quad (*)}{(2567,npr)\to(2567,pqC)}\,\textsc{(Concat)} \quad \dfrac{(4,pq)\,[\rangle\,(4,0)}{(4,pqC)\to(4,C)}\,\textsc{(Firing)}}{(24567,npr)\to(24567,C)}\,\textsc{(Concat)}}{(1234567,A)\to(1234567,C)}\,\textsc{(Concat)}$$

where (*) is

$$\dfrac{\dfrac{\dfrac{(6,r)\,[\rangle\,(6,s)}{(6,pr)\to(6,ps)}\,\textsc{(Firing)} \quad \dfrac{(7,s)\,[\rangle\,(7,tC)}{(7,ps)\to(7,ptC)}\,\textsc{(Firing)}}{(67,pr)\to(67,ptC)}\,\textsc{(Concat)} \quad \dfrac{(5,t)\,[\rangle\,(5,q)}{(5,ptC)\to(5,pqC)}\,\textsc{(Firing)}}{(567,pr)\to(567,ptC)}\,\textsc{(Concat)}$$

Fig. 5. Derivation for Exclusive Router (Figure 1). The primitives are all stateless, so the M (first) component really just records which primitives are used. The Z (second) component records the data flow.

2.3 Zero-Safe Nets

We directly give the semantics of zero-safe nets in terms of monoids, borrowing from (a compressed version of) the development of Bruni et. al. [15].

Definition 8 (Zero-safe net). *A* zero-safe net *is a tuple* $N = (S, T, F, u_{\text{in}}, \mathsf{Z})$ *where*

- S *is the set of* places, $a, a', \ldots$;
- T *is the set of* transitions $t, t', \ldots$ *(with* $S \cap T = \emptyset$*);*
- $F \subseteq (S \times T) \cup (T \times S)$ *is called the* flow relation*; the elements of the flow relation are called* arcs*, and we write* $x\ F\ y$ *for* $(x, y) \in F$*;*
- *a finite multiset* $u_{\text{in}} : S \to \mathbb{N}$ *is the initial marking of* N*; and*
- *the set* $\mathsf{Z} \subseteq S$ *is the set of* zero places*.*

The places $\mathsf{M} = S \setminus \mathsf{Z}$ *are called the* stable places*. A* stable marking *is a multiset of stable places, and the initial marking* u_{in} *must be stable.*

A *marking* for a Petri net with places S is a map $S \to \mathbb{N}$ indicating the number of tokens at each place—this can equally be seen as a multiset of places. Each transition, $t \in T$, connects some places ${}^\bullet t = \{s \in S \mid s\ F\ t\}$ with some places $t^\bullet = \{s \in S \mid t\ F\ y\}$. Transitions generate a firing relation $\Delta \subseteq (S \to \mathbb{N}) \times (S \to \mathbb{N})$ such that $f\,[\rangle\,g \in \Delta$ if and only if

$$f(s) = \begin{cases} 1, & \text{if } s \in {}^\bullet t \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad g(s) = \begin{cases} 1, \text{ if } s \in t^\bullet \\ 0, \text{ otherwise} \end{cases}$$

for some $t \in T$. This could be easily extended to deal with weighted transitions.

We can view markings in $S \to \mathbb{N}$ equivalently as functions from $(\mathsf{M} \times \mathsf{Z}) \to \mathbb{N}$ or as $(\mathsf{M} \to \mathbb{N}) \times (\mathsf{Z} \to \mathbb{N})$, giving the stable and the zero markings. Thus the firing relation $\Delta \subseteq (S \to \mathbb{N}) \times (S \to \mathbb{N})$ can trivially be seen as a relation

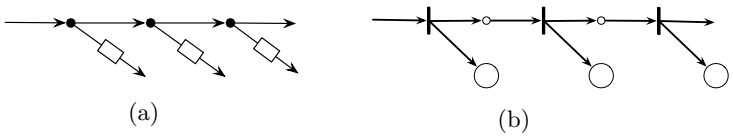


Fig. 6. (a) Synchronous chain 'coordinating' actions (FIFO-buffers) (b) Again in zero-safe nets

$\Delta \subseteq ((\mathsf{M} \to \mathbb{N}) \times (\mathsf{Z} \to \mathbb{N})) \times ((\mathsf{M} \to \mathbb{N}) \times (\mathsf{Z} \to \mathbb{N}))$, which is a relation between the product of two place monoids, namely, $PM(\mathsf{M}) \times PM(\mathsf{Z})$, so, finally, $\Delta \subseteq (PM(\mathsf{M}) \times PM(\mathsf{Z})) \times (PM(\mathsf{M}) \times PM(\mathsf{Z}))$. Thus, the two monoids underlying the semantic model for zero-safe nets are $M = PM(\mathsf{M})$ and $Z = PM(\mathsf{Z})$, where, as mentioned above, M are the stable places and Z are the zero places. The remainder of the operational semantics is given by the rules in Figure 3.

It is an easy exercise to work out the derivations corresponding to the behaviour demonstrated in Figure 2.

A coloured variant can be easily described, simply by replacing $PM(M) \times PM(Z)$ by $CPM(M) \times CPM(Z)$ and redefining the firing relation Δ.

2.4 Comparision of Reo and Zero-Safe Nets

On the surface, Reo and zero-safe nets look quite different, but delving a little deeper into their semantics reveals that they are in fact very similar in many respects. In fact, we can encode Reo into zero-safe nets.

- Both models enforce a notion of transaction in that valid, externally observable macro-steps consist of a sequence of internal micro-steps, and that not all possible sequences of micro-steps result in a valid macro-step.
- Reo's nodes are akin to zero-safe places and its primitives' states are stable, that is, they are part of the observable configuration. A FIFO buffer behaves similarly to a (bounded/1-safe) place in a Petri net.
- Choice in Reo is handled by merger and primitives such as asynchronous drains (which can in fact be encoded using a merger). Choice in Petri nets is handled by multiple transitions from a place competing for the token.
- Replication in Reo is handled by replicators. Replication in Petri nets is handled by multiple target places of a transition.
- A chain of synchronous channels must fire sequentially at the micro level, within an epoch. This is very similar to how a chain of zero-safe places would fire sequentially. Replication (via replicators or Petri transition) can then be used to coordinate activities, respecting the sequential ordering. Parallel reductions occur in Reo as in zero-safe nets in independent parts of a connector. In both cases, the topology of the connector/net guides the sequential/parallel micro-step reduction. Figure 6 illustrates the idea for both Reo and zero-safe nets.

- The semantics of Reo primitives are expressed externally as a set of transitions, whereas the nature of zero-safe networks is predefined, based on the variant of Petri-net under consideration.
- The major difference between the two is that in any macro-step (epoch), a Reo primitive can participate at most once. Equally, in Reo data may flow through each node at most once in an epoch. On the other hand, a zero-safe net permits each transition to fire an arbitrary number of times in an epoch (though this is easy to control, as we show below).
- In zero-safe nets, places can hold an arbitrary number of tokens. Reo nodes (equivalent to zero-safe places) can hold only one token. Variants of Petri nets exist where places hold at most one token (called 1-safe). Such a variant is closer to Reo.

In order to establish a correspondence between the two models, in one direction at least, we need to demonstrate how to restrict zero places so that they can be used at most once per epoch. This is quite simple, and is illustrated in Figure 7. The stable place is used to ensure that the zero place can fire only once, as the token leaves the stable place when the token enters the zero place, and re-enters the stable place when the token leaves the zero place. After these two steps, the stable place can no longer participate in the epoch.

Figure 8 presents the encoding of each of Reo's primitives into zero safe nets. Each encoded connector uses the above trick to ensure that it fires only once per epoch. The labelled zero places represent the boundary/interface ports of the connectors. Composition of two encoded Reo connectors consists of superimposing the two zero places corresponding to the boundary nodes—that is, the two Reo ends plugged together to form a node are encoded as the same zero place.

Encoding more general Reo primitives based on their semantic description is straightforward, as we simple follow the patterns used in the encodings above. The encoding of FIFO1 demonstrates how to encode a state machine, the encodings of AsyncDrain and LossySync demonstrate how to encode choice, and the encodings of the Sync* demonstrate how to encode synchronisation. Thus, we argue, Reo can be compositionally encoded in terms of zero-safe nets by encoding Reo's primitives, and then composing them.

Now that Reo can be encoded into zero-safe nets, extensions to the zero-safe model such as dynamic and reconfigurable nets [13] can be applied in the context of Reo. Exploring these directions is left for future work. Indeed, reconfiguration of Reo connectors, in particular, when the reconfiguration is caused by dynamic behaviour in the connector, is already an active area of research [17,31].

In the other direction, encoding zero-safe nets into Reo is not possible in general. This is simply because places in zero-safe nets may hold more than one token, and thus zero-safe nets can model multiple interactions on a given port within a single epoch.

Finally, observe that the Reo and zero-safe nets could easily be combined into a single model, simply by taking the product of both their normal and zero components, and by adding non-trivial additional firing rules to link the Reo

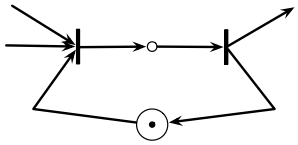


Fig. 7. Ensuring that zero places fire at most once per epoch

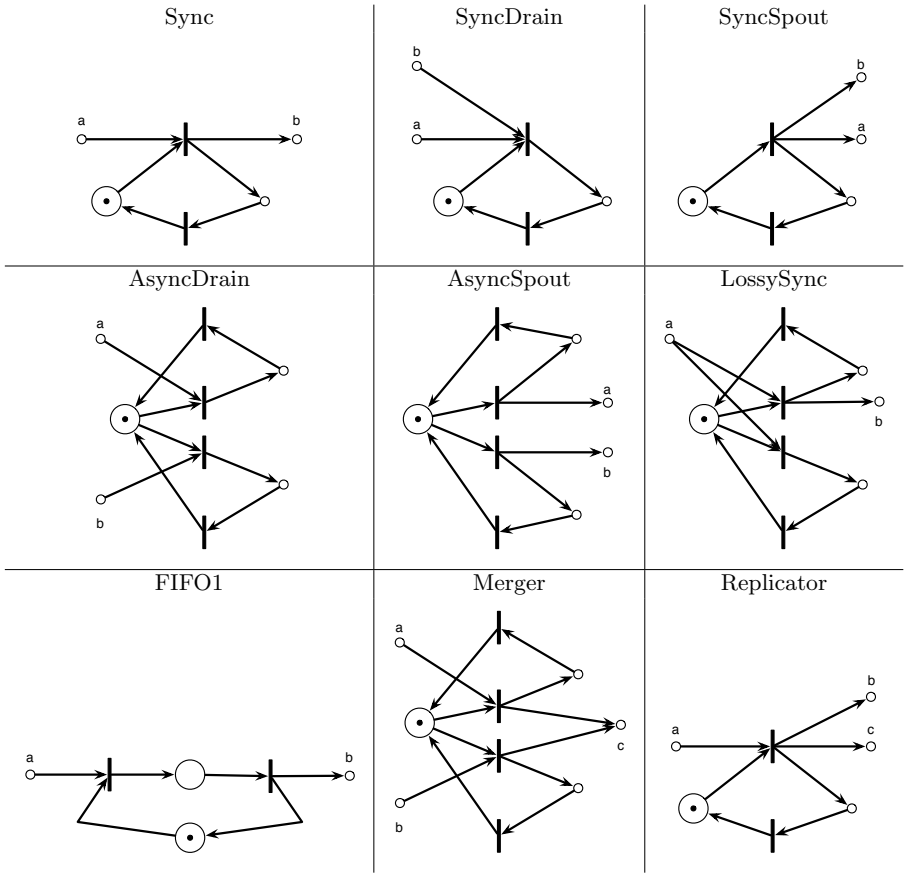


Fig. 8. Zero-safe net Encodings of Standard Reo Primitive. Labelled places represent Reo's ports (ends). Other places are used to ensure that the primitive fires at most once per epoch, plus the buffer for FIFO1. (Compare behaviours with those described in Figure 4).

connector and the zero-safe net, that is, by having transitions from $M_{Reo} \times M_{zsn}$ to $Z_{Reo} \times Z_{zsn}$. This approach is at present being explored to connect Reo and workflow language YAWL [41].

3 Intuitionistic Temporal Linear Logic

We now shift focus away from the Reo and zero-safe nets, and introduce intuitionistic temporal linear logic (ITLL)[2] both as a meta-language for reasoning about the sort of coordination models we are interested in, and as a vehicle for enabling more refined behavioural descriptions beyond those expressible in Reo or zero-safe nets. For example, neither of these models can make the distinction between internal and external choice, whereas in ITLL the formula $A \,\&\, B$ expresses an internal choice, and formula $A \oplus B$ expresses external choice.

Intuitionistic temporal linear logic (ITLL) is an extension of linear logic [24], explored by Hirai [26], that includes linear variants of the temporal modalities $\Box-$, $\bigcirc-$, and $\Diamond-$. ITLL is a resource conscious logic with modalities for reasoning about how resource usage changes over time. ITLL has been used to reason about timed Petri nets [25], for modelling agent choices, agent interaction [38,39], and agent negotiation [33], and as a logic programming language [9].

For now, we will consider the following fragment of ITLL (additional connectives will be discussed later):

$$A ::= a \;\mid\; \mathbf{1} \;\mid\; A \otimes A \;\mid\; A \multimap A \;\mid\; A \,\&\, A \;\mid\; A \oplus A \;\mid\; \Box A \;\mid\; \bigcirc A$$

where a ranges over primitive formulæ. In general, we use lower case letters to range over primitive formulæ and upper case letters to range over formulæ.

Our interpretation of ITLL is that formulæ can describe epochs of time, where within each epoch, formulæ describe resource usage in a linear fashion. Ultimately, we associate coordination (or ability-to-coordinate) with the ability to perform proofs of a certain form. Thus, we base this interpretation on a standard reading of proofs. The standard understanding of the game semantics underlying linear logic formulæ [12], adapted to account for the temporal connectives, has guided our interpretation. The connectives can be read as follows:

a	availability of single resource a, *now*
$\mathbf{1}$	no resources, *now*
$A \otimes B$	availability of (resources) A and B, *now*
$A \multimap B$	availability of a one-time converter of (resources) A to B, *now*
$A \,\&\, B$	an internal choice between (resources) A and B, *now*
$A \oplus A$	an external choice between (resources) A and B, *now*
$\Box A$	(resource) A is available *any time*, exactly once
$\bigcirc A$	(resource) A is available in the *next* epoch, exactly once

It is important to note that $\Box A$ does not mean the same as its linear temporal logic (LTL) counterpart. To be compatible with linear logic, Hirai linearised (in the sense of linear logic) the meaning of $\Box A$: *A can be used **any time**, but only once.* Note that we have no exponentials ($!A$), though we do suggest how they

[2] A distinguishing feature of intuitionistic linear logic is that judgements allow only a single conclusion. Furthermore, it lacks the *par* connective among others which require multiple conclusions.

can be used as an alternative in our encodings. Further interpretation of ITLL from a coordination perspective will be given in Section 5.

The proof rules for ITLL are presented in Figure 9. Let Γ denote a possibly empty sequence of formulæ. Let $\Box\Gamma = \Box A, \Box B, \ldots$. Similarly for $\bigcirc\Gamma$. Rules are labelled to indicate whether the connective is introduced on the left or right of the turnstile $\vdash$, so, for example, $\multimap_l$ labels the rule introducing $\multimap$ on the left.

$$\frac{}{A \vdash A}\, Ax \qquad \frac{\Gamma \vdash A \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash B}\, Cut \qquad \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}\, Ex$$

$$\frac{\Gamma \vdash A \quad B, \Delta \vdash C}{\Gamma, A \multimap B, \Delta \vdash C}\, \multimap_l \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\, \multimap_r$$

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\, \otimes_l \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\, \otimes_r$$

$$\frac{\Gamma, A \vdash C}{\Gamma, A \,\&\, B \vdash C}\, \&_{1l} \qquad \frac{\Gamma, B \vdash C}{\Gamma, A \,\&\, B \vdash C}\, \&_{2l} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}\, \&_r$$

$$\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \oplus B \vdash C}\, \oplus_l \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B}\, \oplus_{1r} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B}\, \oplus_{2r}$$

$$\frac{\Gamma \vdash C}{\Gamma, \mathbf{1} \vdash C}\, \mathbf{1}_l \qquad \frac{}{\vdash \mathbf{1}}\, \mathbf{1}_r$$

$$\frac{A, \Gamma \vdash B}{\Box A, \Gamma \vdash B}\, \Box_l \qquad \frac{\Box\Gamma' \vdash A}{\Box\Gamma' \vdash \Box A}\, \Box_r \qquad \frac{\Box\Gamma, \Gamma'' \vdash A}{\Box\Gamma', \bigcirc\Gamma'' \vdash \bigcirc A}\, \bigcirc$$

Fig. 9. The proof rules for intuitionistic temporal linear logic

The first five rows of rules are standard. From a coordination perspective, the computation interpretation of the rules $\&_l$ and $\oplus_l$ reveals the difference between the two choice connectives. $\&_l$ will enable a proof using $A \,\&\, B$ to go through even if only one of A or B is suitable. The interpretation is the the prover (the coordinator) can choose which. On the other hand, $\oplus_l$ requires that both A and B in $A \oplus B$ can make the proof go through. The coordinator needs to be able to produce an appropriate proof in both cases for coordination to succeed, modelling that the choice can be made externally. This is consistent with existing interpretations of linear logic [1].

Rule $\Box_l$ enables a resource to be used at any time ($\Box A$) to be used *now* (A). Rule $\Box_r$, which we do not use directly, states that a result that depends only on resources that can be used at any time can itself be used anytime. Rule $\bigcirc$ states that reasoning about future epochs (such as the next one) is done by shifting the future (next state) resources to the present state (removing the $\bigcirc$) and then reasoning as per usual. This requires that there are no *now* resources, only $\Box-$ and $\bigcirc-$ ones.

In the encodings of Reo and zero-safe nets presented later in this section, the proof rules for ITLL are used as the workhorse handling details of the semantic

framework, whereas the specific firing rules for the primitives are encoding as externally specified axioms. Let Σ denote a set of axioms of the form $\Gamma \vdash A$. We write $\Gamma \vdash_\Sigma A$ to denote judgements in the proof system extended with axioms Σ and the following rule:

$$\frac{(\Gamma \vdash A) \in \Sigma}{\Gamma \vdash_\Sigma A}\ \Sigma\text{-}axiom$$

Note that proving judgement $\Pi \vdash_\Sigma B$ is equivalent to proving $\Sigma^*, \Pi \vdash B$ in the logic extended with the standard axioms for exponentials along with adaptations of the proof rules above (see [26]), where Σ^* converts each $(\Gamma \vdash A) \in \Sigma$ into the formula $!(\Gamma^\otimes \multimap A)$, and $\Gamma^\otimes$ is the formulæ in Γ connected by an $\otimes$. Thus, if $\Gamma = A_1, \ldots, A_n$, then $\Gamma^\otimes = A_1 \otimes \cdots \otimes A_n$, and if $\Gamma = \epsilon$, then $\Gamma^\otimes = \mathbf{1}$.

3.1 Semantics of ITLL

Hirai [25] presented semantics of ITLL as an adaptation of the original phase space semantics of Girard [24]. No game semantics exists for ITLL. Nonetheless, we can (and have) drawn from game semantics for linear logic when giving our intuitive interpretation of ITLL's connectives. The open question remains: what are the game semantics for full ITLL. Here we provide only some intuition, leaving the complete treatment of game semantics for ITLL for future work.

In games semantics for linear logic [12], one telling feature is the difference between the choice connectives $- \& -$ and $- \oplus -$ is who makes the choice. In the first case, the player needs to only win one of the possible games encoded in the choice in order to win the game. In the second case, the player can only win the game if he can win both games encoded in the choice, as the opponent makes the choice. To extend to deal with the temporal modalities, we anticipate the following changes. Games are played over multiple epochs. $\bigcirc A$ is interpreted as a game that is played in the following epoch. $\Box A$ is a game for which the player (coordinator) can chose which epoch the game corresponding to A is played in — the moves are thus, 'play now' and 'play later', when cast in terms of a game played 'now'. Similarly, $\Diamond A$ is the dual, which means that the opponent (environment) makes the choice when to play the game. Finally, for a player to win a game (= valid/provable judgement), she needs to win every epoch (until the game peters out).

4 Encodings

After investing the effort to cast the semantics of Reo and zero-safe nets into a uniform framework, encoding them into ITLL is quite easy: we need to encode the general framework and then the primitives for each system. The basic assumption we run with is that the elements of the respective monoids can be encoded as primitive formulæ or tensor products ($\otimes$) of such formuæ. We typically do

not insert explicit conversions between the elements of the monoid and their representation as formulæ. We demonstrate the soundness of our encoding, but do not give a completeness result. There are two reasons: (1) completeness simply would not hold for the entire class of monoids we consider, though we expect that it will hold for monoids which are not partial and are sufficiently free; and (2) we do not know whether a cut elimination result holds for ITLL, yet this seems to be crucial for proving completeness.

4.1 Encoding Configurations

The encoding of a configuration depends whether it is on the left or right hand side of the arrow.

Given a reduction $(m, z) \rightarrow_{\Delta} (m', z')$, where $m, m' \in M$ and $z, z' \in Z$, the configuration on the left-hand side is encoded as $(\Box m)^{\otimes} \otimes z^{\otimes}$, (or equivalently, as a sequent of form $(\Box m), z$, without the separating $\otimes$), and the configuration on the right-hand side is encoded as $(\bigcirc\Box m')^{\otimes} \otimes (z')^{\otimes}$. Note that the primitive firings themselves are encoded slightly differently, as we shall see in Section 4.2

The reading of $(\Box m)^{\otimes} \otimes z^{\otimes}$ is that (1) any of the elements in m can be used at any time, and (2) all of the elements in z must be used in this step. Similarly $(\bigcirc\Box m')^{\otimes} \otimes (z')^{\otimes}$ states, in addition, that (3) in the next step, but not now, any of the elements of m' can be used at any time. The part involving z' is analogous to (2).

When encoding the left and right hand sides of $m \Rightarrow_{\Delta} m'$, the encodings can be simplified to $(\Box m)^{\otimes}$ and $(\bigcirc\Box m')^{\otimes}$.

4.2 Encoding a Primitive Firing

Each primitive firing $(m, z)\,[\rangle\,(m', z') \in \Delta$ is encoding as an axiom as follows:

$$[\![(m, z)\,[\rangle\,(m', z')]\!] = m, z \vdash (\bigcirc\Box m')^{\otimes} \otimes (z')^{\otimes}$$

Proofs for a system with primitive firing relation Δ are carried out in the logic $\Gamma \vdash_{\Sigma} A$, where $\Sigma = [\![\Delta]\!]$.

Example 1. The transition from a Reo replicator $(\circ, a)\,[\rangle\,(\circ, bc)$ would be encoded as an axiom: $\circ, a \vdash \bigcirc\Box\circ \otimes b \otimes c$.

An alternative approach is to encode the state-machines as a single formula, using an exponential:

$$\begin{aligned} [\![\Delta]\!] &= !([\![(m_1, z_1)\,[\rangle\,(m_1', z_1')]\!] \,\&\, \cdots \,\&\, [\![(m_n, z_n)\,[\rangle\,(m_n', z_n')]\!]) \\ &\qquad \text{where } \Delta = \{(m_1, z_1)\,[\rangle\,(m_1', z_1'), \ldots, (m_n, z_n)\,[\rangle\,(m_n', z_n')\} \\ [\![(m, z)\,[\rangle\,(m', z')]\!] &= m^{\otimes} \otimes z^{\otimes} \multimap (\bigcirc\Box m')^{\otimes} \otimes (z')^{\otimes} \end{aligned}$$

In this format, the above example would be expressed as $!(\circ \otimes a \multimap \bigcirc\Box\circ \otimes b \otimes c)$.

4.3 Encoding the Semantic Rules into ITLL

We now show how to encode the semantic rules from Figure 3 into ITLL proof fragments. This is the essence of soundness. In our encodings, we will use the exchange rule implicitly on both sides of the turnstile, which implies an implicit use of Cut. We also use $-^*$ to mark multiple applications of a rule or pattern.

Firing. The rule:

$$\frac{(m,z)\,[\rangle\,(m',z') \in \Delta \quad m'' \in M \quad z'' \in N}{(m \circ m'', z \cdot z'') \to_\Delta (m' \circ m'', z' \cdot z'')} \text{(FIRING)}$$

is encoded as proof:

$$\dfrac{\dfrac{\dfrac{(m, z \vdash_\Sigma (\bigcirc\Box m')^\otimes \otimes (z')^\otimes) \in \Sigma}{m, z \vdash_\Sigma (\bigcirc\Box m')^\otimes \otimes (z')^\otimes}\,\Sigma\text{-AXIOM}}{\Box m, z \vdash_\Sigma (\bigcirc\Box m')^\otimes \otimes (z')^\otimes}\,\Box_l^* \quad \dfrac{(1)}{\Box m'' \vdash_\Sigma (\bigcirc\Box m'')^\otimes} \quad \dfrac{(2)}{z'' \vdash_\Sigma (z'')^\otimes}}{\Box m, \Box m'', z, z'' \vdash_\Sigma (\bigcirc\Box m')^\otimes (\bigcirc\Box m'')^\otimes \otimes (z')^\otimes \otimes (z'')^\otimes}\,\otimes_r^*$$

where (1) is multiple occurrences of the following proof, one for each atom m_a, joined using $\otimes_r$:

$$\dfrac{\dfrac{}{\Box m_a \vdash_\Sigma \Box m_a}\,\text{Ax}}{\Box m_a \vdash_\Sigma \bigcirc\Box m_a}\,\bigcirc$$

and (2) is similar, but simpler.

Parallel. The rule:

$$\frac{(m_1, z_1) \to_\Delta (m_1', z_1') \quad (m_2, z_2) \to_\Delta (m_2', z_2')}{(m_1 \circ m_2, z_1 \cdot z_2) \to_\Delta (m_1' \circ m_2', z_1' \cdot z_2')} \text{(PARALLEL)}$$

is encoded as proof fragment:

$$\frac{\Box m_1, z_1 \vdash_\Sigma (\bigcirc\Box m_1')^\otimes \otimes (z_1')^\otimes \qquad \Box m_2, z_2 \vdash_\Sigma (\bigcirc\Box m_2')^\otimes \otimes (z_2')^\otimes}{\Box m_1, \Box m_2, z_1, z_2 \vdash_\Sigma (\bigcirc\Box m_1')^\otimes \otimes (\bigcirc\Box m_2')^\otimes \otimes (z_1')^\otimes \otimes (z_2')^\otimes}\,\otimes_r$$

Concatenation. The rule:

$$\frac{(m_1, z) \to_\Delta (m_1', z'') \quad (m_2, z'') \to_\Delta (m_2', z')}{(m_1 \circ m_2, z) \to_\Delta (m_1' \circ m_2', z')} \text{(CONCATENATION)}$$

is encoded as proof fragment:

$$\dfrac{\Box m_1, z \vdash_\Sigma (\bigcirc\Box m_1')^\otimes \otimes (z'')^\otimes \qquad \dfrac{\Box m_2, z'' \vdash_\Sigma (\bigcirc\Box m_2')^\otimes \otimes (z')^\otimes}{\Box m_2, (z'')^\otimes \vdash_\Sigma (\bigcirc\Box m_2')^\otimes \otimes (z')^\otimes}\,\otimes_l^*}{\Box m_1, \Box m_2, z \vdash_\Sigma (\bigcirc\Box m_1')^\otimes \otimes (\bigcirc\Box m_2')^\otimes \otimes (z')^\otimes}\,\text{Cut}$$

Commit. The rule:

$$\frac{(m,0) \rightarrow_{\Delta} (m',0)}{m \Rightarrow_{\Delta} m'} \text{(COMMIT)}$$

is encoded as proof fragment:

$$\dfrac{l_r\ \dfrac{}{\vdash \mathbf{1}} \qquad \dfrac{\Box m, \mathbf{1} \vdash_{\Sigma} (\bigcirc\Box m')^{\otimes} \otimes \mathbf{1} \qquad \dfrac{\dfrac{\dfrac{}{(\bigcirc\Box m')^{\otimes} \vdash_{\Sigma} (\bigcirc\Box m')^{\otimes}}\text{Ax}}{(\bigcirc\Box m')^{\otimes}, \mathbf{1} \vdash_{\Sigma} (\bigcirc\Box m')^{\otimes}}\mathbf{1}_l}{(\bigcirc\Box m')^{\otimes} \otimes \mathbf{1} \vdash_{\Sigma} (\bigcirc\Box m')^{\otimes}}\otimes_l}{\Box m, \mathbf{1} \vdash_{\Sigma} (\bigcirc\Box m')^{\otimes}}\text{Cut}}{\Box m \vdash_{\Sigma} (\bigcirc\Box m')^{\otimes}}\text{Cut}$$

An interesting aspect of this encoding is that the main coordination rule (CONCATENATION) corresponds to *Cut*. Cut elimination in proof theory often is thought of as communication or interaction. Not all proofs go through, however, so coordination is more accurately thought of as *constrained interaction*, as suggested by Wegner [43].

The result we have established is the soundness of our encoding. Specifically, we have the following:

Theorem 1. *Let* $\Sigma = [\![\Delta]\!]$.

1. *If* $(m,z) \rightarrow_{\Delta} (m',z')$, *then* $\Box m, z \vdash_{\Sigma} (\bigcirc\Box m')^{\otimes} \otimes (z')^{\otimes}$.
2. *If* $m \Rightarrow_{\Delta} m'$, *then* $\Box m \vdash_{\Sigma} (\bigcirc\Box m')^{\otimes}$.

More generally, we have show that ITLL is expressive enough to encode both Reo and zero-safe nets. In the next section, we demonstrate through example that ITLL is more expressive.

5 Coordination via ITLL

We have thus far presented, in very general sense, how ITLL can encode the coordination patterns expressible in Reo and in zero-safe nets. Doing so explored only a fraction of ITLL's expressiveness. We now further explore ITLL, mainly focussing on the various pieces of the coordination puzzle, such as the behaviour or protocol of parties involved in coordination, and on variants of how coordination is done in Reo and in zero-safe nets, in particular, with regard to the role of epochs. We assume in addition that two parties are involved in a particular interaction: the *coordinator*, whose choices are controllable, and the *environment* (or *the rest*), whose choices are uncontrollable.[3] After presenting a number of

[3] We ignore the data flowing through connectors/nets—thus we are working in the 'black-and-white' models. Adding colour raises no difficult issues. For example, the filter over a binary domain that accepts only zeros is modelled by: $\Sigma = \{\mathit{Filter} \otimes a(0) \vdash b(0) \otimes \bigcirc\Box \mathit{Filter},\ \mathit{Filter} \otimes a(1) \vdash \bigcirc\Box \mathit{Filter}\}$, where $a(0)$ means that datum 0 is on node a.

variations, we pull all the pieces together into a SoC protocol which would form the essence of a coordinator for the holiday booking problem presented in the introduction.

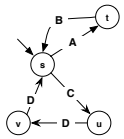
Choice. ITLL can express choice in two different ways: choices made by the coordinator and choices made by the environment. (This is well-known, and is particularly well exemplified in the game semantics of linear logic [12].) Formulæ of the form $A \,\&\, B$ express a choice between A and B that can made by the coordinator, whereas formulæ of the form $A \oplus B$ express that the choice between A and B is made by the environment.

Progress of Time. ITLL can also express the occurrence of events within epochs. Recapping, a formula of the form A without a preceding modality expresses that A must be dealt with in the current epoch. $\Box A$ expresses an action that the coordinator *may* choose to deal with A in the current epoch. $\bigcirc A$ states that A must be dealt with in next epoch. Combinations of these are possible. For example, $\bigcirc\Box A$ states that the coordinator *may* deal with A in some future epoch (but not now), and $\Box\bigcirc A$ states that the coordinator may deal with A in a future epoch, though the decision must be made one epoch in advance. $\bigcirc^7\Box\bigcirc^7 A$ states that any time after 7 epochs the coordinator may decide, 7 epochs in advance, to deal with A.

Note that a dual to $\Box-$ exists [26]: $\Diamond A$ states that the environment decides *some time* when A must be dealt with.[4] $\Diamond A$ behaves very much like $\Box A$, except that the choice is made by the environment instead of the coordinator.

State Machines (Revisited). An encoding of state machines was hinted at above. It is in fact an adaptation of an existing encoding [29, for example]. The difference here is that we use the modalities to control when transitions may or may not be taken.

An example state machine, in the encoding described above, with 4 states (s, t, u, v) and 4 external actions $(\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D})$ is encoded as axioms:[5]



$$\Sigma = \left\{ \begin{array}{ccc} s \multimap (\mathsf{A} \otimes \bigcirc\Box t \,\&\, \mathsf{C} \otimes \bigcirc\Box u), & & t \multimap \mathsf{B} \otimes \bigcirc\Box s, \\ & u \multimap \mathsf{D} \otimes \bigcirc\Box v, & \\ & & v \multimap \mathsf{D} \otimes \bigcirc\Box s \end{array} \right\}$$

The current state of such an automaton is recorded as a formula of the form $\Box s$,

[4] The rules for $\Diamond-$ introduction and elimination are:

$$\frac{\Box\Gamma, A \vdash \Diamond B}{\Box\Gamma, \Diamond A \vdash \Diamond B}\,\Diamond \to \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \Diamond A}\,\to \Diamond$$

[5] There are a number of equivalent ways of presenting our axioms. For example, we could write $(s \vdash t \,\&\, u) \in \Sigma$ or $(\mathbf{1} \vdash s \multimap t \,\&\, u) \in \Sigma$ or even $(\mathbf{1} \vdash s \multimap t), (\mathbf{1} \vdash s \multimap u) \in \Sigma$ or $(s \vdash t), (s \vdash u) \in \Sigma$. We typically choose the single formula representation $s \multimap t \,\&\, u$.

meaning that a transition from s can be taken at a time chosen by the coordinator. This particular automaton implements the regular expression $(\mathsf{AB} \mid \mathsf{CDD})^*$ over events, where each event occurs in a different epoch. The fact that the events occur in different epochs is given by the form of the next-state formulæ, namely, $\bigcirc\Box s$. Hence, the automaton is disabled after performing a transition in any given epoch, but it is re-enabled in the next epoch. All Reo primitives would have state machines of this form.

We can change the formulæ describing the state machine behaviour in a number of ways:

- The next state formula can be modified from $\bigcirc\Box s$ to $\Box s$, giving a transition such as $t \multimap \mathsf{B} \otimes \Box s$, to model that the primitive in state t, after making a transition to state s, may optionally perform an additional transition from state s in the given epoch.
- The next state formula can be simply s, giving a transition such as $t \multimap \mathsf{B} \otimes s$, which means that the primitive, after moving from state t to state s, would be forced to perform another transition in this epoch.
  (Applying the two previous the two previous variations to the semantics of a Reo primitive would give primitives that could perform interactions involving more than one step within a given epoch [20]. We will give an example shortly.)
- The choice in transition $s \multimap (\mathsf{A} \otimes \bigcirc\Box t \,\&\, \mathsf{C} \otimes \bigcirc\Box u)$ is determined by the coordinator, as $- \,\&\, -$ is used. An alternative is to use $- \oplus -$, as follows $s \multimap (A \otimes \bigcirc\Box t \oplus C \otimes \bigcirc\Box u)$, meaning that the choice is made externally. This can model, for example, interaction with an external component.

Naturally, other variations are possible.

Boundary Interactions. In Reo, a component interacts with a connector by issuing a write or a take (read) to a port, which is subsequently satisfied by the connector (unless the writer/taker times-out). We can model these in ITLL in a number of ways (showing only a writer, as a taker is quite similar; and ignoring time-out):

1. A Writer has two states $\{\mathit{Writing}, \mathit{NotWriting}\}$ and the following axioms:

$$\mathit{Writing} \multimap \mathsf{a} \otimes \bigcirc(\Box\,\mathit{Writing} \oplus \mathit{NotWriting})$$
$$\mathit{NotWriting} \multimap \bigcirc(\Box\,\mathit{Writing} \oplus \mathit{NotWriting})$$

   Notice that the next state formula $\Box\,\mathit{Writing} \oplus \mathit{NotWriting}$ forces the environment to choose whether the component will be writing or not. When the component is writing, coordinator can determine whether to allow the write to succeed when it chooses. When the component is not writing, the decision whether to write must be made again in the following epoch, determined by the second axiom.
   The initial state is represented by $\Box\,\mathit{Writing} \oplus \mathit{NotWriting}$, forcing the environment to make a choice in the first epoch.

If we were also considering data, the formula stating that data flows on channel a, namely a, would be replaced by $\bigoplus_{d \in \mathcal{D}ata} \mathsf{a}(d)$ to denote that there is a value on the channel and that it is externally selected.

2. An alternative encoding of a Writer exploits $\Diamond -$ to handle the fact that the decision to perform a write is determined by the environment, after which the coordinator decides when the write succeeds (via $\Box -$):

$$\begin{array}{l} \mathit{Writing} \multimap \mathsf{a} \otimes \bigcirc \mathit{NotWriting} \\ \mathit{NotWriting} \multimap \Diamond\Box\, \mathit{Writing} \end{array}$$

In this case, the initial state is $\mathit{NotWriting}$, which is effectively $\Diamond\Box\, \mathit{Writing}$.

The mode of interaction between the coordinator and the environment need not be restricted to that which is assumed by Reo. For example, interacting by making method calls to standard objects, or by calling web services, are equally useful alteratives, yet distinct from Reo's. In these circumstances the interaction originates with the coordinator, not the environment.

A service/method that takes input described by A and produces output described by B can be modelled by a formula as simple as $A \multimap B$. This, however, makes the assumptions that service is guaranteed to produce a result and that the coordinator must wait for the result. A more asynchronous invariant is modelled by the formula $A \multimap \Diamond B$ or even $A \multimap \Diamond\Box B$. The choice of which model to adopt depends upon the desires of the system builder; the choice of ITLL formula to model the mode of interaction determines the dependency of the coordination on the environment.

Other modes of interaction can be (at least partially) described using ITLL—one way message sends, compulsory interactions, interrupts. A complete study of ITLL's expressiveness and limitations would be very interesting.

User Interaction. ITLL formulæ can capture the essence of synchronous user interaction, which, when combined with the extended Reo we are hinting at, permits coordination of multi-action events that include user interaction, such as obtaining the user's approval of a particular holiday package. A simple formula expressing user interaction is:

$$\mathit{User} \otimes \mathsf{Q} \multimap (\mathsf{Yes} \oplus \mathsf{No}) \otimes \bigcirc\Box\, \mathit{User}$$

User is the (single, uninteresting) state. When asked a question on port Q, a reply can be supplied on either port Yes or port No, where the choice is made by the environment (or user). The question is asked and its answer is given all in the same epoch.

The test to see whether a coordinator can handle both the Yes and No answers is whether certain judgements (of the shape described in Theorem 1) are provable. When the proof does not go through, this means that during coordination-time a choice can be made that the coordinator cannot handle. Roll-back or some sort of recovery would be required when this occurs.

It is worth taking pause to consider whether this kind of user interaction can be handled in Reo. The short answer is that it cannot. External components in

Reo perform either a single write or take in each epoch—this is the only kind of event that the coordinator has to deal with. Any (inter)action that involves multiple steps must be dealt with over multiple epochs in Reo, even though the steps conceptually belong to the same action. Existing implementations of Reo cannot handle a channel whose choices are made externally, within a single epoch (though an attempt was made in the implementation Reolite [16]). Thus the ability to perform multi-step actions, possibly involving external choices, within an epoch goes beyond what Reo can do. Yet, in our opinion, these possibilities makes better use of the abstractions (synchrony and asynchrony) provided by Reo, by enabling more interesting interaction to occur during an atomic step.

A Service-Oriented Computing Protocol. We pull together the ideas presented in this section to give a more realistic example, which ties in with our original holiday booking example. The left hand side of the Figure 10 presents a connector whose job is to monitor a transaction (adopting a simplified version of the WS-Transaction protocol [22]), so that it can be incorporated into a larger Reo connector. An informal description of its behaviour is given by the 'automaton' on the right hand side of Figure 10. This automaton is initiated by a *try* action (try to perform transaction), which results, eventually, in a *success* or *fail* action, the choice of which is made by the environment. Afterwards, the transaction can be *reset* (which also causes a *no* action) or, in the case that it is successful, *committed* (which also causes a *yes* action). The choice of whether to *reset* or *commit* is made by the coordinator.

The key here is that success/failure is externally determined. In order to have this controlled by Reo (or a zero-safe net), we make all steps occur within a single epoch, which would, for example, allow the combination of two or more WC-Transactors to form proper transactions which occur *atomically* within a synchronous slice of time (*i.e.,* epoch).

The ITLL encoding consists of the following formulæ, where the initial state is *Init*: the nodes marked with bold font are on boundary with the user, whereas the others are used to interact with the underlying transaction

$$\Sigma = \left\{ \begin{array}{ll} \mathit{Init} \otimes \mathbf{try} \multimap t, & \mathit{ps} \otimes \mathsf{commit} \multimap \mathbf{yes} \otimes \bigcirc\Box \mathit{Init}, \\ t \multimap s \oplus f, & \mathit{ps} \otimes \mathsf{reset} \multimap \mathbf{no} \otimes \bigcirc\Box \mathit{Init}, \\ s \multimap \mathsf{success} \otimes \mathit{ps}, & \mathit{pf} \otimes \mathsf{reset} \multimap \mathbf{no} \otimes \bigcirc\Box \mathit{Init}, \\ f \multimap \mathsf{fail} \otimes \mathit{pf} & \end{array} \right\}$$

A more compressed variant would replace the $t \multimap s \oplus f$, $s \multimap \mathsf{success} \otimes \mathit{ps}$, and $f \multimap \mathsf{fail} \otimes \mathit{pf}$ by $t \multimap (\mathsf{success} \otimes \mathit{ps}) \oplus (\mathsf{fail} \otimes \mathit{pf})$.

The Reo-like connector presented in Figure 11(a) combines two WC-Transactors so that either both transactions are committed or neither are. The choice $\bigotimes$ in the middle selects whether to commit or reset both WC-Transactors, giving priority to committing (indicated by !).[6] The simple connector in Figure 11(b) enforces that either exactly one **yes** (using the merger) or two **no**s

[6] Priority was not discussed in this paper. Our work on 3-colouring deals with this fairly effectively [18]. A full treatment of priority in Reo, especially in the present setting, is a whole new kettle of worms that we prefer to leave on the back-burner.

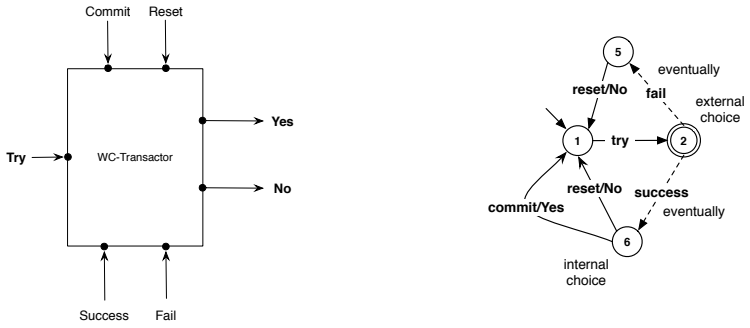


Fig. 10. Coordinator for simplified model of WS-TRANSACTION protocol. There are 7 ports to this coordinator, grouped in the following logical units: **try** initiates part of a transaction; **success & fail** are the results reported by the component being controlled; **commit & reset** commits to the result (only on success), or resets (either on success or failure); and **yes & no** reports whether the transaction succeeded and is committed to.

result, thus ensuring that either one holiday package is booked or none is. This example could be extended with data and user interaction to gain the user's approval of the chosen holiday, but clearly more work would be required to make it completely realistic. In any case, the present example cannot handled by Reo as it exists now, unless the transaction handling is implemented over multiple epochs, and thus failing to fully exploit synchrony.

6 Related Work

Many semantic models for Reo have been presented [7,8,18,35], though all of them, apart from the operational semantics of Mousavi et. al. [35], suffer from causality problems. This problem is avoided in operational models such as the one presented here. The problem is that most models have a non-constructive, classical semantics which would give a semantics to certainly loops stating that data flows in the loop, even though the loop has no source of data. Only the connector colouring model [18] deals with priority. Our semantics is superior to Mousavi et. al.'s on the following counts: the semantics of primitives such as channels are *not* built into our rules, thus we have only 4 rules compared to 16; we more cleanly deal with the treatment of data on nodes and the fact that data flow may occur only in a part of connector; and, perhaps most importantly, our rule format enables an easier comparison with existing systems. On the other hand, Mousavi et. al.'s semantics have a globally defined notion of *maximal progress* for filtering possible behaviours, which we do not consider here.

Many, many variants of Petri nets exists, far too many to review here. Bruni et. al. [13] show how the zero-place approach can be applied to coloured, reconfigurable, and dynamic nets. Given the close correspondence between Reo and zero-safe nets, it would be be fruitful for the development of Reo to transfer these

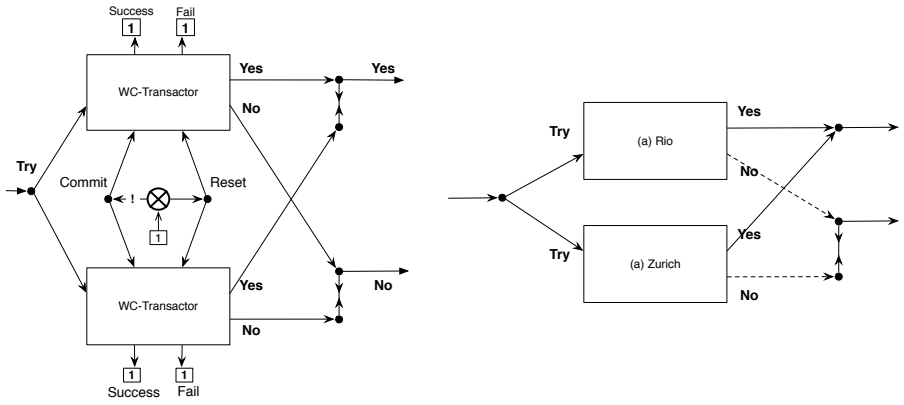


Fig. 11. (a) Combining Two WC-Transactors. The **1** connectors either always offer or accept a signal, depending upon the direction of the arrow. $\otimes$ denotes an exclusive router, which makes a choice of sending the data to exactly one of its outputs. ! indicates priority. A three-way variant of this can be constructed to implement the coordination for each of the holiday options (ignoring the actual data). (b) The connector combines two instances of the connector in (a), enforcing that either exactly one **yes** or two **no** results, thus ensuring that either one holiday package is booked or none is.

ideas into Reo. More generally, Bruni et. al. [14] present a modular high-level account of the relationship between, in our terms, the mirco- and macro-step levels, demonstrating that the naturalness of the two-level approach to describing transactions.

A large volume of literature considers the relationship between Linear Logic and Petri nets (and other concurrency models) [21,10,42,28, among others]. We adapted a simple encoding of Petri nets to use the temporal modalities of ITLL to model epochs. The closest work to ours, in this respect, is Hirai's thesis and subsequent article [26,25]. Hirai reasons about Timed Petri nets [11] (a model wherein transitions have a waiting time during which they are unusable), whereas we encode zero-safe nets (and Reo), which have a more transactional flavour. A different study of Petri nets uses the logic of bunched implications (BI) [36,40]. From a semantic perspective, it makes a lot of sense to adapt this approach to our domain, as one semantic model of BI involves partial commutative monoids. We would be interested to see what BI could bring to our application domain.

Hirai [26,25] also briefly considers variants of process encodings into ITLL. Specifically, he observes that $\bigcirc\Box-$ can be used to model synchronous message passing, which, he claims, is not expressible in linear logic. Specifically, he compares the pair of linear logic 'processes' $!(p \multimap (m \otimes p))$ and $!(q \multimap (m \multimap q))$ with the ITLL processes $!(p \multimap (m \otimes \bigcirc\Box p))$ and $!(q \multimap (m \multimap \bigcirc\Box q))$. In both cases, the first process sends an m message to the second one, repeatedly. In the LL encoding, the m is sent asynchronously, as both parts of $m \otimes p$ proceed in parallel. In contrast, m is sent and received in the ITLL encoding, as the recursive call to p cannot occur until the next step. Even though we are using the same

ingredients, our encoding (or our reading) is quite different, as we equate synchrony with what occurs (necessarily) within some epoch. In fact, we would say that the first case expresses entirely synchronous processes—that is, their entire development occurs within an epoch—whereas the second expresses synchrony for message m, asynchronously with any subsequent behaviour. Because of the different granularity, we typically exploit the parallelism inherent in $-\otimes-$, rather than worry about the asynchrony it introduces.

Küngas [33] and Pham et. al. [38] uses temporal linear logic to model the choices of agents and agent negotiation. Both articles make use of the well-known the distinction between internal $(-\,\&\,-)$ and external $(-\oplus-)$ choice. Formalæ express agent choices and time-sliced resource usage (such as printers). Küngas also models a kind of promise by formula such as $\bigcirc A \multimap B$, meaning that the offer to do A in the next step is traded for B now. Küngas uses partial evaluation (via a technique called *partial deduction*) to give more efficient programs (applicable to the ideas presented here), but also to derive *offers*, so that agents can negotiate the use of resources. Pham et. al. includes reductions corresponding to the choices made by the various choice makers. This feature would be useful in our setting to express the dynamics in the models we considered beyond Reo and zero-safe nets. Both papers only use the $\bigcirc-$ modality, whereas we suggest uses for the whole spectrum.

Kanovich and Ito [30] present an alternative (and earlier) formulation of ITLL. Hirai [26] extends Kanovich and Ito's formulation with the modal storage operator (that is, the exponential !), and removes possible infinite derivations which were at odds with the notion of *passage of time*—specifically, Zeno's paradox could arise.

ITLL has also been used as the basis for a logic programming language [9]. The main goal was to give a more fine-grained control over resources than is possible in linear logic programming languages, and to enable complex data structures which are not entirely available all the time, and thus can be the target for more efficient implementations.

Linear logic has been used as the basis for the coordination language LO (Linear Objects) [5]—though this can equally be seen as a way of combining logic and object-oriented programming—and, its successor, CLF (the Coordination Language Facility) [3]. Both languages have a very strong logic programming flavour and rely on a notion of transaction, so are close in spirit to our logical encoding. Their base language, however, does not have temporal modalities, so they cannot make as fine-grained distinctions as we can.

Several authors [32, for example] have used linear logic to analyse and synthesise planning problems. This corresponds to our intuition that in both Reo and in zero-safe networks one cannot just send the data optimistically through channels or push tokens through the network and hope to arrive at a suitable (external) configuration. Either backtracking, planning or constraint solving [19] is required.

Alexiev [2] presents a now-dated overview of many applications of Linear Logic. One of the key points that he makes is that "everything is connected

to everything else." This is certainly made no less true by the present article. Kamide [27] presents linear and affine logics extended with temporal, spatial and epistemic modalities. This looks like very fertile ground for exploring further extensions to coordination language, perhaps even taking on a more agent-like character.

7 Conclusion and Future Work

In this article, we cast both the coordination language Reo and zero-safe nets into a common semantic framework. The key difference between the two models is the choice of monoids used in the framework, demonstrating that, although the two models differ significantly on the surface, they are very similar underneath. In fact, Reo can be compsitionally encoded into zero-safe nets. We then encoded both systems into *intuitionistic temporal linear logic* (ITLL), to explore its applicability as unifying framework for coordination languages. We illustrated this by encoding both Reo and zero-safe nets into ITLL. The meaning of the logical formulæ used in the encodings provided useful intuition into the meaning of configurations and reductions in the respective semantics. In both cases, the reading corresponds to our intuitive understanding of the language being modelled.

In the latter part of the paper, we explored the remainder of ITLL, and demonstrated, mostly through examples, the range of possible behaviour that can be expressed in ITLL, but in neither Reo nor zero-safe nets. In particular, we described more complex *per epoch* behaviour, choices attributable to different players (the coordinator vs. the environment), and optional and compulsory actions. None of these distinctions can be made in Reo or in zero-safe nets.

This work merely scratches the surface of an interesting and useful connection between Reo, zero-safe nets, and ITLL. We hope that this paper will serve as a source of ideas for further developing Reo and the coordination languages that will follow it. A detailed game-semantic study of ITLL is warranted in order to fully elucidate the ideas presented here. In particular, such a model would provide not only a semantics for Reo, but of all the extensions described in Section 5. Additional ideas we would like to investigate include increasing the expressiveness to cover notions such as *co-operation*, wherein the actual possibilities for a choice may be limited by both the coordinator and the environment, but the actual choice is determined by both; the distinction between reversible and irreversible actions; groups and operations on them; and many more.

References

1. Abramsky, S.: Computational interpretations of linear logic. Theor. Comput. Sci. 111(1-2), 3–57 (1993)
2. Alexiev, V.: Applications of linear logic to computation: An overview. Logic Journal of IGPL 2(1), 77–107 (1994)
3. Andreoli, J.-M., Freeman, S., Pareschi, R.: The Coordination Language Facility: coordination of distributed objects. Theory and Practice of Object Systems 2(2), 77–94 (1996)

4. Andreoli, J.-M., Hankin, C., Le Metayer, D. (eds.): Coordination Programming: Mechanisms, Models and Semantics. Imperial College Press (1996)
5. Andreoli, J.-M., Pareschi, R.: Linear objects: logical processes with built-in inheritance. New Generation Computing 9 (1991)
6. Arbab, F.: Reo: a channel-based coordination model for component composition. Math. Struct. in Comp. Science 14(3), 329–366 (2004)
7. Arbab, F., Rutten, J.J.M.M.: A coinductive calculus of component connectors. In: Wirsing, M., Pattinson, D., Hennicker, R. (eds.) WADT 2003. LNCS, vol. 2755, pp. 34–55. Springer, Heidelberg (2003)
8. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in Reo by constraint automata. Science of Computer Programming 61(2), 75–113 (2006)
9. Banbara, M., Kang, K.-S., Hirai, T., Tamura, N.: Logic programming in a fragment of intuitionistic temporal linear logic. In: Codognet, P. (ed.) ICLP 2001. LNCS, vol. 2237, pp. 315–330. Springer, Heidelberg (2001)
10. Bellin, G., Scott, P.J.: On the pi-calculus and linear logic. Theoretical Computer Science 135, 11–65 (1994)
11. Bestuzheva, I.I., Rudnev, V.V.: Timed Petri nets: Classification and comparative analysis. Automation and Remote Control 51(10), 1308–1318 (1990)
12. Blass, A.: A game semantics for linear logic. Annals of Pure and Applied Logic 56, 151–156 (1992)
13. Bruni, R., Melgratti, H.C., Montanari, U.: Extending the zero-safe approach to coloured, reconfigurable and dynamic nets. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 291–327. Springer, Heidelberg (2004)
14. Bruni, R., Meseguer, J., Montanari, U.: Tiling transactions in rewriting logic. In: WRLA 2002, Rewriting Logic and Its Applications, Electronic Notes in Theoretical Computer Science, vol. 71, pp. 90–109 (April 2004)
15. Bruni, R., Montanari, U.: Zero-safe nets: Comparing the collective and individual token approaches. Information and Computation 156(1-2), 46–89 (2000)
16. Clarke, D.: Reolite implementation (December 2005), `http://www.cwi.nl/~dave/reolite`
17. Clarke, D.: A basic logic for reasoning about connector reconfiguration. Fundam. Inform. 82(4), 361–390 (2008)
18. Clarke, D., Costa, D., Arbab, F.: Connector colouring I: Synchronisation and context dependency. Science of Computer Programming 66(3), 205–225 (2007)
19. Clarke, D., Proença, J., Lazovik, A., Arbab, F.: Deconstructing Reo. In: FOCLASA 2008, ENTCS (July 2008) (to appear)
20. Diakov, N., Arbab, F.: Adaptation of software entities for synchronous exogenous coordination: An initial approach. In: Proceedings of The Second International Workshop on Coordination and Adaptation of Software Entities, W-CAT 2005 (July 2005)
21. Engberg, U.H., Winskel, G.: Linear logic on Petri nets. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) REX 1993. LNCS, vol. 803, pp. 176–229. Springer, Heidelberg (1994)
22. Cabrera, L.F., et al.: Web Services Atomic Transaction (WS-AtomicTransaction). MSDN Library (November 2004)
23. Gelernter, D., Carriero, N.: Coordination languages and their significance. Commun. ACM 35(2), 97–107 (1992)
24. Girard, J.-Y.: Linear logic. Theoretical Computer Science 50, 1–102 (1987)

25. Hirai, T.: Propositional temporal linear logic and its application to concurrent systems. EICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences (Special Section on Concurrent Systems Technology) E83-A(11), 2219–2227 (2000)
26. Hirai, T.: Temporal Linear Logic and Its Application. PhD thesis, The Graduate School of Science and Technology, Kobe University, Japan (September 2000)
27. Kamide, N.: Linear and affine logics with temporal, spatial and epistemic logics. Theoretical Computer Science 252, 165–207 (2006)
28. Kanovich, M.I.: Linear logic as a logic of computations. Annals of Pure and Applied Logic 67(1–3), 183–212 (1994)
29. Kanovich, M.I.: Linear logic automata. Annals of Pure and Applied Logic 78, 147–188 (1996)
30. Kanovich, M.I., Ito, T.: Temporal linear logic specifications for concurrent processes (extended abstract). In: Twefth Annual IEEE Symposium on Logic in Computer Science, pp. 48–57 (1997)
31. Koehler, C., Costa, D., Proença, J., Arbab, F.: Reconfiguration of Reo connectors triggered by dataflow. In: Electronic Communications of the EASST: Graph Transformation and Visual Modeling Techniques, vol. 10 (2008)
32. Küngas, P.: Analysing AI planning problems in linear logic – a partial deduction approach. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS, vol. 3171, pp. 52–61. Springer, Heidelberg (2004)
33. Küngas, P.: Temporal linear logic for symbolic agent negotiation. In: Zhang, C., W. Guesgen, H., Yeap, W.-K. (eds.) PRICAI 2004. LNCS, vol. 3157, pp. 23–32. Springer, Heidelberg (2004)
34. Meseguer, J., Montanari, U.: Petri nets are monoids. Information and Computation 88, 105–155 (1990)
35. Mousavi, M.R., Sirjani, M., Arbab, F.: Formal semantics and analysis of component connectors in Reo. Electronic Notes in Computer Science 154(1), 83–99 (2006)
36. O'Hearn, P.W., Yang, H.: Petri net semantics of bunched implications(October 1999) (unpublished) (available from Peter's webpage)
37. Papadopoulos, G.A., Arbab, F.: Coordination models and languages. In: Zelkowitz, M. (ed.) The Engineering of Large Systems. Advances in Computers, vol. 46, pp. 329–400. Academic Press, London (1998)
38. Pham, D.Q., Harland, J., Winikoff, M.: Modelling agent's choices in temporal linear logic. In: Baldoni, M., Son, T.C., van Riemsdijk, M.B., Winikoff, M. (eds.) DALT 2007. LNCS, vol. 4897, pp. 140–157. Springer, Heidelberg (2008)
39. Pham, D.Q., Harland, J.: Temporal linear logic as a basis for flexible agent interactions. In: Durfee, E.H., Yokoo, M., Huhns, M.N., Shehory, O. (eds.) 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), IFAAMAS (2007)
40. Pym, D.J.: The Semantics and Proof Theory of the Logic of Bunched Implications. Applied Logic Series, vol. 26. Kluwer Academic Publishers, Dordrecht (2002)
41. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet another workflow language. Information Systems 30(4), 245–275 (2005)
42. Watkins, K., Cervasato, I., Pfenning, F., Walker, D.: A concurrent logical framework: The propositional fragment. In: Berardi, S., Coppo, M., Damiani, F. (eds.) TYPES 2003. LNCS, vol. 3085, pp. 355–377. Springer, Heidelberg (2004)
43. Wegner, P.: Coordination as constrained interaction. In: Ciancarini, P., Hankin, C. (eds.) COORDINATION 1996. LNCS, vol. 1061, pp. 28–33. Springer, Heidelberg (1996)