ReLo: a Dynamic Logic to Reason About Reo Circuits*

Erick Grilo
Instituto de Computação
Universidade Federal Fluminense
simas_grilo@id.uff.br

Bruno Lopes
Instituto de Computação
Universidade Federal Fluminense
bruno@ic.uff.br

Critical systems require high reliability and are present in many domains. They are systems in which failure may result in financial damage or even loss of lives. Standard techniques of software engineering are not enough to ensure the absence of unacceptable failures and/or that critical requirements are fulfilled. Reo is a component-based modelling language that aims to provide a framework to build software based on existing pieces of software, which has been used in a wide variety of domains. Its formal semantics provides grounds to certify that systems based on Reo models satisfy specific requirements (i.e., absence of deadlocks). Current logical approaches for reasoning over Reo require the conversion of formal semantics into a logical framework. *ReLo* is a dynamic logic that naturally subsumes Reo's semantics. It provides a means to reason over Reo circuits. This work extends *ReLo* by introducing the iteration operator, and soundness and completeness proofs for its axiomatization.The core aspects of this logic are also formalized in the Coq proof assistant.

1 Introduction

In software development, service-oriented computing [31] and model-driven development [6] are examples of techniques that take advantage of software models. The first technique advocates computing based on preexisting systems (services) as described by Service-Oriented Architecture (SOA), while the latter is a development technique that considers the implementation of a system based on a model. A model is an abstraction of a system (or some particular portion of it) in a specific language, which will be used as a specification basis for the system's implementation. It can be specified in languages such as Unified Modeling Language (UML) or formal specification languages like B [1] or Alloy [16]. Researchers also have applied approaches such as formal methods in software development to formalize and assure that certain (critical) systems have some required properties [19, 30].

Reo [2] is a prominent modelling language, enabling coordination of communication between interconnected systems without focusing on their internal properties. Reo models are compositionally built from base connectors, where each connector in Reo stands for a specific communication pattern. Reo has proven to be successful in modeling the organization of concurrent systems' interaction, being used in a variety of applications, from process modeling to Web-Services integration [4] and even in the construction of frameworks to verify specifications in Reo [22, 34].

Reo's ability to model communication between software interfaces has also attracted research on verification of Reo circuits, resulting in many different formal semantics [17] like automata-based models [3, 7, 23], coalgebraic models [2], Intuitionistic Logic with Petri Nets [12] (to name a few), and some of their implementations [22, 33, 35, 26, 29, 36, 23]. However, as far as the authors are concerned, there is no logic apart from *ReLo* [13] to specific reason about Reo models naturally, where the usage of other logic-based approaches requires conversion between different formal semantics.

*This work was supported by CNPq and FAPERJ.

P. Fontaine, D. Nantes-Sobrinho (Eds.):
Logical and Semantic Frameworks with Applications 2022 (LSFA'22)
EPTCS 376, 2023, pp. 16–33, doi:10.4204/EPTCS.376.4

This work extends *ReLo* [13] by introducing an iteration operator and the soundness and completeness proofs of its axiomatic system. A prototypical implementation of this framework in Coq proof assistant, enabling the verification of properties of Reo programs in *ReLo* within a computerized environment is available at http://github.com/frame-lab/ReoLogicCoq.

This work is structured as follows. Section 3 discusses briefly a related logic formalism with the one hereby proposed and introduces Reo modelling language, along with some examples. Section 4 discuss *ReLo*'s main aspects, from its core definitions (such as language, models, transitions firing) and its soundness and completeness proofs. Finally, Section 5 closes the work by discussing the obtained results and assessing possible future work.

2 Related Work

The fact that Reo can be used to model many real-world situations has attracted attention from researchers all around the world, resulting in a great effort directed in formalizing means to verify properties of Reo models [18, 32, 20, 22, 28, 27, 17]. Such effort also resulted in the proposal of several formal semantics for this modelling language [17], varying from operational semantics to coalgebric models.

One of the most known formal semantics for Reo consists of Constraint Automata [8], an operational semantic in which Reo connectors are modelled as automata for TDS-languages [5]. It enables reasoning over the data flow of Reo connectors and when they happened. Constraint Automata have been extended to some variants which aim to enrich the reasoning process by capturing properties like the timing of the data flows or possible actions over the data, respectively as Timed Constraint Automata [22] and Action Constraint Automata [21]. Some of them are briefly discussed below, along with other formal semantics for Reo.

The approach presented by Klein et al. [18] provides a platform to reason about Reo models using Vereofy,[1] a model checker for component-based systems, while Pourvatan et al. [32] propose an approach to reason about Reo models employing symbolic execution of Constraint Automata. Kokash & Arbab [20] formally verify Long-Running Transactions (LRTs) modelled as Reo connectors using Vereofy, enabling expressing properties of these connectors in logics such as Linear Temporal Logic (LTL) or a variant of Computation Tree Logic (CTL) named Alternating-time Stream Logic (ASL). Kokash et al. [22] use mCRL2 to encode Reo's semantics in Constraint Automata and other automata-based semantics, encoding their behaviour as mCRL2 processes and enabling the expression of properties regarding deadlocks and data constraints which depend upon time. mCRL2 also supports model-checking of Reo in a dynamic logic (with fixed points), where modalities are regular expressions, atomic actions are sets of nodes that fire at the same time. Mouzavi et al. [28] propose an approach based on Maude to model checking Reo models, encoding Reo's operational semantics of the connectors.

Proof assistants have been used to reason about Reo connectors [25, 26, 29, 35, 36, 14]. The approaches adopted by Li et al. [25, 35, 14] are among the ones that employ Coq to verify Reo models formally. In [25] a formalization of four of the Reo canonical connectors (Sync, FIFO1, SyncDrain, and LossySync) along with an LTL-based language defined as an inductive type in Coq is presented, while [35] proposes the formalization of five Reo canonical channels and a procedure that creates composite channels by logical conjunction of the connectors modelled.

In [14], a framework to provide means of graphically model Reo connectors and validate the generated model in Constraint Automata using Coq and NuSMV[2] is discussed. It also enables the automatic

[1] http://www.vereofy.de
[2] https://nusmv.fbk.eu/

generation of Coq code to a Haskell model employing the Coq code extraction apparatus. When restricting the works considering logics and Reo, as far as the authors know there is only the work by [12] which focuses on formalizing the semantics of Reo connectors Sync, LossySync, FIFO1, SyncDrain, AsyncDrain, Filter, Transform, Merger, and Replicator in terms of zero-safe Petri nets [11], a special class of Petri-nets with two types of places: zero and stable places. This encoding is then converted to terms in Intuitionistic Temporal Linear Logic, enabling reasoning about Reo connectors in this logic.

3 Background

This section provides a succinct overview of Reo [2, 3], considering its main characteristics and a modelling examples as it is the target language *ReLo* provides a formal semantic to reason over.

3.1 The Reo Modelling Language

As a coordination model, Reo focuses on connectors, their composition, and how they behave, not focusing on particular details regarding the entities that are connected, communicate, and interact through those connectors. Connected entities may be modules of sequential code, objects, agents, processes, web services, and any other software component where its integration with other software can be used to build a system [2]. Such entities are defined as component instances in Reo.

Channels in Reo are defined as a point-to-point link between two distinct nodes, where each channel has its unique predefined behavior. Each channel in Reo has exactly two ends, which can be of the following types: the source end, which accepts data into the channel, and the sink end, which dispenses data out of the channel. Channels are used to compose more complex connectors, being possible to combine user-defined channels amongst themselves and with the canonical connectors provided by Baier et al. [8]. Figure 1 shows the basic set of connectors as presented by Kokash et al. [22].

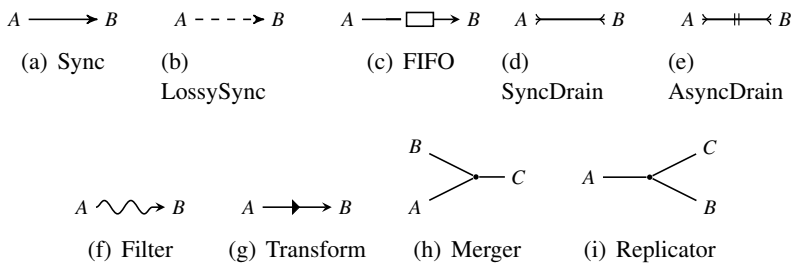


Figure 1: Canonical Reo connectors

Channel ends can be used by any entity to send/receive data, given that the entity belongs to an instance that knows these ends. Entities may use channels only if the instance they belong to is connected to one of the channel ends, enabling either sending or receiving data (depending on the kind of channel end the entity has access to).

The bound between a software instance and a channel end is a logical connection that does not rely on properties such as the location of the involved entities. Channels in Reo have the sole objective to enable the data exchange following the behaviour of the connectors composing the channel, utilizing I/O operations predefined for each entity in an instance. A channel can be known by zero or more instances at a time, but its ends can be used by at most one entity at the same time.

Figure 2 introduces a Reo connector known as Sequencer[3]. It models the data flow between three entities sequentially. The data flows from the first FIFO connector (a buffer), which will be sequentially synchronized with entities in port names names A, B, and C. The Sequencer can be used to model scenarios where processes sequentially interact between themselves.

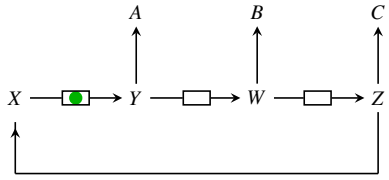


Figure 2: Modelling of the Sequencer in Reo

In short, Reo circuits may be understood as data flowing from different interfaces (i.e., port names connected to a node), where the connector itself models the communication pattern between two of these interfaces. A *ReLo* program is composed of one or more Reo connectors as introduced in Figure 1.

4 A *ReLo* Primer

ReLo [13] was tailored to subsume Reo models' behaviour naturally in a logic, without needing any mechanism to convert a Reo model denoted by one of its formal semantics to some logical framework. Each basic Reo connector is modelled in the logic's language, which is defined as follows.

Definition 1 (*ReLo*'s language)**.** The language of *ReLo* consists of the following:

- An enumerable set of propositions Φ.
- Reo channels as denoted by Figure 1
- A set of port names $\mathcal{N}$
- A sequence $Seq_\Pi = \{\varepsilon, s_1, s_2, \dots\}$ of data flows in ports of a *ReLo* program Π (defined below). We define $s_i \leq s_j$ if s_i is a proper (i.e., s_j contains all of s_i's data). Each sequence s_i denotes the data flow of the Reo program Π (i.e., all ports that have data synchronized at a specific moment in time) and ε is the empty sequence
- Program composition symbol : $\odot$
- A sequence t of data flows of ports p with data values $\{0,1\}$, which denotes whether p contains a data item. This describes a data flow occurring in the Reo channel. A BNF describing t is defined as follows:
  $\langle t \rangle$ ::= $\langle portName \rangle \langle data \rangle , \langle t \rangle \mid \langle data \rangle \langle portName \rangle \langle data \rangle , \langle t \rangle$
  $\mid \langle data \rangle \langle portName \rangle \langle data \rangle \mid \langle portName \rangle \langle data \rangle$
  $\langle portName \rangle$ ::= $p \in \mathcal{N}$
  $\langle data \rangle$ ::= $0 \mid 1$
- Iteration operator *

A *ReLo* program is defined as any Reo model built from the composition of Reo channels π_i. In *ReLo* their composition is $\Pi = (f, b)$, $\Pi = \pi_1 \odot \pi_2 \odot \cdots \odot \pi_n$, and $\pi_i = (f_1, b_1)$. $\odot$ follows the same notion of Reo composition, by "gluing" sink nodes of a connector to the source nodes of the other connector.

[3] http://arcatools.org/reo

The set f is the set of connectors p of the model where data flows in and out of the channel (the connector has at least a source node and a sink node), namely Sync, LossySync, FIFO, Filter, Transform, Merger and Replicator. The set b is the set of blocking channels (channels without sink nodes whose inability to fire prevents the remainder of connectors related to their port names from fire), namely SyncDrain and AsyncDrain.

The following is a simple yet intuitive example of the structure of data flows in *ReLo*. Let the sequence t be $t = \{A1, B1C\}$. It states that the port A has the data item 1 in its current data flow, while there is a data item 1 in the FIFO between B and C.

Definition 2 (*ReLo* formulae)**.**
We define formulae in *ReLo* as follows: $\phi = p \mid \top \mid \neg\phi \mid \phi \wedge \psi \mid \langle t, \pi \rangle \phi$, such that $p \in \Phi$. We use the standard abbreviations $\top \equiv \neg\bot, \phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi), \phi \rightarrow \psi \equiv \neg\phi \vee \psi$ and $[t, \pi]\varphi \equiv \neg\langle t, \pi \rangle \neg\phi$, where π is some Reo program and t a data flow.

The connectors in Figure 3 exemplify compound Reo connectors. The model SyncFIFO is composed of a FIFO and a Sync connector in which the data leaving the FIFO is sent to C from B synchronously. Suppose that there is data in the FIFO and in port B ($t = \{A1B, B0\}$). If the FIFO from A to B is processed first then the Sync between B and C, the data flow in B will be overwritten before it is sent to C, which is not the correct behaviour. The Sync from B to C must fire before the FIFO from A to B.

Another example is denoted by the model Sync2Drain. Suppose there is data only in port name A ($t = \{A1\}$). If the Sync from B to A is evaluated first then the SyncDrain between B and C, the restriction imposed by the fact that the condition required for the SyncDrain to fire was not met (as C's data flow differs from B's at this moment) is not considered, and data will wrongly flow from B to A. The SyncDrain must be first evaluated before all flows as they may block the flow from data of its ports to other channels.

(a) SyncFIFO (b) Sync2Drain

Figure 3: Examples of Reo models

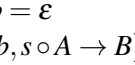
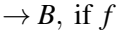
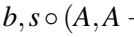
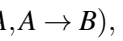
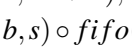
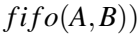
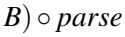
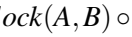
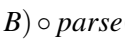
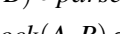
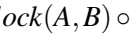
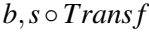
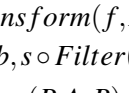
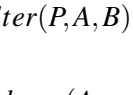
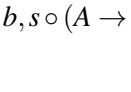
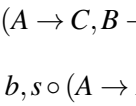
The next definition maps each canonical connector that composes a Reo model to a *ReLo* program. The left hand side of each mapping rule in Definition 3 is the atomic Reo connector, while the right hand size is the resulting *ReLo* atomic program $\pi_i = (f_i, b_i)$, with the same behaviour as of the Reo connector.

Definition 3 (*parse* base cases)**.** Each canonical Reo connector is mapped to a *ReLo* program in *parse*:

- to $A \rightarrow B$
- to $(A, A \rightarrow B)$
- to $fifo(A,B)$
- to $SBlock(A,B)$
- to $ABlock(A,B)$
- to $Transform(f,A,B)$, f: $Data \rightarrow Data$ is a transformation function.
- to $Filter(P,A,B)$, P is a logical predicate over the data item in A.
-  to $(A \rightarrow C, B \rightarrow C)$
-  to $(A \rightarrow B, A \rightarrow C)$

Considering that each *ReLo* program Π is the composition of programs $\pi_1 \odot \pi_2, \odot \cdots \odot \pi_n, \pi_i = (f_i, b_i)$ as Reo programs, *parse* is formalized in Definition 4. The symbol $\circ$ denote the addition of an element to s, the resulting set of *parse*'s processing.

Definition 4 (*parse* function)**.** The function that interprets the execution of a *ReLo* program is defined as $parse(f,b,s)$. We define ε as an abbreviation to denote when there is no *ReLo* program left to process (i.e. the base case when no program is parametrized). Its outcome is detailed as below.

- s, if $f = b = \varepsilon$
- $parse(f_j,b,s \circ A \to B)$, if $f =$  $\odot f_j$
  - $s \circ A \to B$, if $f =$
- $parse(f_j,b,s \circ (A, A \to B))$, if $f =$  $\odot f_j$
  - $s \circ (A, A \to B)$, if $f =$ 
- $parse(f_j,b,s) \circ fifo(A,B)$, if $f =$  $\odot f_j$
  - $(s \circ fifo(A,B))$, if $f =$ 
- $SBlock(A,B) \circ parse(f,b_j,s)$, if $b =$ $\odot b_j$
  - $(SBlock(A,B) \circ s)$, if $b =$ 
- $ABlock(A,B) \circ parse(f,b_j,s)$, if $b =$  $\odot b_j$
  - $(ABlock(A,B) \circ s)$, if $b =$ 
- $parse(f_j,b,s \circ Transform(f,A,B))$, if $f =$  $\odot f_j$
  - $(Transform(f,A,B) \circ s)$, if $f =$ 
- $parse(f_j,b,s \circ Filter(P,A,B))$, if $f =$  $\odot f_j$
  - $(Filter(P,A,B) \circ s)$, if $f =$ 
- $parse(f_j,b,s \circ (A \to C, B \to C))$, if $f =$  $\odot f_j$
  - $(s \circ (A \to C, B \to C))$, if $f =$ 
- $parse(f_j,b,s \circ (A \to B, A \to C))$, if $f =$  $\odot f_j$
  - $(s \circ (a \to b, a \to c))$, if $f =$ 

We employ *parse* to interpret Reo programs Π as a sequence of occurrences of possible data flow (where each flow corresponds to the execution of a Reo connector). These data flow may denote data transfer (*ReLo* programs $(A \to B)$ and $(A, A \to B)$, flow "blocks" induced by connectors such as SyncDrain and aSyncDrain (*ReLo* programs $SBlock(A,B)$ and $ABlock(A,B)$ — the first one requires that data flow synchronously through its ports, while the latter requires that data flow asynchronously through its ports). There is also the notion of a buffer introduced by FIFO connectors (*ReLo* program $fifo(A,B)$), which data flow into a buffer before flowing out of the channel, and merging/replicating data flow between ports, respectively denoted by channels Merger and Replicator (*ReLo* programs $(A \to C, B \to C)$ and $(A \to B, A \to C)$ respectively).

There are also special data flows, denoting the "transformation" of some data flowing from A to B as $Transform(f,A,B)$ which will apply f with the data in A before it sends $f(D_A)$ (D_A denoting the data

item in A) to B, and the filtering of data flow by some predicate as $Filter(P,A,B)$, P as a quantifiable-free predicate over the data item seen in A. Therefore, data will flow to B only if $P(D_A)$ is satisfied.

After processing π with *parse*, the interpretation of the execution of π is given by $go(t,s,acc), go: s \times s \to s$, where s is a string denoting the processed program π as the one returned by *parse*, and t is the initial data flow of ports of the Reo program π. The parameter *acc* holds all connectors of the Reo circuit that satisfy their respective required conditions for data to flow. In what follows we define $ax \prec t$ as an operator which states that ax is in t, ax a single data of a port and t a structure containing data flows for ports $p \in \mathcal{N}$.

Example 1 shows how *parse* functions and illustrates why it is necessary. The programs that depict the FIFO connectors from Fig. 2 are the last programs to be executed, while the ones that denote "immediate" flow (the Sync channels) come first. This is done to preserve the data when these connectors fire (if eligible). Suppose that there is a data item in the buffer between X and Y and a data item in Y (i.e., $t = X1Y,Y0$). If the data item leaves the buffer first then the data item in Y, the latter will be overwritten and the information is lost.

Example 1. let π be the Reo program corresponding to the circuit in Fig. 2:

$\pi =$ X —□→ Y ⊙ Y ⟶ A ⊙ Y —□→ W ⊙ W ⟶ B ⊙ W —□→ Z
⊙ Z ⟶ C ⊙ Z ⟶ X

$parse(\pi,\{\}) = \{Y \to A; W \to B; Z \to C; Z \to X; fifo(X,Y); fifo(Y,W); fifo(W,Z)\}$

The usage of *parse* is required to eliminate problems regarding the execution order of π's Reo channels, which could be caused by processing π the way it is inputted (i.e., its connectors can be in any order). Consider, for example, the behavior of SyncDrain and aSyncDrain programs as "blocking" programs as discussed earlier. In a single step, they must be evaluated before the flow programs, because if they fail to execute due to missing requirements, data should not flow from their port names to other connectors. In a nutshell, *parse* organizes the program so this verification can be performed.

Therefore, the interpretation of a π program processed by *parse* is performed by $go(t,s,acc)$, where s is a string containing π as processed by *parse*, t is π's initial data flow, and *acc* filters the connectors of the *ReLo* program that can be fired if the requirements to do so are met.

Definition 5 will check for each of the Reo connectors processed by *parse* satisfies the required condition to fire, following the connectors' behaviour. Operator $\prec$ denotes whether the data flow is within the current data flow t being evaluated. It is also used to denote whether the program currently being evaluated in s repeats in Π. Operator $\setminus$ denotes the removal of an connector from the accumulator *acc*.

Definition 5 (Relation *go* for a single execution step)**.** We define $go(t,s,acc)$ as follows:

- $s = \varepsilon : fire(t,acc)$
- $s = A \to B \circ s' :$
  - $go(t,s',acc \circ (A \to B))$, iff $Ax \prec t, (A \to B) \not\prec s'$
  - $go(t,s',(acc \circ (A \to B)) \setminus s'_j) \cup go(t,s',acc)$, iff $\begin{cases} Ax \prec t, \\ (A \to B) \not\prec s' \\ \exists s'_j \in acc \mid sink(s'_j) = B \end{cases}$
  - $go(t,s',acc)$, otherwise
- $s = (A, A \to B) \circ s' :$
  - $go(t,s',acc \circ (A \to B)) \cup go(t,s',acc \circ (A \to A))$, iff $Ax \prec t, (A \to B) \not\prec s'$

- $go(t,s',(acc \circ (A \to B)) \setminus s'_j) \cup go(t,s',acc)$, iff $\begin{cases} Ax \prec t, \\ (A \to B) \not\prec s' \\ \exists s'_j \in acc \mid sink(s'_j) = B \end{cases}$
- $go(t,s',acc)$, otherwise

• $s = fifo(A,B) \circ s'$:
- $go(t,s',acc \circ (AxB))$, iff $Ax \prec t, fifo(A,B) \not\prec s', (AxB) \not\prec acc$
- $go(t,s',acc \circ (AxB \to Bx))$, iff $AxB \prec t, fifo(A,B) \not\prec s'$
- $go(t,s',(acc \circ (AxB \to Bx)) \setminus s'_j) \cup go(t,s',acc)$, iff $\begin{cases} AxB \prec t, \\ fifo(A,B) \not\prec s', \\ \exists s'_j \in acc \mid sink(s'_j) = B \end{cases}$
- $go(t,s',acc)$, otherwise

• $s = Sblock(A,B) \circ s'$:
- $go(t,s',acc)$, iff $\begin{cases} (Ax \prec t \wedge Bx \prec t) \vee (Ax \not\prec t \wedge Bx \not\prec t) \\ Sblock(A,B) \not\prec s' \end{cases}$
- $go(t,halt(A,B,s'),acc)$, iff $\begin{cases} (Ax \prec t \wedge Bx \not\prec t) \vee (Ax \not\prec t \wedge Bx \prec t) \\ Sblock(A,B) \not\prec s' \end{cases}$

• $s = Ablock(A,b) \circ s'$:
- $go(t,s',acc)$, iff $\begin{cases} (Ax \not\prec t \wedge Bx \prec t) \vee (Ax \prec t \wedge Bx \not\prec t) \vee \\ (Ax \not\prec t \wedge Bx \not\prec t), Ablock(A,B) \not\prec s' \end{cases}$
- $go(t,halt(A,B,s'),acc)$, iff $\begin{cases} (Ax \prec t \wedge Bx \prec t), \\ Ablock(A,B) \not\prec s' \end{cases}$

• $s = Transform(f,A,B) \circ s'$:
- $go(t,s',acc \circ (f(D_A) \to B))$, iff $\begin{cases} ax \prec t \\ Transform(f,A,B) \not\prec s' \end{cases}$
- $go(t,s',(acc \circ (f(D_A) \to B)) \setminus s'_j) \cup go(t,s',acc)$, iff $\begin{cases} Ax \prec t, \\ Transform(f,A,B) \not\prec s' \\ \exists s'_j \in acc \mid sink(s'_j) = B \end{cases}$
- $go(t,s',acc)$, otherwise

• $s = Filter(f,A,B) \circ s'$:
- $go(t,s',acc \circ (A \to B))$, iff $\begin{cases} Ax \prec t \\ P(D_A) \text{ holds} \\ Filter(f,A,B) \not\prec s' \end{cases}$
- $go(t,s',(acc \circ (A \to B)) \setminus s'_j) \cup go(t,s',acc)$, iff $\begin{cases} Ax \prec t, \\ P(D_A) \text{ holds} \\ Filter(f,A,B) \not\prec s' \\ \exists s'_j \in acc \mid sink(s'_j) = B \end{cases}$
- $go(t,s',acc)$, otherwise

The existing condition after each return condition of *go* denotes the case where two or more Reo connectors within a circuit have the same sink node. This implies that if both of their respective source

nodes have data flowing simultaneously, their sink nodes will have data flowing nondeterministically. Such condition models this scenario, considering when both cases may happen as two nondeterministic "distinct" possible executions. Therefore, the operation $acc \circ (X \to Y)) \setminus s'_j$ removes every interpretation of s' which sink node equals Y, while $go(t, s', acc)$ denotes an execution containing the removed s'_j but not considering $X \to Y$. The return condition $s = \varepsilon$ denotes that the program as a whole has already been processed.

Considering the cases including block programs induced by SyncDrain and AsyncDrain connectors, $halt(A, B, s')$ is defined as a supporting function that will be used in the case the block program conditions fail. Then, data flow that was in the ports of the SyncDrain/AsyncDrain evaluated cannot be further considered in this execution steps: channels that have their sink node pointed to A or B.

Intuitively, *go* is a function that processes a program π with input t as the program's data initially available at ports $p \in \pi$ and returns the next data configuration after processing all connectors and verifying whether they are eligible for data to flow. The return of *go* depends on a function $fire$ which is bound to return the final configuration of the Reo circuit after an iteration (i.e., the last ports that data flow). We define $sink(s'_j)$ as the sink node of a connector, in this case, the port name where a data item flowing into a Reo connector is bound to. The operation denoted by $\cup$ is the standard set union.

Definition *go* employs a function named $fire\colon T \times s \to T$ which returns the firing of all possible data flows in the Reo connector, given the Reo program π and an initial data flow on ports of π. The set T is the set of possible data flows as constructed by the BNF grammar in Definition 1. The function $fire$ returns the resulting data flow of this execution step by considering the program processed by *go* as s and the current step's data flow t. Parameter s contains *ReLo* programs as yielded by *parse*.

Definition 6 (Data marking relation $fire$)**.**

$$fire(t,s) = \begin{cases} \varepsilon, \text{ if } s = \varepsilon \\ AxB \circ fire(t,s'), \text{ if } s = (AxB) \circ s' \text{ and } Ax \prec t \\ B(f(a)) \circ fire(t,s'), \text{ if } s = (f(D_A) \to B) \circ s' \text{ and } Ax \prec t \\ Bx \circ fire(t,s'), \text{ if } \begin{cases} s = (A \to B) \circ s' \text{ and } Ax \prec t, or \\ s = (AxB \to Bx) \circ s' \text{ and } axb \prec t \end{cases} \end{cases} \tag{1}$$

We define f_{ReLo} as the transition relation of a *ReLo* model. It denotes how the transitions of the model fire, i.e., given an input t and a program π denoting a Reo circuit, $f_{ReLo}(t, \pi)$ interfaces with *go* to return the resulting data flow of π given that data depicted by t are flowing in the connector's ports.

Definition 7. Transition relation $f_{ReLo}(t, \pi) = go(t, (parse(\pi, [])), [])$

We define $f_{ReLo}(t, \pi^\star)$ as the application of $f_{ReLo}(t, \pi)$ iteratively for the (nondeterministic finite) number of steps denoted by $\star$, starting with t with π, and considering the obtained intermediate t' in the steps.

A *ReLo* frame is a structure based on Kripke frames [24] formally defined as a tuple $\mathscr{F} = \langle S, \Pi, R_\Pi, \delta, \lambda \rangle$, where each element of $\mathscr{F}$ is described by Definition 8.

Definition 8 (*ReLo* frame)**.** S is a non-empty enumerable set of states and Π a Reo program.

- $R_\Pi \subseteq S \times S$ is a relation defined as follows.
  - $R_{\pi_i} = \{uR_{\pi_i}v \mid f_{ReLo}(t, \pi_i) \prec \delta(v),\ t \prec \delta(u)\}$, π_i is any combination of any atomic program which is a subprogram of Π.
  - $R_{\pi_i^\star} = R^\star_{\pi_i}$, the reflexive transitive closure (RTC) of R_{π_i}.

- $\lambda : S \times \mathcal{N} \to \mathbb{R}$ is a function that returns the time instant a data item in a data markup flows through a port name of $\mathcal{N}$.
- $\delta : S \to T$, is a function that returns data in ports of the circuit in a state $s \in S$, T being the set of possible data flows in the model.

From Definition 8, a *ReLo* model is formally defined as a tuple $\mathscr{M} = \langle \mathscr{F}, \mathbf{V} \rangle$ by Definition 9. Intuitively, it is a tuple consisting of a *ReLo* frame and a valuation function, which given a state w of the model and a propositional symbol $\varphi \in \Phi$, maps to either $true$ or $false$.

Definition 9 (*ReLo* models)**.** A model in *ReLo* is a tuple $\mathscr{M} = \langle \mathscr{F}, \mathbf{V} \rangle$, where $\mathscr{F}$ is a *ReLo* frame and $V : S \times \Phi \to \{true, false\}$ is the model's valuation function

Definition 10 (Satisfaction notion)**.**

- $\mathscr{M}, s \Vdash p$ iff $V(s, p) = true$
- $\mathscr{M}, s \Vdash \top$ always
- $\mathscr{M}, s \Vdash \neg\varphi$ iff $\mathscr{M}, s \nVdash \varphi$
- $\mathscr{M}, s \Vdash \varphi_1 \wedge \varphi_2$ iff $\mathscr{M}, s \Vdash \varphi_1$ and $\mathscr{M}, s \Vdash \varphi_2$
- $\mathscr{M}, s \Vdash \langle t, \pi \rangle \varphi$ if there exists a state $w \in S$, $sR_\pi w$, and $\mathscr{M}, s \Vdash \varphi$

We denote by $\mathscr{M} \Vdash \varphi$ if φ is satisfied in all states of $\mathscr{M}$. By $\Vdash \varphi$ we denote that φ is valid in any state of any model.

We recover the circuit in Fig. 2 as an example. Let us consider s = D_X, (i.e. t = D1) and the Sequencer's corresponding model $\mathscr{M}$. Therefore, $\mathscr{M}, D_X \Vdash \langle t, \pi \rangle p$ holds if $V(D_{X fifo Y}, p) = true$ as $D_{X fifo Y}$ is the only state where $D_X R_\Pi D_{X fifo Y}$. For example, one might state p as "There is no port with any data flow", hence $V(D_{X fifo Y}, p) = true$.

As another usage example, we formalize some properties which may be interesting for this connector to have. Let us consider that the data markup is $t = X1$, $\mathscr{M}$ the model regarding the Sequencer, and the states' subscript denoting which part of the connector has data. The following example state that for this data flow, after every single execution of π, it is not the case that the three connected entities have their data equal to 1 simultaneously, but it does have data in its buffer from X to Y.

Example 2. $[X1, \pi]\neg(D_A = 1 \wedge D_B = 1 \wedge D_C = 1) \wedge t' = X1Y$, where $t' = f_{ReLo}(t, \pi)$
$\mathscr{M}, D_X \Vdash [X1, \pi]\neg(D_A = 1 \wedge D_B = 1 \wedge D_C = 1) \wedge t' = X1Y$.
$\mathscr{M}, D_{x \multimap \square \to y} \Vdash \neg(D_A = 1 \wedge D_B = 1 \wedge D_C = 1) \wedge t' = X1Y$.
$\mathscr{M}, D_{x \multimap \square \to y} \Vdash \neg(D_A = 1 \wedge D_B = 1 \wedge D_C = 1)$ and $\mathscr{M}, D_{x \multimap \square \to y} \Vdash t' = X1Y$.

The notion of $\mathscr{M}, D_X \Vdash \langle t, \pi^\star \rangle p$ holds if a state s is reached from D_X by means of $R^\star_\pi$ with $V(s, p) = \top$. If we state p as "the data item of port X equals 1", it holds because $D_X R^\star_\pi D_X$ and $V(D_X, p) = \top$. If there is an execution of π that lasts a nondeterministic finite number of iterations, and there is data in C equals to 1, then there is an execution under the same circumstances where the same data has been in B.

Example 3. $\langle t, \pi^\star \rangle D_C = 1 \to \langle t, \pi^\star \rangle D_B = 1$
$\mathscr{M}, D_X \Vdash \langle t, \pi^\star \rangle D_C = 1 \to \langle t, \pi^\star \rangle D_B = 1$
$\mathscr{M}, D_X \Vdash \neg(\langle t, \pi^\star \rangle D_C = 1) \vee \langle t, \pi^\star \rangle D_B = 1$
$\mathscr{M}, D_X \Vdash [t, \pi^\star]\neg D_C = 1 \vee \langle t, \pi^\star \rangle D_B = 1$
$\mathscr{M}, D_X \Vdash [t, \pi^\star]\neg D_C = 1$ or $\mathscr{M}, D_X \Vdash \langle t, \pi^\star \rangle D_B = 1$
$\mathscr{M}, D_X \Vdash \langle t, \pi^\star \rangle D_B = 1$, because $\mathscr{M}, D_B \Vdash D_B = 1$ and $D_X R_{\pi^\star} R_B$.

4.1 Axiomatic System

We define an axiomatization of *ReLo*, discuss its soundness and completeness.

Definition 11 (Axiomatic System)**.**

(PL) Enough Propositional Logic tautologies
(K) $[t,\pi](\varphi \to \psi) \to ([t,\pi]\varphi \to [t,\pi]\psi)$
(And) $[t,\pi](\varphi \wedge \psi) \leftrightarrow [t,\pi]\varphi \wedge [t,\pi]\varphi$
(Du) $[t,\pi]\varphi \leftrightarrow \neg\langle t,\pi\rangle\neg\varphi$
(R) $\langle t,\pi\rangle\varphi \leftrightarrow \varphi$ iff $f_{ReLo}(t,\pi) = \varepsilon$
(It) $\varphi \wedge [t,\pi][t_{(f,b)},\pi^\star]\varphi \leftrightarrow [t,\pi^\star]\varphi$, $t_{(f,b)} = f_{ReLo}(t,\pi)$
(Ind) $\varphi \wedge [t,\pi^\star](\varphi \to [t_{(f,b)^\star},\pi]\varphi) \to [t,\pi^\star]\varphi$, $t_{(f,b)^\star} = f_{ReLo}(t,\pi^\star)$

$$\textbf{(MP)}\ \frac{\varphi \qquad \varphi \to \psi}{\psi} \qquad \textbf{(Gen)}\ \frac{\varphi}{[t,\pi]\varphi}$$

Lemma 1 (Soundness)**.** *Proof.*
Axioms **(PL)**, **(K)**, **(And)** and **(Du)** are standard in Modal Logic literature, along with rules **(MP)** and **(Gen)** [15]. Axiom **(It)** and **(Ind)** are similar from PDL. **(R):** $\langle t,\pi\rangle\varphi \leftrightarrow \varphi$ iff $f_{ReLo}(t,\pi) = \varepsilon$
Suppose by contradiction that exists a state s from a model $\mathscr{M} = \langle S,\Pi,R_\Pi,\delta,\lambda,V\rangle$ where **(R)** does not hold. There are two possible cases.
($\Rightarrow$) Suppose by contradiction $\mathscr{M},s \Vdash \langle t,(f,b)\rangle\varphi$ and $\mathscr{M},s \nVdash \varphi$. $\mathscr{M},s \Vdash \langle t,(f,b)\rangle\varphi$ iff there is a state $v \in S$ such that $sR_\pi v$. Because $f_{ReLo}(t,(f,b)) = \varepsilon, s = v$ (i.e., in this execution no other state is reached from s). Therefore, $\mathscr{M},s \Vdash \varphi$, contradicting $\mathscr{M},s \nVdash \varphi$.
($\Leftarrow$) Suppose by contradiction $\mathscr{M},s \Vdash \varphi$ and $\mathscr{M},s \nVdash \langle t,(f,b)\rangle\varphi$. In order to $\mathscr{M},s \nVdash \langle t,(f,b)\rangle\varphi$, for every state $v \in S$ such that $sR_\pi v$, $\mathscr{M},v \nVdash \varphi$. Because $f_{ReLo}(t,(f,b)) = \varepsilon, s = v$ (i.e., in this execution no other state is reached from s). Therefore, $\mathscr{M},v \nVdash \varphi$, contradicting $\mathscr{M},v \Vdash \varphi$.

□

4.2 Completeness

We start by defining the Fisher-Ladner closure of a formula as the set closed by all of its subformulae, following the idea employed in other modal logic works [15, 9] as follows.

Definition 12 (Fisher-Ladner Closure)**.** Let Φ be a the set of all formulae in *ReLo*. The Fischer-Ladner closure of a formula, notation $FL(\varphi)$ is inductively defined as follows:

- $FL\colon \Phi \to 2^\Phi$
- $FL_{(f,b)}\colon \{\langle t,(f,b)\rangle\varphi\} \to 2^\Phi$, where (f,b) is a *ReLo* program and φ a *ReLo* formula.

These functions are defined as

- $FL(p) = \{p\}$, p an atomic proposition;
- $FL(\varphi \to \psi) = \{\varphi \to \psi\} \cup FL(\varphi) \cup FL(\psi)$
- $FL_{(f,b)}(\langle t,(f,b)\rangle\varphi) = \{\langle t,(f,b)\rangle\varphi\}$
- $FL(\langle t,(f,b)\rangle\varphi) = FL_{(f,b)}((\langle t,(f,b)\rangle\varphi) \cup FL(\varphi)$
- $FL_{(f,b)}(\langle t,(f,b)^\star\rangle\varphi) = \{\langle t,(f,b)^\star\rangle\varphi\} \cup FL_{(f,b)}(\langle t,(f,b)\rangle\langle t,(f,b)^\star\rangle\varphi)$
- $FL(\langle t,(f,b)^\star\rangle\varphi) = FL_{(f,b)}((\langle t,(f,b)^\star\rangle\varphi) \cup FL(\varphi)$

From the definitions above, we prove two lemmas that can be understood as properties that formulae need to satisfy to belong to their Fisher-Ladner closure.

Lemma 2. *If* $\langle t,(f,b)\rangle\psi \in FL(\varphi)$*, then* $\psi \in FL(\varphi)$

Lemma 3. *If* $\langle t,(f,b)^\star\rangle\psi \in FL(\varphi)$*, then* $\langle t,(f,b)\rangle\langle t,(f,b)^\star\rangle\psi \in FL(\varphi)$

The proofs for Lemmas 2 and 3 are straightforward from Definition 12. The following definitions regard the definitions of maximal canonical subsets of *ReLo* formulae. We first extend Definition 12 to a set of formulae Γ. The Fisher-Ladner closure of a set of formulae Γ is $FL(\Gamma) = \bigcup_{\varphi\in\Gamma} FL(\varphi)$. Therefore, $FL(\Gamma)$ is closed under subformulae. For the remainder of this section, we will assume that Γ is finite.

Lemma 4. *If* Γ *is a finite set of formulae, then* $FL(\Gamma)$ *also is a finite set of formulae*

Proof. The proof is standard in literature [10]. Intuitively, because FL is defined recursively over a set of formulae Γ into formulae ψ of a formula $\varphi \in \Gamma$, Γ being finite leads to the resulting set of $FL(\Gamma)$ also being finite (at some point, all atomic formulae composing φ will have been reached by FL). □

Definition 13 (Atom)**.** Let Γ be a set of consistent formulae. An atom of Γ is a set of formulae Γ' that is a maximal consistent subset of $FL(\Gamma)$. The set of all atoms of Γ is defined as $At(\Gamma)$.

Lemma 5. *Let* Γ *a consistent set of formulae and* ψ *a ReLo formula. If* $\psi \in FL(\Gamma)$*, and* ψ *is satisfiable then there is an atom of* Γ*,* Γ' *where* $\psi \in \Gamma'$*.*

Proof. The proof follows from Lindembaum's lemma. From Lemma 4, as $FL(\Gamma)$ is a finite set, its elements can be enumerated from $\gamma_1,\gamma_2,\ldots,\gamma_n, n = |FL(\Gamma)|$. The first set, Γ'_1 contains ψ as the starting point of the construction. Then, for $i = 2,\ldots,n$, Γ'_i is the union of Γ'_{i-1} with either $\{\gamma_i\}$ or $\{\neg\gamma_i\}$, respectively whether $\Gamma'_i \cup \{\gamma_i\}$ or $\Gamma'_i \cup \{\neg\gamma_i\}$ is consistent. In the end, we make $\Gamma' = \Gamma'_n$ as it contains the union of all $\Gamma_i, 1 \le i \le n$. This is summarized in the following bullets:

- $\Gamma'_1 = \{\psi\}$;
- $\Gamma'_i, = \begin{cases} \Gamma'_{i-1} \cup \{\gamma_i\}, \text{ if } \Gamma_{n-1} \cup \{\gamma_n\} \text{ is consistent} \\ \Gamma'_{i-1} \cup \{\neg\gamma_i\}, \text{ otherwise} \end{cases}$ for $1 < i < n$;
- $\Gamma = \bigcup_{i=1}^{n} \Gamma_i$

□

Definition 14 (Canonical relations over Γ)**.** Let Γ a set of formulae, A,B atoms of Γ ($A,B \in At(\Gamma)$), Π a *ReLo* program and $\langle t,(f,b)\rangle\varphi \in At(\Gamma)$. The canonical relations on $At(\Gamma)$ is defined as S^Γ_Π as follows:
$AS^\Gamma_\Pi B \leftrightarrow \bigwedge A \wedge \langle t,(f,b)\rangle \bigwedge B)$ is consistent , $AS^\Gamma_{\Pi^\star} B \leftrightarrow \bigwedge A \wedge \langle t,(f,b)^\star\rangle \bigwedge B)$ is consistent

Definition 14 states that the relation between two atoms of Γ, A and B is done by the conjunction of the formulae in A with all formulae in B which can be accessed from A with a diamond formula, such that this conjunction is also a consistent formula. Intuitively, it states that A and B are related in S^Γ_Π by every formula φ of B which conjunction with A by means of a diamond results in a consistent scenario.

The following definition is bound to formalize the canonical version of δ as the data markup function.

Definition 15 (Canonical data markup function δ^Γ_c)**.**
Let $F = \{\langle t_1,(f_1,b_1)\rangle\varphi_1, \langle t_2,(f_2,b_2)\rangle\varphi_2,\ldots,\langle t_n,(f_n,b_n)\rangle\varphi_n\}$ be the set of all diamond formula occurring on an atom A of Γ. The canonical data markup is defined as $\delta^\Gamma_c : At(\Gamma) \to T$ as follows:

- The sequence $\{t_1,t_2,\ldots,t_n\} \subseteq \delta(A)$ Therefore, $\{t_1,t_2,\ldots,t_n\} \subseteq \delta^\Gamma_c(A)$. Intuitively, this states that all the data flow in the set of formulae must be valid data markups of A, which leads to them to also be valid data markups of δ^Γ_c following Definition 14.
- for all programs $\pi = (f,b) \in \Pi$, $f_{ReLo}((\delta^\Gamma_c(A)),(f,b)) \prec \delta^\Gamma_c(B) \leftrightarrow AS^\Gamma_\Pi B$.

Definition 16 (Canonical model)**.** A canonical model over a set of formulae Γ is defined as a *ReLo* model $\mathscr{M}_c^\Gamma = \langle At(\Gamma), \Pi, S_\Pi^\Gamma, \delta_c^\Gamma, \lambda_c, V_c^\Gamma \rangle$, where:

- $At(\Gamma)$ is the set of states of the canonical model;
- Π is the model's *ReLo* program;
- S_Π^Γ are the canonical relations over Γ;
- δ_c^Γ is the canonical markup function;
- $\lambda_c \colon At(\Gamma) \times \mathscr{N} \to \mathbb{R}$;
- $V_c^\Gamma \colon At(\Gamma) \times \varphi \to \{true, false\}$, namely $V_c^\Gamma(A,p) = \{A \in At(\Gamma) \mid p \in A\}$;

Lemma 6. *For all programs $\pi = (f,b)$ that compose Π, $t = \delta_c^\Gamma(A)$:*

1. *If $f_{ReLo}(t,(f,b)) \neq \varepsilon$, then $f_{ReLo}(t,(f,b)) \prec \delta_c^\Gamma(B)$ iff $AS_\Pi^\Gamma B$.*
2. *If $f_{ReLo}(t,(f,b)) = \varepsilon$, then $(A,B) \notin S_\Pi^\Gamma$.*

Proof. The proof for 1. is straightforward from Definition 15. The proof for 2. follows from axiom *R*. Because $f_{ReLo}(t,(f,b)) = \varepsilon$, no other state is reached from the current state, hence no state *B* related with *A* by R_Π^Γ can be reached. □

The following lemma states that canonical models always exists if there is a formula $\langle t,(f,b)\varphi\rangle \in FL(\Gamma)$, a set of formulae Γ and a Maximal Consistent Set $A \in At(\Gamma)$. This assures that given the required conditions, a canonical model can always be built.

Lemma 7 (Existence Lemma for canonical models)**.** *Let A be an atom of $At(\Gamma)$ and $\langle t,(f,b)\rangle\varphi \in FL(\Gamma)$. $\langle t,(f,b)\rangle\varphi \in A \iff \exists$ an atom $B \in At(\Gamma)$ such that $AS_\Pi^\Gamma B$, $t \prec \delta_c^\Gamma(A)$ and $\varphi \in B$.*

Proof. $\Rightarrow$ Let $A \in At(\Gamma)$ $\langle t,(f,b)\rangle\varphi \in FL(\Gamma)$ and $\langle t,(f,b)\rangle\varphi \in A$. Because $A \in At(\Gamma)$, from Definition 15 we have $t \prec \delta_c^\Gamma(A)$. From Lemma 5 we have that if $\psi \in FL(\Gamma)$ and ψ is consistent, then there is an atom of Γ, Γ' where $\psi \in \Gamma'$. Rewriting φ as $(\varphi \wedge \gamma) \vee (\varphi \wedge \neg\gamma)$ (a tautology from Propositional Logic), an atom $B \in At(\Gamma)$ can be constructed, because either $\langle t,(f,b)\rangle(\varphi \wedge \gamma)$ or $\langle t,(f,b)\rangle(\varphi \wedge \neg\gamma)$ is consistent. Therefore, considering all formulae $\gamma \in FL(\Gamma)$, $B \in At(\Gamma)$ is constructed with $\varphi \in B$ and $A \wedge (\langle t,(f,b)\rangle\varphi \bigwedge B$. From Definition 14, $AS_\Pi^\Gamma B$.
$\Leftarrow$ Let $A \in At(\Gamma)$ and $\langle t,(f,b)\rangle\varphi \in FL(\Gamma)$. Also, let $B \in At(\Gamma)$, $AS_\Pi^\Gamma B$, $t \prec \delta_c^\Gamma(A)$, and $\varphi \in B$. As $AS_\Pi^\Gamma B$, from Definition 14, $AS_\Pi^\Gamma B \leftrightarrow (A \wedge \langle t,(f,b)\rangle \bigwedge B)$, $\forall \varphi_i \in B$ is consistent. From $\varphi \in B$, $(A \wedge \langle t,(f,b)\rangle\varphi)$ is also consistent. As $A \in At(\Gamma)$ and $\langle t,(f,b)\varphi \in FL(\Gamma)$, by Definition 13, as *A* is maximal, then $\langle t,(f,b)\rangle\varphi \in A$.

□

The following lemma formalizes the truth notion for a canonical model $\mathscr{M}_c^\Gamma$, given a state *s* and a formula φ. It formalizes the semantic notion for canonical models in *ReLo*.

Lemma 8 (Truth Lemma)**.** *Let $\mathscr{M}_c^\Gamma = \langle At(\Gamma), \Pi, S_\Pi^\Gamma, \delta_c^\Gamma, \lambda, V_c^\Gamma \rangle$ be a canonical model over a formula γ. Then, for every state $A \in At(\Gamma)$ and every formula $\varphi \in FL(\gamma)$: $\mathscr{M}_c^\Gamma, A \Vdash \varphi \iff \varphi \in A$.*

Proof. The proof proceeds by induction over the structure of φ.

- Induction basis: suppose φ is a proposition *p*. Therefore, $\mathscr{M}_c^\Gamma, A \Vdash p$. From Definition 16, $\mathscr{M}_c^\Gamma$'s valuation function is $V_c^\Gamma(p) = \{A \in At(\Gamma) \mid p \in A\}$. Therefore, $p \in A$.
- Induction Hypothesis: Suppose φ is a non atomic formula ψ. Then, $\mathscr{M}_c^\Gamma, A \Vdash \psi \iff \psi \in A$, ψ a strict subformula of φ.
- Inductive step: Let us prove it holds for the following cases (we ommit propositional operators):

- Case $\varphi = \langle t,(f,b)\rangle\phi$. Then, $\mathscr{M}_c^\Gamma, A \Vdash \langle t,(f,b)\rangle\phi \iff \langle t,(f,b)\rangle\phi \in A$:
  $\Rightarrow$ Let $\mathscr{M}_c^\Gamma, A \Vdash \langle t,(f,b)\rangle\phi$. From Definition 14, there is a state B where $AS_\Pi^\Gamma B$ and $\phi \in B$. By Lemma 7, $\langle t,(f,b)\rangle\phi \in A$. Therefore, it holds.
  $\Leftarrow$ Let $\mathscr{M}_c^\Gamma, A \nVdash \langle t,(f,b)\rangle\phi$. From Definition 16's valuation function V_c^Γ and Lemma 5, we have $\mathscr{M}_c^\Gamma, A \Vdash \neg\langle t,(f,b)\rangle\phi$. Therefore, for every B where $AS_\Pi^\Gamma B, \mathscr{M}_c^\Gamma, B \Vdash \neg\phi$. From the induction hypothesis, $\phi \notin B$. Hence, From Lemma 7, $\langle t,(f,b)\rangle\phi \notin A$.
- Case $\varphi = \langle t,(f,b)^\star\rangle\phi$. Then, $\mathscr{M}_c^\Gamma, A \Vdash \langle t,(f,b)^\star\rangle\phi \iff \langle t,(f,b)^\star\rangle\phi \in A$:
  $\Rightarrow$ Let $\mathscr{M}_c^\Gamma, A \Vdash \langle t,(f,b)^\star\rangle\phi$. From Definition 14, there is a state B where $AS_{\Pi^\star}^\Gamma B$ and $\phi \in B$. By Lemma 7, $\langle t,(f,b)^\star\rangle\phi \in A$. Therefore, it holds.
  $\Leftarrow$ Let $\mathscr{M}_c^\Gamma, A \nVdash \langle t,(f,b)^\star\rangle\phi$. From Definition 16's valuation function V_c^Γ and Lemma 5, we have $\mathscr{M}_c^\Gamma, A \Vdash \neg\langle t,(f,b)^\star\rangle\phi$. Therefore, for every B where $AS_{\Pi^\star}^\Gamma B, \mathscr{M}_c^\Gamma, B \Vdash \neg\phi$. From the induction hypothesis, $\phi \notin B$. Hence, From Lemma 7, $\langle t,(f,b)^\star\rangle\phi \notin A$.

□

We proceed by formalizing the following lemma, which is bound to show that the properties that define $\star$ for regular *ReLo* models also holds in *ReLo* canonical models.

Lemma 9. *Let $A,B \in At(\Gamma)$ and Π a ReLo program. If $AS_{\Pi^\star}B$ then $AS_\Pi^\star B$*

Proof. Suppose $AS_{\Pi^\star}B$. Define $C = \{C' \in At(\Gamma) \mid AS_\Pi^\star C\}$ as the set of all atoms C' which A reaches by means of $S_{\Pi^\star}$. We will show that $B \in C$. Let C_c be the maximal consistent set obtained by means of Lemma 5, $C_c = \{\bigwedge C_1 \vee C_2 \vee \ldots \bigwedge C_n\}$, where the conjunction of each C_i is consistent, and each C_i is a maximal consistent set. Also, define $t = \delta_c^\Gamma(C_c)$ as the canonical markup of C_c.

Note that $C_c \wedge \langle t,(f,b)\rangle\neg C_c$ is inconsistent: if it was consistent, then for some $D \in At(\Gamma)$ which A cannot reach, $C_c \wedge \langle t,(f,b)\rangle \bigwedge D$ would be consistent, which leads to $\bigwedge C_1 \vee C_2 \vee \cdots \vee C_i \vee \langle t,(f,b)\rangle \bigwedge D$ also being consistent, for some C_i. By the definition of C_c, this means that $D \in C$ but that is not the case (because $D \in C_c$ contradicts D not being reached from A and consequently C_c's definition, as $D \in C_c$ leads to D being reachable from A). Following a similar reasoning, $\bigwedge A \wedge \langle t,(f,b)\rangle C_c$ is also inconsistent and therefore its negation, $\bigwedge \neg(A \wedge \langle t,(f,b)\rangle C_c)$ is consistent, which can be rewritten as $\bigwedge A \rightarrow [t,(f,b)]C_c$.

Because $C_c \wedge \langle t,(f,b)\rangle\neg C_c$ is inconsistent, its negation $\neg(C_c \wedge \langle t,(f,b)\rangle\neg C_c)$ is valid, which can be rewritten to $\vdash C_c \rightarrow [t,(f,b)]C_c$ (I). Therefore, by applying generalization we have $\vdash [t,(f,b)^\star](C_c \rightarrow [t,(f,b)]C_c)$. By axiom **(It)**, we derive $\vdash [t,(f,b)]C_c \rightarrow [t,(f,b)^\star]C_c$ (II). By rewriting (II) in (I) we derive $C_c \rightarrow [t,(f,b)^\star]C_c$. As $\bigwedge A \rightarrow [t,(f,b)]C_c$ is valid, from (II) $\bigwedge A \rightarrow [t,(f,b)^\star]C_c$ also is valid. From the hypothesis $AS_{\pi^\star}B$ and C_c's definition, $\bigwedge A \wedge \langle t,(f,b)^\star\rangle B$ and $\bigwedge B \wedge C_c$ are consistent (the latter from C_c's definition). Then, there is a $C_i \in C_c$ such that $\bigwedge B \wedge \bigwedge C$ is consistent. But because each C_i is a maximal consistent set, it is the case that $B = C_i$, which by the definition of C_c leads to $AS_\Pi^\star B$.

□

Definition 17 (Proper Canonical Model)**.** The proper canonical model over a set of formulae Γ is defined as a tuple $\langle At(\Gamma), \Pi, R_\Pi^\Gamma, \delta_\Pi^\Gamma, \lambda_c, V_\Pi^\Gamma\rangle$ as follows:

- $At(\Gamma)$ as the set of atoms of Γ;
- Π as the *ReLo* program;
- The relation R of a *ReLo* program Π is inductively defined as:
  - $R_\pi = S_\pi$ for each canonical program π;
  - $R_{\Pi^\star}^\Gamma = (R_\Pi^\Gamma)^\star$;
  - $\Pi = \pi_1 \odot \pi_2 \odot \cdots \odot \pi_n$ a *ReLo* program, $R_\Pi \subseteq S \times S$ as follows:

  - $R_{\pi_i} = \{uR_{\pi_i}v \mid f_{ReLo}(t,\pi_i) \prec \delta(v)\}$, $t \prec \delta(u)$ and π_i is any combination of any atomic programs which is a subprogram of Π.
- δ^{Γ}_{Π} as the canonical markup function;
- λ_c: $At(\Gamma) \times \mathscr{N} \to \mathbb{R}$;
- $V^{\Gamma}_c(A,p) = \{A \in At(\Gamma) \mid p \in A\}$ as the canonical valuation introduced by Definition 16.

Lemma 10. *Every canonical model for* Π *has a corresponding proper canonical model: for all programs* Π, $S^{\Gamma}_{\Pi} \subseteq R^{\Gamma}_{\Pi}$

Proof. The proof proceeds by induction on Π's length

- For basic programs π, it follows from Definition 17:
- $\Pi^{\star}$: From Definition 8, $R_{\pi^{\star}} = R^{\star}_{\pi}$. By the induction hypothesis, $S^{\Gamma}_{\Pi} \subseteq R^{\Gamma}_{\Pi}$, also from the definition of RTC, we have that if $(S^{\Gamma}_{\Pi}) \subseteq (R^{\Gamma}_{\Pi})$, then $(S^{\Gamma}_{\Pi})^{\star} \subseteq (R^{\Gamma}_{\pi})^{\star}$ (i). From Lemma 9, $S^{\Gamma}_{\Pi^{\star}} \subseteq (S^{\Gamma}_{\Pi})^{\star}$, which leads to $(S^{\Gamma}_{\Pi})^{\star} \subseteq (R^{\Gamma}_{\Pi})^{\star}$ by (i). Finally, $(R^{\Gamma}_{\Pi})^{\star} = (R^{\Gamma}_{\Pi^{\star}})$. Hence, $(S^{\Gamma}_{\Pi^{\star}}) \subseteq (R^{\Gamma}_{\Pi^{\star}})$

□

Lemma 11 (Existence Lemma for Proper Canonical Models)**.** *Let* $A \in At(\Gamma)$ *and* $\langle t,(f,b)\rangle\varphi \in FL(\Gamma)$. *Then,* $\langle t,(f,b)\rangle\varphi \in A \leftrightarrow$ *exists* $B \in At(\Gamma), AR^{\Gamma}_{\Pi}B, t \prec \delta^{\Gamma}_{c}(A)$ *and* $\varphi \in B$.

Proof. $\Rightarrow$ Let $\langle t,(f,b)\rangle\varphi \in A$. From Lemma 7 (Existence Lemma for canonical models), there is an atom $B \in At(\Gamma)$ where $AS^{\Gamma}_{\Pi}B$, $t \prec \delta^{\Gamma}_{c}(A)$ and $\varphi \in B$. From Lemma 10, $S^{\Gamma}_{\Pi} \subseteq R^{\Gamma}_{\Pi}$. Therefore, there is an atom $B \in At(\Gamma)$ where $AR^{\Gamma}_{\Pi}B$, $t \prec \delta^{\Gamma}_{c}(A)$ and $\varphi \in B$.
$\Leftarrow$ Let B an atom, $B \in At(\Gamma), AR_{\Pi}B, t \prec \delta^{\Gamma}_{c}(A)$ and $\varphi \in B$. The proof follows by induction on the program $\Pi = (f,b)$ as follows:

- a canonical program π_i: this case is straightforward as from Definition 17, $S_{\pi_i} = R_{\pi_i}$, and consequently $AS_{\pi_i}B, t \prec \delta^{\Gamma}_{c}(A)$ and (i) $\varphi \in B$. From Lemma 7 and (i), $\langle t,(f,b)\rangle\varphi \in A$.
- $\Pi^{\star}$: from Definition 17, $R_{\Pi^{\star}} = R^{\star}_{\Pi}$. Then, let $B \in At(\Gamma), AR_{\Pi^{\star}}B, t \prec \delta^{\Gamma}_{c}(A)$ and $\varphi \in B$. This means that there is a finite nondeterministic number n where $AR_{\Pi^{\star}}B = AR_{\Pi}A_1R_{\Pi}A_2 \ldots R_{\Pi}A_n$, where $A_n = B$. The proof proceeds by induction on n:
  - $n = 1$: $AR_{\Pi}B$ and $\varphi \in B$. Therefore, from Lemma 7, $\langle t,(f,b)\rangle\varphi \in A$. From axiom Rec, one may derive $\Vdash \langle t,(f,b)\rangle\varphi \to \langle t,(f,b)^{\star}\rangle\varphi$. By the definition of FL and A's maximality (as it is an atom of Γ) $\langle t,(f,b)^{\star}\rangle\varphi \in A$.
  - $n > 1$: From the previous proof step and the induction hypothesis, $\langle t,(f,b)^{\star}\rangle \in A_2$ and $\langle t,(f,b)\rangle\langle t,(f,b)^{\star}\rangle \in A_1$. From axiom Rec, one can derive $\Vdash \langle t,(f,b)\rangle\langle t,(f,b)^{\star}\rangle\varphi \to \langle t,(f,b)^{\star}\rangle\varphi$. By the definition of FL, and A's maximality (as it is an atom of Γ), $\langle t,(f,b)^{\star}\rangle\varphi \in A$.

□

Lemma 12 (Truth Lemma for Proper Canonical Models)**.** *Let* $\mathscr{M}^{\Gamma}_{c} = \langle At(\Gamma), \Pi, R^{\Gamma}_{\Pi}, \delta^{\Gamma}_{\Pi}, \lambda_c, V^{\Gamma}_{\Pi}\rangle$ *a proper canonical model constructed over a formula* γ. *For all atoms A and all* $\varphi \in FL(\gamma)$. $\mathscr{M}, A \Vdash \varphi \leftrightarrow \varphi \in A$.

Proof. The proof proceeds by induction over φ.

- Induction basis: φ is a proposition p. Therefore, $\mathscr{M}^{\Gamma}_{c}, A \Vdash p$ holds from Definition 17 as $V^{\Gamma}_{c}(p) = \{A \in At(\Gamma) \mid p \in A\}$.
- Induction hypothesis: suppose φ is a non atomic formula ψ. Then, $\mathscr{M}, A \Vdash \varphi \iff \varphi \in A$, ψ a strict subformula of φ.

- Inductive step: let us prove it holds for the following cases (we show only for modal cases):
  - Case $\varphi = \langle t,(f,b)\rangle\phi$. Then, $\mathscr{M}_c^\Gamma, A \Vdash \langle t,(f,b)\rangle\phi \iff \langle t,(f,b)\rangle\phi \in A$:
    $\Rightarrow$ Let $\mathscr{M}_c^\Gamma, A \Vdash \langle t,(f,b)\rangle\phi$. From Definition 14, there is an atom B where $AS_\Pi^\Gamma B$ and $\phi \in B$. By Lemma 11, $\langle t,(f,b)\rangle\phi \in A$. Therefore, it holds.
    $\Leftarrow$ Let $\mathscr{M}_c^\Gamma, A \nVdash \langle t,(f,b)\rangle\phi$. From Definition 16's valuation function V_c^Γ and Lemma 5, we have $\mathscr{M}_c^\Gamma, A \Vdash \neg\langle t,(f,b)\rangle\phi$. Therefore, for every B where $AS_\Pi^\Gamma B, \mathscr{M}_c^\Gamma \Vdash \neg\phi$. From the induction hypothesis, $\phi \notin B$. Hence, from Lemma 11 $\langle t,(f,b)\rangle\phi \notin A$.
  - Case $\varphi = \langle t,(f,b)^\star\rangle\phi$. Then, $\mathscr{M}_c^\Gamma, A \Vdash \langle t,(f,b)^\star\rangle\phi \iff \langle t,(f,b)^\star\rangle\phi \in A$:
    $\Rightarrow$ Let $\mathscr{M}_c^\Gamma, A \Vdash \langle t,(f,b)^\star\rangle\phi$. From Definition 14, there is a state B where $AS_{\Pi^\star}^\Gamma B$ and $\phi \in B$. By Lemma 7, $\langle t,(f,b)^\star\rangle\phi \in A$. Therefore, it holds.
    $\Leftarrow$ Let $\mathscr{M}_c^\Gamma, A \nVdash \langle t,(f,b)^\star\rangle\phi$. From Definition 16's valuation function V_c^Γ and Lemma 5, we have $\mathscr{M}_c^\Gamma, A \Vdash \neg\langle t,(f,b)^\star\rangle\phi$. Therefore, for every B where $AS_{\Pi^\star}^\Gamma B, \mathscr{M}_c^\Gamma, B \Vdash \neg\phi$. From the induction hypothesis, $\phi \notin B$. Hence, From Lemma 7, $\langle t,(f,b)^\star\rangle\phi \notin A$.

□

Theorem 1 (Completeness of *ReLo*)**.** *Proof.* For every consistent formula A, a canonical model $\mathscr{M}$ can be constructed. From Lemma 5, there is an atom $A' \in At(A)$ with $A \in A'$, and from Lemma 12, $\mathscr{M}, A' \Vdash A$. Therefore, *ReLo*'s modal system is complete with respect to the class of proper canonical models as Definition 17 proposes. □

5 Conclusions and Further Work

Reo is a widely used tool to model new systems out of the coordination of already existing pieces of software. It has been used in a variety of domains, drawing the attention of researchers from different locations around the world. This has resulted in Reo having many formal semantics proposed, each one employing different formalisms: operational, co-algebraic, and coloring semantics are some of the types of semantics proposed for Reo.

This work extends *ReLo*, a dynamic logic to reason about Reo models. We have discussed its core definitions, syntax, semantic notion, providing soundness and completeness proofs for it. *ReLo* naturally subsumes the notion of Reo programs and models in its syntax and semantics, and implementing its core concepts in Coq enables the usage of Coq's proof apparatus to reason over Reo models with *ReLo*.

Future work may consider the integration of the current implementation of *ReLo* with ReoXplore[4], a platform conceived to reason about Reo models, and extensions to other Reo semantics. Investigations and the development of calculi for *ReLo* are also considered for future work.

References

[1] JR Abrial (1991): *B-Tool Reference Manual. B-Core (UK) Ltd.*

[2] Farhad Arbab (2004): *Reo: a channel-based coordination model for component composition. Mathematical Structures in Computer Science* 14(3), p. 329–366, doi:10.1017/S0960129504004153.

[3] Farhad Arbab (2006): *Coordination for Component Composition. Electronic Notes in Theoretical Computer Science* 160, pp. 15 – 40, doi:10.1016/j.entcs.2006.05.013. Proceedings of the International Workshop on Formal Aspects of Component Software (FACS 2005).

[4] https://github.com/frame-lab/ReoXplore2

[4] Farhad Arbab, Natallia Kokash & Sun Meng (2008): *Towards using reo for compliance-aware business process modeling*. In: *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, Springer, pp. 108–123, doi:10.1007/978-3-540-88479-8_9.

[5] Farhad Arbab & Jan JMM Rutten (2002): *A coinductive calculus of component connectors*. In: *International Workshop on Algebraic Development Techniques*, Springer, pp. 34–55, doi:10.1007/978-3-540-40020-2_2.

[6] Colin Atkinson & Thomas Kuhne (2003): *Model-driven development: a metamodeling foundation*. *IEEE software* 20(5), pp. 36–41, doi:10.1109/MS.2003.1231149.

[7] Christel Baier (2005): *Probabilistic Models for Reo Connector Circuits*. *J. UCS* 11(10), pp. 1718–1748.

[8] Christel Baier, Marjan Sirjani, Farhad Arbab & Jan Rutten (2006): *Modeling component connectors in Reo by constraint automata*. *Science of computer programming* 61(2), pp. 75–113, doi:10.1016/j.scico.2005.10.008.

[9] Mario Benevides, Bruno Lopes & Edward Hermann Haeusler (2018): *Towards reasoning about Petri nets: A Propositional Dynamic Logic based approach*. *Theoretical Computer Science* 744, pp. 22–36, doi:10.1016/j.tcs.2018.01.007.

[10] Patrick Blackburn, M De Rijke & Y Venema (2001): *Cambridge tracts in theoretical computer science*.

[11] Roberto Bruni & Ugo Montanari (2000): *Zero-safe nets: Comparing the collective and individual token approaches*. *Information and computation* 156(1-2), pp. 46–89, doi:10.1006/inco.1999.2819.

[12] Dave Clarke (2007): *Coordination: Reo, nets, and logic*. In: *International Symposium on Formal Methods for Components and Objects*, Springer, pp. 226–256, doi:10.1007/978-3-540-92188-2_10.

[13] Erick Grilo & Bruno Lopes (2020): *ReLo: a dynamic logic to reason about Reo circuits1*. In: *Pre-Proceedings of the 15th International Workshop on Logical and Semantic Frameworks, with Applications (LSFA)*, p. 32.

[14] Erick Grilo, Daniel Toledo & Bruno Lopes (2022): *A logical framework to reason about Reo circuits*. *Journal of Applied Logics* 9, pp. 199–254.

[15] David Harel, Dexter Kozen & Jerzy Tiuryn (2001): *Dynamic logic*. In: *Handbook of philosophical logic*, Springer, pp. 99–217, doi:10.1007/978-94-017-0456-4_2.

[16] Daniel Jackson (2002): *Alloy: a lightweight object modelling notation*. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11(2), pp. 256–290, doi:10.1145/505145.505149.

[17] Sung-Shik TQ Jongmans & Farhad Arbab (2012): *Overview of Thirty Semantic Formalisms for Reo*. *Scientific Annals of Computer Science* 22(1), doi:10.7561/SACS.2012.1.201.

[18] Joachim Klein, Sascha Klüppelholz, Andries Stam & Christel Baier (2011): *Hierarchical modeling and formal verification. An industrial case study using Reo and Vereofy*. In: *International Workshop on Formal Methods for Industrial Critical Systems*, Springer, pp. 228–243, doi:10.1007/978-3-642-24431-5_17.

[19] John C Knight (2002): *Safety critical systems: challenges and directions*. In: *Proceedings of the 24th International Conference on Software Engineering*, ACM, pp. 547–550.

[20] Natallia Kokash & Farhad Arbab (2011): *Formal design and verification of long-running transactions with extensible coordination tools*. *IEEE Transactions on Services Computing* 6(2), pp. 186–200, doi:10.1109/TSC.2011.46.

[21] Natallia Kokash, Behnaz Changizi & Farhad Arbab (2010): *A semantic model for service composition with coordination time delays*. In: *International Conference on Formal Engineering Methods*, Springer, pp. 106–121, doi:10.1007/978-3-642-16901-4_9.

[22] Natallia Kokash, Christian Krause & Erik De Vink (2012): *Reo+ mCRL2: A framework for model-checking dataflow in service compositions*. *Formal Aspects of Computing* 24(2), pp. 187–216, doi:10.1007/s00165-011-0191-6.

[23] Natallia Kokash, Christian Krause & Erik P de Vink (2010): *Data-aware design and verification of service compositions with Reo and mCRL2*. In: *Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 2406–2413, doi:10.1145/1774088.1774590.

[24] Saul A Kripke (1959): *A completeness theorem in modal logic*. *The journal of symbolic logic* 24(1), pp. 1–14, doi:10.2307/2964568.

[25] Yi Li & Meng Sun (2015): *Modeling and verification of component connectors in Coq*. Science of Computer Programming 113, pp. 285–301, doi:10.1016/j.scico.2015.10.016.

[26] Yi Li, Xiyue Zhang, Yuanyi Ji & Meng Sun (2017): *Capturing Stochastic and Real-Time Behavior in Reo Connectors*. In: Formal Methods: Foundations and Applications - 20th Brazilian Symposium, SBMF 2017, Recife, Brazil, November 29 - December 1, 2017, Proceedings, pp. 287–304, doi:10.1007/978-3-319-70848-5_18.

[27] Yi Li, Xiyue Zhang, Yuanyi Ji & Meng Sun (2019): *A Formal Framework Capturing Real-Time and Stochastic Behavior in Connectors*. Science of Computer Programming, doi:10.1016/j.scico.2019.02.005.

[28] Mohammad Reza Mousavi, Marjan Sirjani & Farhad Arbab (2006): *Formal semantics and analysis of component connectors in Reo*. Electronic Notes in Theoretical Computer Science 154(1), pp. 83–99, doi:10.1016/j.entcs.2005.12.034.

[29] M. Saqib Nawaz & Meng Sun (2018): *Reo2PVS: Formal Specification and Verification of Component Connectors*. In: The 30th International Conference on Software Engineering and Knowledge Engineering, Hotel Pullman, Redwood City, California, USA, July 1-3, 2018., pp. 391–390, doi:10.18293/SEKE2018-024.

[30] Jonathan S Ostro (1992): *Formal methods for the specification and design of real-time safety critical systems*. Journal of Systems and Software 18(1), pp. 33–60, doi:10.1016/0164-1212(92)90045-L.

[31] Mike P Papazoglou (2003): *Service-oriented computing: Concepts, characteristics and directions*. In: Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on, IEEE, pp. 3–12, doi:10.1109/WISE.2003.1254461.

[32] Bahman Pourvatan, Marjan Sirjani, Hossein Hojjat & Farhad Arbab (2009): *Automated analysis of Reo circuits using symbolic execution*. Electronic Notes in Theoretical Computer Science 255, pp. 137–158, doi:10.1016/j.entcs.2009.10.029.

[33] Meng Sun & Yi Li (2014): *Formal modeling and verification of complex interactions in e-government applications*. In: Proceedings of the 8th International Conference on Theory and Practice of Electronic Governance, ACM, pp. 506–507, doi:10.1145/2691195.2691296.

[34] Samira Tasharofi & Marjan Sirjani (2009): *Formal modeling and conformance validation for WS-CDL using Reo and CASM*. Electronic Notes in Theoretical Computer Science 229(2), pp. 155–174, doi:10.1016/j.entcs.2009.06.034.

[35] Xiyue Zhang, Weijiang Hong, Yi Li & Meng Sun (2016): *Reasoning about connectors in Coq*. In: International Workshop on Formal Aspects of Component Software, Springer, pp. 172–190, doi:10.1007/978-3-319-57666-4_11.

[36] Xiyue Zhang, Weijiang Hong, Yi Li & Meng Sun (2019): *Reasoning about connectors using Coq and Z3*. Science of Computer Programming 170, pp. 27–44, doi:10.1016/j.scico.2018.10.002.