Erick Grilo*, Thiago Cordeiro**
e Bruno Lopes***

DOI: http://doi.org/10.32334/oqnfp.2021n49a803

Interpreting Reo Circuits as PDL models[1]

Interpretando Circuitos Reo como modelos PDL

Abstract

Reo is a coordination-based language with the proposal of connecting different systems and interfaces. It aims to develop communication between different systems with a high level of abstraction and without many restrictions. PDL is a multimodal logic tailored to reason about programs. It is proved to be sound, complete, decidable and has a simple Kripke semantics. This work intends to provide an interpretation of Reo circuits as PDL models. These are the first steps towards providing a dynamic logic tailored to reason directly about Reo circuits.

Keywords: Reo circuits; PDL models; coordination-based language; multi-modal logic.

1 Supported by FAPERJ, CAPES and CNPq.

* Instituto de Computação. Universidade Federal Fluminense. Contato: simas_grilo@id.uff.br
** Instituto de Computação. Universidade Federal Fluminense. Contato: thiago_cordeiro@id.uff.br
*** Instituto de Computação. Universidade Federal Fluminense. Contato: bruno@ic.uff.br

Recebido em: 10/04/2021 Aceito em: 17/11/2021

Resumo

Reo é uma linguagem baseada em coordenação com a proposta de conectar diferentes sistemas e interfaces. Objetiva modelar a comunicação entre diferentes sistemas com um alto nível de abstração e sem muitas restrições. PDL é uma lógica multimodal adaptada para raciocinar sobre programas, consistente, completa, decidível e com uma semântica de Kripke simples. Este trabalho pretende fornecer uma interpretação dos circuitos Reo como modelos PDL. Esses são os primeiros passos para fornecer uma lógica dinâmica próppria para raciocinar diretamente sobre os circuitos Reo.

Palavras-chave: circuitos Reo; modelos PDL; linguagem baseada em coordenação; lógica multimodal

1. Introduction

Reo [1] is a coordination-based language that emerged in the 90's with the proposal of connecting different systems and interfaces, and develop communication between different systems. With a high level of abstraction and without many restrictions, Reo's range of possibilities is widefull. Reo is used in distributed and parallel systems and inter-process communication [4]. As a basis for its operation, Constraint Automata formalize its components.

Constraint Automata (CA) [4] are used to define a formal semantics for Reo. They allow us to analyse the modelling of a system as the behaviour of information over time. A Constraint Automaton extends finite automata theory twofoldly: (i) its input has a time condition and (ii) there exists a logical constraint to enable a transition.

Propositional Dynamic Logic (PDL) [6] is a logic tailored to reason about programs. A formalisation in PDL leads to the possibility of reasoning about the behaviour of programs. Such behaviour includes properties as a correction in relation to its requirements, fairness, liveliness and equivalence. A formula in PDL has the form "$\langle\pi\rangle p$" meaning that there exists an execution of the program π such that after it p is true, supposing that π halts.

PDL has been widely used to reason about distributed systems. Some encodings and extensions of net systems were already proposed and compose a huge theoretical framework [5, 10, 11, 17, 18].

An example of PDL usage in industry can be found in [15], where IBM uses PDL to formalize the zSeries e-server, thereby guaranteeing properties such as a termination and determinism. XPath, which is a major element in the XSLT standard, may be encoded in PDL [12, 13].

This paper presents an encoding of CA in a PDL model. The goal is to translate a Constraint Automaton into a PDL theory and for this theory to build a PDL model. The input of the CA (Timed Data Stream) is translated to the valuation function of the model. It leads to the usage of PDL theoretical framework to reason about Reo circuits interpreted as Constraint Automata and is the first step towards a Dynamic Logic to reason diretctly about Reo circuits in a more expressive semantics.

2. Related Work

Reo's capabitiy of model real-world interaction scenarios has resulted in a great effort directed in formalizing means in order to verify properties of Reo models [19, 20, 21, 22, 23, 24, 27, 29], with its application domain ranging from component-based software engineering to the verification of e-governance applications [29]. This has also contributed to the existence of Reo's many formal semantics, each focusing on specific properties that can be captured in Reo models, such as time or action constraints [25].

The approach presented by Klein et al. [19] provides a platform to reason about Reo models using Vereofy[1], a model checker for component-based systems, while Pourvatan et al. [24] employ Constraint Automata in reasoning about Reo models by means of symbolic execution. 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. [8] encode Reo in mCRL2 model checker using Constraint Automata and its main variants, encoding their behaviour as mCRL2 processes and enabling the expression of properties regarding deadlocks and data constraints which

1 http://www.vereofy.de

depend upon time. Mouzavi et al. [23] propose an approach based on Maude in order to model checking Reo models, encoding Reo's operational semantics of the connectors, and Li et al. [222,28] respectively propose a real-time extension to Reo, implementing new channels and relying on Stochastic Timed Automata for Reo (STA) as its formal semantics, also providing a translation of STA to PRISM[2] for model checking and a translation of Reo models to PVS. UPPAAL[3] model checker has also been employed in the verification of Reo connectors employing the usage of Timed Constraint Automata [26] to build the corresponding UPPAAL model, and in the simulation of Hybrid Reo Connectors [27]. Zhang et al. [30, 31] employ Coq and Z3 proof assistants to reason over Reo connectors by encoding Reo's connectors.

The usage of Constraint Automata as formal semantics for Reo is advantageous when considering a systematic notion of a Reo connector. In this approach, states of the automaton denote its possible data configurations, enabling one to see "How the system behaves as a whole". When not considering only Reo, Constraint Automata provides a formal basis to reason about general coordination modeling languages, not only Reo [4]. The approach proposed in this paper presents an algorithm to encode Reo models denoted by Constraint Automata as PDL sentences, its soundness proofs and an implementation of this algorithm.

3. Constraint Automata: a formal semantics to Reo

In this Section, we provide a brief overview over what is Reo, addressing its main characteristics regarding how it works, along with a modelling example. We also introduce Constraint Automata, Reo's formal semantic employed in the presented work, briefly recovering the main concepts regarding the theory introduced by Baier et al. [4].

2 https://www.prismmodelchecker.org

3 http://www.uppaal.org/

3.1. Reo as a modelling language

Reo [2, 3] is a graphic-based language used in modelling coordination between systems, enabling the modelling of systems which are built by reusing fragments of existing software. It is based on channels, where complex coordinators (i.e., Reo models) are compositionally built from simpler ones. These complex coordinators are called connectors and compose the modelling language's core. Its main goal is to act as a "glue language", a language that connects ("glues together") instances of different components that act together in a component based system by orchestrating how these entities interact between themselves, providing the model of the code that performs such orchestration.

Reo focuses on modelling connectors. These structures denote how modelled entities communicate (send/receive data) with each other by means of this connector, instead of focusing on particular features of entities that are connected, communicate and work together through those connectors. These entities, called component instances in Reo, may be modules of sequential code, objects, agents, processes, web services and any other software component [2]. The only means of performing such operations are through channel ends known by these entities. In other words, entities must communicate only with other entities connected in Reo connectors they are part of.

A node in Reo is defined as a logical organization denoting the structure of how channel ends are linked to each other in Reo connectors. Nodes composing channel ends in Reo can be either source nodes, sink nodes or mixed nodes. Source nodes are nodes that accepts data into the channel (i.e., nodes that serves as gateway to data flow into the channel), while sink nodes are nodes where data flows out of the channel and mixed nodes are nodes that act both as source nodes and sink nodes (not simultaneously).

A Reo channel is designed to tunnel point-to-point communication between exactly two different Reo nodes. Each channel has its own previously defined behavior, however Reo also enables users to formalize their own channels. It also enables one to build complex connectors, where the composition of Reo channels (predefined or user-provided) are the core of these connectors. Figure 1 shows the basic set of connectors as seen in [8].

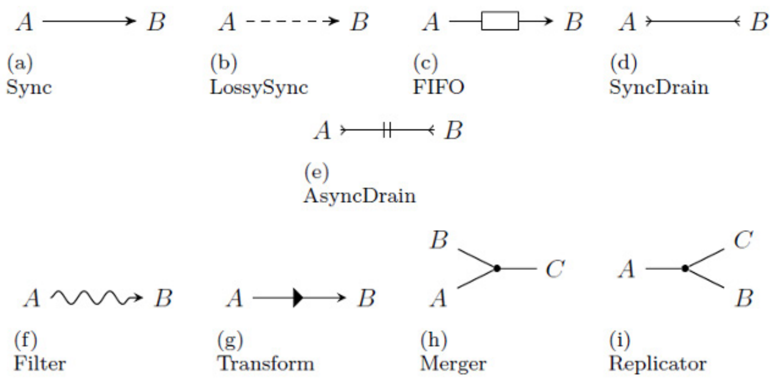


Figure 1: Canonical Reo connectors

As an modelling example employing Reo, we introduce the model presented in Figure 2. It consists of a Reo connector named Sequencer[4]. This connector models scenarios regarding the sequencing of processes which are interconnected by means of this connector, enabling the study and verification of this scenario, along with properties one wish to guarantee regarding how data flow between the interconnected entities.

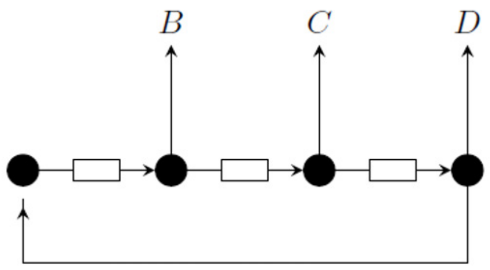


Figure 2: Modelling of the Sequencer in Reo

4 http://reo.project.cwi.nl/v2/\#examples-of-complex-connectors

3.2 Constraint Automata as Reo's formal semantics

Constraint Automata (CA) [4] are defined as the most basic operational models for Reo. The present work focuses on Constraint Automata as proposed by Reo creators. It consists of a simple yet powerful formalism tailored to reason about Reo models which is formally defined in Definition 3.2.

Definition 1 (Constraint Automata)
A Constraint Automaton is a tuple A=(Q,Names,→,Q_0) *where*
Q *is a finite set of states, denoting possible configurations of A*
Names *is a finite set of port names,*
→ : ***Q*** *×* ***2***^{Names} *×* ***DC*** *×* ***Q*** *is the transition relation with DC a set of (propositional) Data Constraints, and*
Q0⊆ Q *is the set of initial states.*

Constraint Automata are understood as a variation of Finite Automata where the transitions do not depend only on the value seen in the input, but also on data flowing into ports p ∈ Names. By using Constraint Automata as Reo's formal semantics, an automaton's states stnds for the possible configurations (i.e., possible data flow between the entities modelled), the set Names contains abstractions of the software components modelled in the Reo connectors, and transitions models how the configurations depicted by its states are reached.

Such formalism proposes grounds for modeling and verifying properties regarding coordination mechanisms by usage of formal methods. An example of this approach is proposed by Navidpour et al. [14] which applies model checking to validate properties of automata specified in temporal logic.

In order to model data flow within Reo connectors, Constraint Automata are seen as Timed Data Stream acceptors. A Timed Data Stream (TDS) is a pair containing two (possibly infinite) streams that describe, respecitvely, the behavior of a given port, namely the data flow and the time instant each data item in this flow is seen in the port. An input for an constraint automaton is then defined as Θ ∈ TDS^{Names}, Θ containing a TDS for each port p ∈ Names, which is formalized as depicted in Definition .

Definition 2 (TDS^{Names}) TDS^{Names} *is a set containing a TDS for each port* A$_i$ ∈ Names *as*

$$TDS^{\mathcal{N}ames} = \{((\alpha_1, a_1), (\alpha_2, a_2), \ldots, (\alpha_n, a_n)) \colon (\alpha_i, a_i) \in TDS, i = 1, 2, \ldots, n,$$
$$with\ n = |\mathcal{N}ames|.\ \}$$

Therefore, Constraint Automata supports verification of Reo models upon specification of structural properties, regarding how its configuration should change (based on the automaton's transitions, which models how data flow in Reo channels) and verification of specific situations, regarding a given data flow as input for an automaton depicted by Θ, with TDS^{Names}, assuring whether a specific scenario can be achieved in the corresponding Reo model.

4. Propositional Dynamic Logic

Propositional Dynamic Logic (PDL) [6] is a logic tailored to reason about programs. It extends multimodal K logic by means of indexing modalities with programs. A formula in PDL has the form $\langle\pi\rangle\phi$ which means that "after some execution of the program π, ϕ holds, supposing that π halts." A box correspondent formula $[\pi]\phi$ means that "after any execution of the program π, ϕ holds, supposing that π halts."

Definition 3 (PDL language) *The language of PDL is defined as*
$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \varphi_1 \leftrightarrow \varphi_2 \mid \langle\pi\rangle\varphi \mid [\pi]\varphi$,
where $\pi ::= \alpha \mid \pi_1;\pi_2 \mid \pi_1 \cup \pi_2 \mid \pi^\star \mid \varphi?$.

Let $p \in \Phi$, *such that* Φ *is an enumerable set of propositional symbols and* $\alpha \in \Pi$, *where* Π *is an enumerable set of basic programs.*

Definition 4 *A frame for PDL is a tuple* $F = \langle W, R_\pi\rangle$ *where*
- W *is a non-empty set of states;*
- R_α *is a binary relation over* W, *for each basic program a* $\in \Pi$;
- *We can inductively define a binary relation* R_π, *for each non-basic program* π, *as follows*
- $R_{\pi_1;\pi_2} = R_{\pi_1} \circ R_{\pi_2}$,
- $R_{\pi_1 \cup \pi_2} = R_{\pi_1} \cup R_{\pi_2}$,
- $R_{\pi^\star} = R_\pi^\star$, *where* $R_\pi^\star$, *where denotes the reflexive transitive closure of* R_π,
- $R_\varphi? = \{(w,w) \mid w \in W$ *and* $M, w \vDash \varphi$, *according to Definition .*

Definition 5 *A model for* PDL *is a pair* $M = \langle F, V\rangle$, *where* F *is a* PDL *frame and* V *is a valuation function* $V: \Phi \rightarrow 2^W$.

The semantical notion of satisfaction for PDL is defined as follows:

> ***Definition 6*** *Let* M= ⟨F,V⟩ *be a model. The notion of satisfaction of a formula* ϕ *in a model* M *at a state* w *is denoted by notation* $M,w \vDash \varphi$ *and can be inductively defined as follows:*
>
> - $M, w \vDash p$ iff $w \in V(p)$
> - $M, w \vDash \top$ *always;*
> - $M, w \vDash \neg \varphi$ iff $M, w \vDash \varphi$;
> - $M, w \vDash \varphi_1 \wedge \varphi_2$ iff $M, w \vDash \varphi_1$ *and* $M, w \vDash \varphi_2$;
> - $M, w \vDash \varphi_1 \vee \varphi_2$ iff $M, w \vDash \varphi_1$ *or* $M, w \vDash \varphi_2$;
> - $M, w \vDash \varphi_1 \rightarrow \varphi_2$ iff $M, w \vDash \varphi_1$ *or* $M,w \vDash \varphi_2$;
> - $M, w \vDash \langle\pi\rangle \varphi$ iff *there is* $w' \in$ *such that* $wR_\pi w'$ *and* $M,w' \vDash \varphi$;
> - $M, w \vDash [\pi] \varphi$ iff *for any* $w' \in$ if $wR_\pi w'$ *then* $M, w' \vDash \varphi$;.

PDL is proved to be sound, complete and decidable. For more details on PDL see [7].

5. From Constraint Automata to PDL models

In this section we propose and explain an algorithm which aims to build PDL models from Constraint Automata. The proposed approach aims to recover and translate each transition in an automaton's transition relation to sentences in PDL.

5.1. Denoting PDL theories from automata

The first part of the algorithm comprehend the modelling process of automata transitions into propositional logical formulae. In order to do so, let Φ be a set of propositional symbols. The idea is to consider each state of the automaton $q \in Q$, each port name of the automaton $n \in N$ and each data constraint $g \in D$ (with D as a set of data constraints) as a propositional symbol, resulting in $Q, N, D \subset \Phi$.

The next step is to formalize the automaton's transition relation $\vDash$ as logical sentences $p \in \Gamma$. Taking advantage of the automaton's transition relation as the notation $q \xrightarrow{\{N,g\}} p$ (we recover the transition notation introduced by Baier et al. [4]), an automaton's single transition may be defined as $(q_0 \wedge \bigwedge_{gi\in g} g_i \leftrightarrow \langle t \rangle\, q_1)$, denoting the same idea behind the firing of a transition in Constraint Automata: if the automaton is in state q_0 and the data constraint g (which may be composed of conjunction of zero or more propositional formulae g_i), then exists a transition t (the transition currently being modelled) which from q_0 reaches q_1.

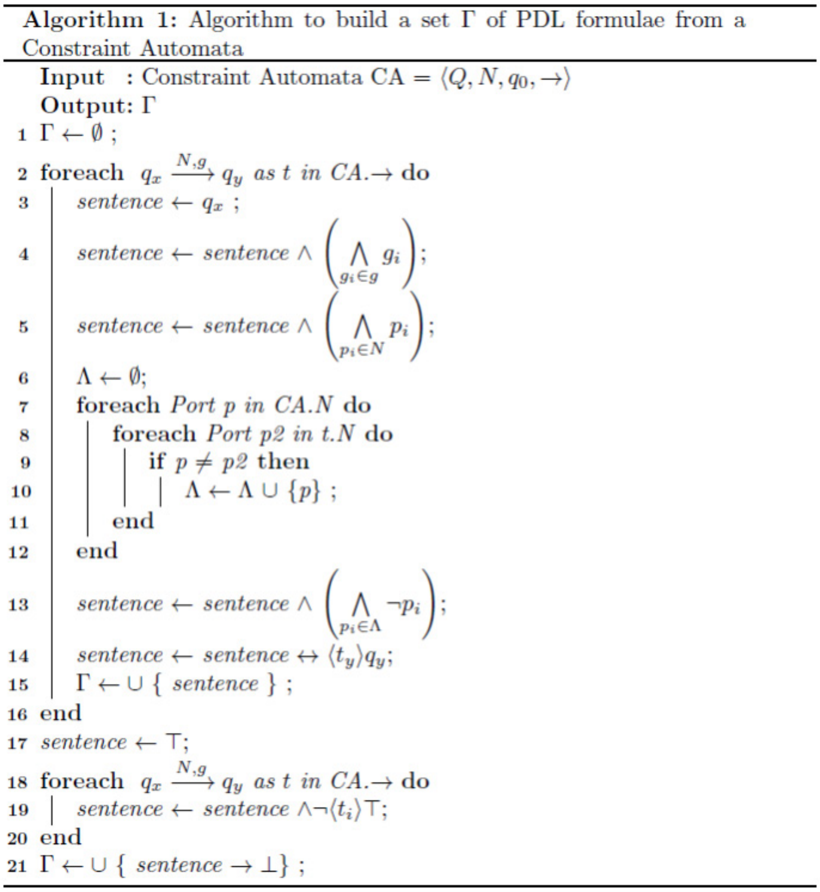
However, this approach still misses the point that, in order for a transition to fire in constraint automata, not only its data constraint must be satisfied, but the ports denoted by its set of port names are the only ones that must have data flowing at the time the transition is to be fired. Therefore, the procedure also needs to add this information in the derived formula. This is acheived as follows: port names pinN of the automaton which are in the transition t may be depicted as p_g and the remainder as p_{ng} , the ports which does not relate to the transition being evaluated. This can be rewritten as $\left(\bigwedge_{p_i \in p_g} p_i \;\wedge\; \bigwedge_{p_{ni} \in p_{ng}} \neg p_{ni}\right)$ Then, we add this new piece of information to the first proposed resulting formula, which now is $\left(q_0 \wedge \bigwedge_{g_i\in} g_i \wedge \bigwedge_{p_i\in p_g} p_i \wedge \bigwedge_{p_{ni}\in p_{ng}} \neg p_{ni}\right) \leftrightarrow \langle t\rangle q_1$. The set of logical sentences Γ can be defined as follows.

$$\Gamma = \left\{\bigwedge_{t_i\in\rightarrow}\left(\left(s_{t_i} \wedge \bigwedge_{g_i\in g_{t_i}} g_i \wedge \bigwedge_{p_i\in p_{g_{t_i}}} p_i \wedge \bigwedge_{p_{ni}\in p_{ng_{t_i}}} \neg p_{ni}\right) \leftrightarrow \langle t_i\rangle p_{t_i}\right)\right\}$$

There is still a situation which the above definition does not capture: the case where there is no transition available to be fired, resulting in an invalid configuration of the automaton. The corresponding PDL model must capture this scenario. We add a formula modelling each transition t_i that can't be fired. Γ is now rewritten then as below.

> ***Definition 7*** *Let* Γ *as the aforementioned set. The resulting set of formulae denoting all automata's transitions and how they behave is achieved by the following definition* $\Gamma = \Gamma \cup \left\{\bigwedge_{t_i\in\rightarrow}(\neg\langle t_i\rangle\top) \rightarrow\bot\right\}$.

Γ then models all transitions of a single constraint automaton, considering its initial state (s_{ti}), its corresponding context regarding transition firing (g_i), the ports $p \in N$ that must have data flowing (p_i) and the remainder of N which must not have any data flowing at the moment (p_{ni}). In what follows we briefly introduce and discuss the proposed algorithm.



Algorithm 1: Algorithm to build a set Γ of PDL formulae from a Constraint Automata

```
Input  : Constraint Automata CA = $\langle Q, N, q_0, \rightarrow\rangle$
Output: $\Gamma$
$\Gamma \leftarrow \emptyset$ ;
foreach $q_x \xrightarrow{N,g} q_y$ as $t$ in $CA.\rightarrow$ do
    $sentence \leftarrow q_x$ ;
    $sentence \leftarrow sentence \wedge \left(\bigwedge_{g_i \in g} g_i\right)$;
    $sentence \leftarrow sentence \wedge \left(\bigwedge_{p_i \in N} p_i\right)$;
    $\Lambda \leftarrow \emptyset$;
    foreach Port $p$ in $CA.N$ do
        foreach Port $p2$ in $t.N$ do
            if $p \neq p2$ then
                $\Lambda \leftarrow \Lambda \cup \{p\}$ ;
        end
    end
    $sentence \leftarrow sentence \wedge \left(\bigwedge_{p_i \in \Lambda} \neg p_i\right)$;
    $sentence \leftarrow sentence \leftrightarrow \langle t_y\rangle q_y$;
    $\Gamma \leftarrow \cup \{ sentence \}$ ;
end
$sentence \leftarrow \top$;
foreach $q_x \xrightarrow{N,g} q_y$ as $t$ in $CA.\rightarrow$ do
    $sentence \leftarrow sentence \wedge\neg\langle t_i\rangle\top$;
end
$\Gamma \leftarrow \cup \{ sentence \rightarrow \bot\}$ ;
```

Algorithm 1 describes each step executed in order to obtain Γ from a constriant automaton. The algorithm expects as input a constraint automaton, and from its transition relation, it builds Γ from the extracted information, executing the following (simplified) steps:

1. Initialization of the logical sentence (denotihg the result of the evaluation of the current transition);
2. Assign the transition's origin state to the sentence;
3. Iterate over the transition's whole data constraint, adding them to a conjunction with the sentence;

4. Iterate over the transition's set of port names, adding them to a conjunction with the sentence;
5. Iterate over the automaton's set of port names; check whether a given port is not in the transition's set of port names, and add it to a conjunction with the sentence;
6. Add the biconditional with the symbol denoting the existence of a transition t_i that reaches a state q_y;
7. Add the stopping condition considering an rejecting scenario (no transiitions could be fired).

5.2. Formalizing automata input: TDS^{Names} in PDL

The second part of the algorithm aims to formalize the notion of inputs for constraint automata in PDL as described in Section 3. From Baier et al. [4], recall that Constraint Automata are seen as TDS acceptors. We define a function, Val: $N \times W \times N \rightarrow D$, as a function that given a port name of the automaton A, a state w from the PDL model, and a natural number n denoting the index of the time stamp of TDS^{Names} it returns the current data assignment of the port A. Its definition is summarized in Algorithm 2.

```
Algorithm 2: Val definition
Function Val($A$,$w$, $n$):
    Input  : Port $A$
    Input  : state $w$
    Input  : Integer $n$
    if $\exists w' \in W$ such that $w' R_t w$ then then
        return Val($A$,$w'$,$n+1$)
    else
        return $\theta.\delta_A(n)$ ;
    end
return
```

Val is formalized as an auxiliary function that will be used in the process of formalizing TDS^{Names} in PDL. We use the valuation function as depicted in Definition , relying on Val, in order to implement TDS^{Names} in PDL. The valuation function then will have its semantics directed in evaluate whether the data flowing in a given set of ports of the constraint automaton (in a given time instant) satisfy any transition in the transition set Γ. Such function is denoted by Equation 1.

$$\forall x \in \Phi,\ x \in V(w) \text{ iff } \begin{cases} \text{if } x \approx g_{d_A = d_B} & \text{and } Val(A,w,0) = Val(B,w,0) \\ \text{if } x \approx g_{D_A = n \in \mathbb{N}} & \text{and } Val(A,w,0) = n \\ \text{if } x \approx g_{D_a} & \text{and } Val(A,w,0) \neq \emptyset \\ \text{if } x \approx g_{q_i} & \text{and } \exists w' \in W, w' R_{t_i} w \\ \text{if } x \approx g_{P_i} & \text{and } Val(p_i,w,0) \neq \emptyset \\ \text{if } x \approx g_{P_{ni}} & \text{and } Val(p_{ni},w,0) = \emptyset \end{cases} \quad (1)$$

5.3. Soundness Proofs

This Section presents the soundness proof of the aforementioned approach. It shows that for any Constraint Automata and a $\Theta \in TDS^{Names}$ there exists a PDL model.

Theorem 1 (The translation of a CA to a PDL model using Algorithm 1 preserves the semantics of the execution of the CA over Θ modelled as in Equation 1)

The output of Algorithm 1 is a set Γ of formulae. Let $\phi = \bigwedge_{i=1}^{n} \gamma_i$ where $\gamma_i \in \Gamma$ and $|\Gamma|=n$ Let π the composition of all PDL programs extracted from Γ such that if a transition t_i is sequenced by t_j we have that $t_i ; t_j$ and if t_i is an alternative transition from the same state to t_j we have that $t_i \cup t_j$.

So we build a PDL model such that $M=\langle W,R_\pi,V\rangle$ corresponds to Definition over π and let $w\ 0 \in V(s_0)$ (according to Equation (1)). It is needed to show that the state notion from the CA is preserved in the model M (cases 1 and 2), that the ports constraints over Θ are preserved (case 3) and that the model validates only what is valid on the CA (case 4).

Proof 1 The four cases are considered.

Case 1: While a constraint automaton is running, whenever its current state is $q_i \in Q$, the propositional symbol $s_i \in \Phi$ will have its truth value as true in the PDL model's current state $w \in W$, and $\forall s_j \in \Phi, j \neq i$ will have their truth values as false.

Given $q_0 \in Q$ a initial state of the constraint automaton, suppose that there is a corresponding initial state in the equivalent PDL model $w_0 \in W$, where $w_0 \in W$ and $\forall s_j \in \Phi, j \neq i$, $s_j \notin V(w_0)$, according to Equation (1).

If there is a firing transition of the automaton originating in a state $q_x \in Q$ bounded to a state $q_y \in Q$, from Γ there is at least one execution of a program

t_{s_x} leading to a state $w' \in W$ which after its execution, s_y will hold and $\forall s_j \in \Phi$, $j \neq y$, $s_j \notin V(w')$ also holds. This follows from the Algorithm 1 and from Equation (1).

Case 2: let $\rightarrow$ be the transition relation of the constraint automaton. Given a PDL state $w_i \in W$, there will be only relations $w_i R w_{ji}$, $w_j \in W$ that corresponds to the automaton's transition relation $\rightarrow$.

Given $q_0 \in Q$ an initial state of the automaton, there is a initial state in the corresponding PDL model $q_0 \in W$, where $s_i \in V(w_0)$ and $\forall s_j \in \Phi, j \neq i$. If there is a transition $\rightarrow i \in \rightarrow$ from q_0 to q_y, then there is a relation between two PDL states from w to $w' \in W$ by means fo the PDL program $t_{\rightarrow_i}$, with $M,w' \vDash g$ and $M,w' \nvDash p_i$, where $g \in \rightarrow_i$ and p_i is not a subformula of g, according to lines 4 to 14 in Algorithm 1, which define the existence of the transition and that it only fires if its corresponding data constraint g holds, and its associated set of port names are the only port in the automaton where data flows in the moment of this transition's firing, and from Equation (1).

If there is a reachable state by the transition previously defined $\rightarrow_i$, then there is a state $q_x \in Q$ of the automaton and a state $w_x \in W$ of the corresponding PDL model, where $s_i \in V(w_x)$ and $\forall s_j \in \Phi, j \neq i, s_j / \in V(w_x)$. If there is a transition $\rightarrow_i$ from q_x to q_y, then there is a transition by means of $t_{\rightarrow_i}$ to a state $w' \in W$, where $M,w' \vDash g$ and $w' \notin p_i$

This case is complete when there is no more transitions in $\rightarrow$. Line 21 of Algorithm 1 also guarantees the rejection of non accepting input, in the case during its execution, no transitions could be fired by the automaton.

Case 3: Let N the set of port names of the automaton and $N' \in t$ with $t \in \rightarrow$. In order to guarantee that the ports depicted by the port names $p_i \notin N'$ will not have data flow (enabling t's firing), the following property must be satisfied: $p_i \in N \neq p'_i \in N'$, for any p_i and p'_i.

Let $p_i \in N$ and $p'_i \in N'$ any port name. If $p_i \neq p'_i$, the propositional symbol corresponding to p_i is negated and included in the formula as follows: $\neq p_i$.

All combination of ports p_i e p'_i are iterated, and if $p_i \neq p'_i$, the negation of the propositional symbol corresponding to the port name p_i is added to the formula as p_i.

After iterating all possible port names combinations, the ports used in each transition are the only ones in the formula.

Case 4: Let $\Theta \in TDS^{Names}$. The behavior of the automaton must correspond to the behavior of the generated PDL model based on the same input Θ. In short, the model's behavior must be the same as the base automaton.

Suppose N is the set of port names involved with a transition $t \in \rightarrow$. Also, let w be the initial state of the corresponding PDL model, $w \leftarrow s_0$. In order to advance to another state, there must exist a state w' that is reachable b ya transition. In other words, given $w' \in W$, if $\exists$ w', with $w\, R_t\, w'$, this state is reachable according to the TDS Θ if $\forall_{p_i \in R_t}$ the function Val $(p_i, w', 0)$ returns a value, then exists a state w' which denotes a state of the automaton.

Although the model is potentially infinite, from any state it is possible to reach (co-inductively) an initial state.

Hence the behaviour of the CA is preserved in M

5.4 Implementation of the algorithm

The algorithm proposed in Section 5.1 and Section 5.2 is implemented in C. Its source code may be found on https://github.com/ThiagoUff/CA_PDL. The implementation is supplied with a text file containing the structure of the constraint automaton, followed by a parsing process of this file in order to validate its structure. A valid file returns a text file containing Γ's structure as proposed by the algorithm.

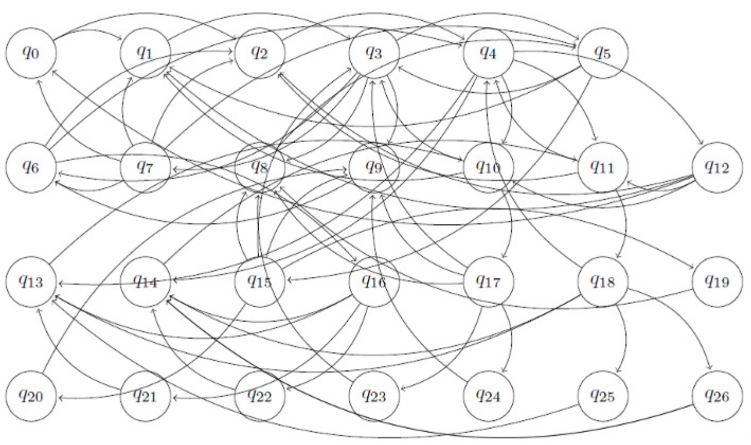


Figure 3: Constraint automaton of a Sequencer circuit (transitions labels are ommited due to lack of space)

5.5. Usage example

The Following example recovers the Reo circuit introduced in Figure 2 as denoted by the automaton in Figure 3. This automaton has 27 states, 7 Ports and 68 transitions with 142 Data Constraints, leading to 166 logical symbols. In what follows, the transitions of the automaton are transformed into logic sentences by iterating over the automaton's set of states and, for each state q ∈ Q, it applies the proposed algorithm with each transition which departing state is q.

1. state: q_0
   (a) $q_0 \xrightarrow{\{A\},d_a=0} q_1$ will result on the PDL sentence:

   $q_0 \wedge d_a = 0 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_0 \rangle q_1$

   (b) $q_0 \xrightarrow{\{A\},d_a=1} q_2$ will result in the PDL sentence:

   $q_0 \wedge d_a = 1 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_1 \rangle q_2$

2. state: q_1
   (a) $q_1 \xrightarrow{\{E,B\},d_e=0 \wedge d_e=d_b} q_3$ will result in the PDL sentence:

   $q_1 \wedge d_e = 0 \wedge d_e = d_b \wedge E \wedge B \wedge \neg A \wedge \neg C \wedge \neg D \wedge \neg G \wedge \neg H \leftrightarrow \langle t_2 \rangle q_3$

3. state: q_2
   (a) $q_2 \xrightarrow{\{E,B\},d_e=1 \wedge d_e=d_b} q_4$will result in the PDL sentence:

   $q_2 \wedge d_e = 1 \wedge d_e = d_b \wedge E \wedge B \wedge \neg A \wedge \neg C \wedge \neg D \wedge \neg G \wedge \neg H \leftrightarrow \langle t_3 \rangle q_4$

4. state: q_3
   (a) $q_3 \xrightarrow{\{A,C,G\},d_a=0 \wedge d_g=0 \wedge d_c=d_g} q_5$ will result in the PDL sentence:

   $q_3 \wedge d_a = 0 \wedge d_g = 0 \wedge d_c = d_g \wedge A \wedge C \wedge G \wedge \neg B \wedge \neg D \wedge \neg E \wedge \neg H \leftrightarrow \langle t_4 \rangle q_5$

   (b) $q_3 \xrightarrow{\{A,C,G\},d_a=1 \wedge d_g=0 \wedge d_c=d_g} q_6$ will result in the PDL sentence:

   $q_3 \wedge d_a = 1 \wedge d_g = 0 \wedge d_c = d_g \wedge A \wedge C \wedge G \wedge \neg B \wedge \neg D \wedge \neg E \wedge \neg H \leftrightarrow \langle t_5 \rangle q_6$

   (c) $q_3 \xrightarrow{\{G,C\},d_c=d_g \wedge d_g=0} q_7$ will result in the PDL sentence:

   $q_3 \wedge d_c = d_g \wedge d_g = 0 \wedge G \wedge C \wedge \neg A \wedge \neg B \wedge \neg D \wedge \neg E \wedge \neg H \leftrightarrow \langle t_6 \rangle q_7$

(d) $q_3 \xrightarrow{\{A\},d_a=0} q_8$ will result in the PDL sentence:

$$q_3 \wedge d_a = 0 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_7 \rangle q_8$$

(e) $q_3 \xrightarrow{\{A\},d_a=1} q_9$ will result in the PDL sentence:

$$q_3 \wedge d_a = 1 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_8 \rangle q_9$$

5. state: q_4

(a) $q_4 \xrightarrow{\{A,C,G\},d_a=0 \wedge d_g=1 \wedge d_g=d_c} q_{10}$ will result in the PDL sentence:

$$q_4 \wedge d_a = 0 \wedge d_g = 1 \wedge d_g = d_c \wedge A \wedge C \wedge G \wedge \neg B \wedge \neg D \wedge \neg E \wedge \neg H \leftrightarrow \langle t_9 \rangle q_{10}$$

(b) $q_4 \xrightarrow{\{A,C,G\},d_a=0 \wedge d_g=1 \wedge d_g=d_c} q_{11}$ will result in the PDL sentence:

$$q_4 \wedge d_a = 0 \wedge d_g = 1 \wedge d_g = d_c \wedge A \wedge C \wedge G \wedge \neg B \wedge \neg D \wedge \neg E \wedge \neg H \leftrightarrow \langle t_{10} \rangle q_{11}$$

(c) $q_4 \xrightarrow{\{C,G\},d_g=1 \wedge d_g=d_c} q_{12}$ will result in the PDL sentence:

$$q_4 \wedge d_g = 1 \wedge d_g = d_c \wedge C \wedge G \wedge \neg A \wedge \neg B \wedge \neg D \wedge \neg E \wedge \neg H \leftrightarrow \langle t_{11} \rangle q_{12}$$

(d) $q_4 \xrightarrow{\{A\},d_a=0} q_{13}$ will result in the PDL sentence:

$$q_4 \wedge d_a = 0 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{12} \rangle q_{13}$$

(e) $q_4 \xrightarrow{\{A\},d_a=1} q_{14}$ will result in the PDL sentence:

$$q_4 \wedge d_a = 1 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{13} \rangle q_{14}$$

6. state: q_5

(a) $q_5 \xrightarrow{\{B,D,E,H\},d_e=0 \wedge d_e=d_b \wedge d_h=0 \wedge d_h=d_b} q_3$ will result in the PDL sentence:

$$q_5 \wedge d_e = 0 \wedge d_e = d_b \wedge d_h = 0 \wedge d_h = d_b \wedge B \wedge D \wedge E \wedge H \wedge \neg A \wedge \neg C \wedge \neg G \leftrightarrow \langle t_{14} \rangle q_4$$

(b) $q_5 \xrightarrow{\{D,H\},d_h=0 \wedge d_d=d_h} q_1$ will result in the PDL sentence:

$$q_5 \wedge d_h = 0 \wedge d_d = d_h \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{15} \rangle q_1$$

(c) $q_5 \xrightarrow{\{B,E\},d_e=0 \wedge d_e=d_b} q_{15}$ will result in the PDL sentence:

$$q_5 \wedge d_e = 0 \wedge d_e = d_b \wedge B \wedge E \wedge \neg A \wedge \neg C \wedge \neg D \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{16} \rangle q_{15}$$

7. state: q_6

(a) $q_6 \xrightarrow{\{B,D,E,H\}, d_e=1 \wedge d_e=d_b \wedge d_h=0 \wedge d_h=d_b} q_4$ will result in the PDL sentence:

$$q_6 \wedge d_e = 1 \wedge d_e = d_b \wedge d_h = 0 \wedge d_h = d_b \wedge B \wedge D \wedge E \wedge H \wedge \neg A \wedge \neg C \wedge \neg G \leftrightarrow \langle t_{17} \rangle q_4$$

(b) $q_6 \xrightarrow{\{D,H\}, d_h=0 \wedge d_b=d_h} q_2$ will result in the PDL sentence:

$$q_6 \wedge d_h = 0 \wedge d_b = d_h \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{18} \rangle q_2$$

(c) $q_6 \xrightarrow{\{B,E\}, d_e=1 \wedge d_e=d_b} q_{16}$ will result in the PDL sentence:

$$q_6 \wedge d_e = 1 \wedge d_e = d_b \wedge B \wedge E \wedge \neg A \wedge \neg C \wedge \neg D \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{19} \rangle q_{16}$$

8. state: q_7

(a) $q_7 \xrightarrow{\{A,D,H\}, d_a=0 \wedge d_d=d_h \wedge d_h=0} q_1$ will result in the PDL sentence:

$$q_7 \wedge d_a = 0 \wedge d_d = d_h \wedge d_h = 0 \wedge A \wedge D \wedge H \wedge \neg B \wedge \neg C \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{20} \rangle q_1$$

(b) $q_7 \xrightarrow{\{D,H\}, d_h=0 \wedge d_d=d_h} q_0$ will result in the PDL sentence:

$$q_7 \wedge d_h = 0 \wedge d_d = d_h \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{21} \rangle q_0$$

(c) $q_7 \xrightarrow{\{A\}, d_a=0} q_5$ will result in the PDL sentence:

$$q_7 \wedge d_a = 0 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{22} \rangle q_5$$

(d) $q_7 \xrightarrow{\{A\}, d_a=1} q_6$ will result in the PDL sentence:

$$q_7 \wedge d_a = 1 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{23} \rangle q_6$$

(e) $q_7 \xrightarrow{\{A,D,H\}, d_a=1 \wedge d_d=d_h \wedge d_h=0} q_1$ will result in the PDL sentence:

$$q_7 \wedge d_a = 1 \wedge d_d = d_h \wedge d_h = 0 \wedge A \wedge D \wedge H \wedge \neg B \wedge \neg C \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{24} \rangle q_1$$

9. state: q_8

(a) $q_8 \xrightarrow{\{C,G\}, d_c=d_g \wedge d_g=0} q_5$will result in the PDL sentence:

$$q_8 \wedge d_c = d_g \wedge d_g = 0 \wedge C \wedge G \wedge \neg A \wedge \neg B \wedge \neg D \wedge \neg E \wedge \neg H \leftrightarrow \langle t_{25} \rangle q_5$$

10. state: q_9

(a) $q_9 \xrightarrow{\{C,G\}, d_c=d_g \wedge d_g=0} q_6$ will result in the PDL sentence:

$$q_9 \wedge d_c = d_g \wedge d_g = 0 \wedge C \wedge G \wedge \neg A \wedge \neg B \wedge \neg D \wedge \neg E \wedge \neg H \leftrightarrow \langle t_{26} \rangle q_6$$

11. state: q_{10}

(a) $q_{10} \xrightarrow{\{B,D,E,H\},d_e=0\wedge d_e=d_b\wedge d_h=1\wedge d_h=d_d} q_3$ will result in the PDL sentence:

$$q_{10} \wedge d_e = 0 \wedge d_e = d_b \wedge d_h = 1 \wedge d_h = d_d \wedge B \wedge D \wedge E \wedge H \wedge \neg A \wedge \neg C \wedge \neg G \leftrightarrow \langle t_{27}\rangle q_3$$

(b) $q_{10} \xrightarrow{\{D,H\},d_h=1\wedge d_h=d_d} q_1$ will result in the PDL sentence:

$$q_{10} \wedge d_h = 1 \wedge d_h = d_d \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{28}\rangle q_1$$

(c) $q_{10} \xrightarrow{\{E,B\},d_e=0\wedge d_b=d_e} q_{17}$ will result in the PDL sentence:

$$q_{10} \wedge, d_e = 0 \wedge d_b = d_e \wedge E \wedge B \wedge \neg A \wedge \neg C \wedge \neg D \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{29}\rangle q_{17}$$

12.state: q_{11}

(a) $q_{11} \xrightarrow{\{B,D,E,H\},d_e=1\wedge d_e=d_b\wedge d_h=1\wedge d_h=d_d} q_4$ will result in the PDL sentence:

$$q_{11} \wedge d_e = 1 \wedge d_e = d_b \wedge d_h = 1 \wedge d_h = d_d \wedge B \wedge D \wedge E \wedge H \wedge \neg A \wedge \neg C \wedge \neg G \leftrightarrow \langle t_{30}\rangle q_4$$

(b) $q_{11} \xrightarrow{\{D,H\},d_h=1\wedge d_h=d_d} q_2$ will result in the PDL sentence:

$$q_1 1 \wedge d_h = 1 \wedge d_h = d_d \wedge D \wedge H \wedge \neq A \wedge \neq B \wedge \neq C \wedge \neq E \wedge \neq G \leftrightarrow \langle t_{31}\rangle q_2$$

(c) $q_{11} \xrightarrow{\{E,B\},d_e=0\wedge d_b=d_e} q_{18}$ will result in the PDL sentence:

$$q_{11} \wedge, d_e = 0 \wedge d_b = d_e \wedge E \wedge B \wedge \neg A \wedge \neg C \wedge \neg D \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{32}\rangle q_{18}$$

13. state: q_{12}

(a) $q_{12} \xrightarrow{\{A,H,D\},d_a=1\wedge d_d=d_h\wedge d_h=1} q_2$ will result in the PDL sentence:

$$q_{12} \wedge d_a = 1 \wedge d_d = d_h \wedge d_h = 1 \wedge A \wedge H \wedge D \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg G \leftrightarrow \langle t_{33}\rangle q_2$$

(b) $q_{12} \xrightarrow{\{A,H,D\},d_a=0\wedge d_d=d_h\wedge d_h=1} q_1$ will result in the PDL sentence:

$$q_{12} \wedge d_a = 0 \wedge d_d = d_h \wedge d_h = 1 \wedge A \wedge H \wedge D \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg G \leftrightarrow \langle t_{34}\rangle q_1$$

(c) $q_{12} \xrightarrow{\{H,D\},d_h=1\wedge d_d=d_h} q_0$ will result in the PDL sentence:

$$q_{12} \wedge d_h = 1 \wedge d_d = d_h \wedge H \wedge D \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg G \leftrightarrow \langle t_{35}\rangle q_0$$

(d) $q_{12} \xrightarrow{\{A\},d_a=0} q_{10}$ will result in the PDL sentence:

$$q_{12} \wedge d_a = 0 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{36}\rangle q_{10}$$

(e) $q_{12} \xrightarrow{\{A\},d_a=1} q_{11}$ will result in the PDL sentence:

$$q_{12} \wedge d_a = 1 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{37}\rangle q_{11}$$

14. state: q_3

(a) $q_{13} \xrightarrow{\{C,G\},d_g=1\wedge d_c=d_g} q_{10}$ will result in the PDL sentence:

$$q_{13} \wedge d_g = 1 \wedge d_c = d_g \wedge C \wedge G \wedge \neg A \wedge \neg B \wedge \neg D \wedge \neg E \wedge \neg H \leftrightarrow \langle t_{38} \rangle q_{10}$$

15. state: q_4

(a) $q_{14} \xrightarrow{\{C,G\},d_g=1\wedge d_c=d_g} q_{11}$ will result in the PDL sentence:

$$q_{14} \wedge d_g = 1 \wedge d_c = d_g \wedge C \wedge G \wedge \neg A \wedge \neg B \wedge \neg D \wedge \neg E \wedge \neg H \leftrightarrow \langle t_{39} \rangle q_{11}$$

16. state: q_{15}

(a) $q_{15} \xrightarrow{\{A,D,H\},d_a=0\wedge d_d=d_h\wedge d_h=0} q_8$will result in the PDL sentence:

$$q_{15} \wedge d_a = 0 \wedge d_d = d_h \wedge d_h = 0 \wedge A \wedge D \wedge H \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{40} \rangle q_8$$

(b) $q_{15} \xrightarrow{\{A,D,H\},d_a=1\wedge d_d=d_h\wedge d_h=0} q_9$ will result in the PDL sentence:

$$q_{15} \wedge d_a = 1 \wedge d_d = d_h \wedge d_h = 0 \wedge A \wedge D \wedge H \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{41} \rangle q_9$$

(c) $q_{15} \xrightarrow{\{D,H\},d_d=d_h\wedge d_h=0} q_3$ will result in the PDL sentence:

$$q_{15} \wedge d_d = d_h \wedge d_h = 0 \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{42} \rangle q_3$$

(d) $q_{15} \xrightarrow{\{A\},d_a=0} q_{19}$ will result in the PDL sentence:

$$q_{15} \wedge d_a = 0 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{43} \rangle q_{19}$$

(e) $q_{15} \xrightarrow{\{A\},d_a=1} q_{20}$will result in the PDL sentence:

$$q_{15} \wedge d_a = 1 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{44} \rangle q_{20}$$

17. state: q_{16}

(a) $q_{16} \xrightarrow{\{A,D,H\},d_a=0\wedge d_d=d_h\wedge d_h=0} q_{13}$ will result in the PDL sentence:

$$q_{16} \wedge d_a = 0 \wedge d_d = d_h \wedge d_h = 0 \wedge A \wedge D \wedge H \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{45} \rangle q_{13}$$

(b) $q_{16} \xrightarrow{\{A,D,H\},d_a=0\wedge d_d=d_h\wedge d_h=0} q_{14}$ will result in the PDL sentence:

$$q_{16} \wedge d_a = 0 \wedge d_d = d_h \wedge d_h = 0 \wedge A \wedge D \wedge H \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{46} \rangle q_{14}$$

(c) $q_{16} \xrightarrow{\{D,H\},d_d=d_h\wedge d_h=0} q_4$ will result in the PDL sentence:

$$q_{16} \wedge d_d = d_h \wedge d_h = 0 \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{47} \rangle q_4$$

(d) $q_{16} \xrightarrow{\{A\},d_a=0} q_{19}$ will result in the PDL sentence:

$$q_{16} \wedge d_a = 0 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{48} \rangle q_{21}$$

(e) $q_{16} \xrightarrow{\{A\},d_a=1} q_{20}$will result in the PDL sentence:

$$q_{16} \wedge d_a = 1 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{49} \rangle q_{22}$$

18. state: q_{17}

(a) $q_{17} \xrightarrow{\{A,D,H\},d_a=0 \wedge d_d=d_h \wedge d_h=1} q_8$ will result in the PDL sentence:

$$q_{17} \wedge d_a = 0 \wedge d_d = d_h \wedge d_h = 1 \wedge A \wedge D \wedge H \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{50} \rangle q_8$$

(b) $q_{17} \xrightarrow{\{A,D,H\},d_a=1 \wedge d_d=d_h \wedge d_h=1} q_9$ will result in the PDL sentence:

$$q_{17} \wedge d_a = 1 \wedge d_d = d_h \wedge d_h = 1 \wedge A \wedge D \wedge H \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{51} \rangle q_9$$

(c) $q_{17} \xrightarrow{\{D,H\},d_d=d_h \wedge d_h=1} q_3$ will result in the PDL sentence:

$$q_{17} \wedge d_d = d_h \wedge d_h = 1 \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{52} \rangle q_3$$

(d) $q_{17} \xrightarrow{\{A\},d_a=0} q_{23}$will result in the PDL sentence:

$$q_{17} \wedge d_a = 0 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{53} \rangle q_{23}$$

(e) $q_{17} \xrightarrow{\{A\},d_a=1} q_{24}$ will result in the PDL sentence:

$$q_{17} \wedge d_a = 1 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{54} \rangle q_{24}$$

19. state: q_{18}

(a) $q_{18} \xrightarrow{\{A,D,H\},d_a=0 \wedge d_d=d_h \wedge d_h=1} q_{13}$ will result in the PDL sentence:

$$q_{18} \wedge d_a = 0 \wedge d_d = d_h \wedge d_h = 1 \wedge A \wedge D \wedge H \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{55} \rangle q_{13}$$

(b) $q_{18} \xrightarrow{\{A,D,H\},d_a=1 \wedge d_d=d_h \wedge d_h=1} q_{14}$ will result in the PDL sentence:

$$q_{18} \wedge d_a = 1 \wedge d_d = d_h \wedge d_h = 1 \wedge A \wedge D \wedge H \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{56} \rangle q_{14}$$

(c) $q_{18} \xrightarrow{\{D,H\},d_d=d_h \wedge d_h=1} q_4$ will result in the PDL sentence:

$$q_{18} \wedge d_d = d_h \wedge d_h = 1 \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{57} \rangle q_4$$

(d) $q_{18} \xrightarrow{\{A\},d_a=0} q_{25}$ will result in the PDL sentence:

$$q_{18} \wedge d_a = 0 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{58} \rangle q_{25}$$

(e) $q_{18} \xrightarrow{\{A\},d_a=1} q_{26}$will result in the PDL sentence:

$$q_{18} \wedge d_a = 1 \wedge A \wedge \neg B \wedge \neg C \wedge \neg D \wedge \neg E \wedge \neg G \wedge \neg H \leftrightarrow \langle t_{59} \rangle q_{26}$$

20. state: q_{19}

(a) $q_{19} \xrightarrow{\{D,H\},d_h=0 \wedge d_d=d_h} q_8$will result in the PDL sentence:

$$q_{19} \wedge d_h = 0 \wedge d_d = d_h \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{60} \rangle q_8$$

21. state: q_{20}

(a) $q_{20} \xrightarrow{\{D,H\},d_h=0 \wedge d_d=d_h} q_9$ will result in the PDL sentence:

$$q_{20} \wedge d_h = 0 \wedge d_d = d_h \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{61} \rangle q_9$$

22. state: q_{21}

(a) $q_{21} \xrightarrow{\{D,H\},d_h=0 \wedge d_d=d_h} q_{13}$ will result in the PDL sentence:

$$q_{21} \wedge d_h = 0 \wedge d_d = d_h \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{62} \rangle q_{13}$$

23. state: q_{22}

(a) $q_{22} \xrightarrow{D,H,d_h=0 \wedge d_d=d_h} q_{14}$ will result in the PDL sentence:

$$q_{22} \wedge d_h = 0 \wedge d_d = d_h \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{63} \rangle q_{14}$$

24. state: q_{23}

(a) $q_{23} \xrightarrow{\{D,H\},d_h=1 \wedge d_d=d_h} q_8$ will result in the PDL sentence:

$$q_{23} \wedge d_h = 1 \wedge d_d = d_h \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{64} \rangle q_8$$

25. state: q_{24}

(a) $q_{24} \xrightarrow{\{D,H\}, d_h = 1 \wedge d_d = d_h} q_9$ will result in the PDL sentence:

$$q_{24} \wedge d_h = 1 \wedge d_d = d_h \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{65} \rangle q_9$$

26. state: q_{25}

(a) $q_{25} \xrightarrow{\{D,H\}, d_h = 1 \wedge d_d = d_h} q_{13}$ will result in the PDL sentence:

$$q_{25} \wedge d_h = 1 \wedge d_d = d_h \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{66} \rangle q_{13}$$

27. state: q_{26}

(a) $q_{26} \xrightarrow{\{D,H\}, d_h = 1 \wedge d_d = d_h} q_{14}$ will result in the PDL sentence:

$$q_{26} \wedge d_h = 1 \wedge d_d = d_h \wedge D \wedge H \wedge \neg A \wedge \neg B \wedge \neg C \wedge \neg E \wedge \neg G \leftrightarrow \langle t_{67} \rangle q_{14}$$

Each one of the original transitions of the automaton were transformed in a PDL sentence, following the proposed algorithm in Section 5.

6. Conclusions and further work

Formalizing and validating Reo circuits may bring some advantages regarding modelled systems in Reo. The proposed approach introduces a framework to reason about Reo circuits by means of Constraint Automata and Propositional Dynamic Logic, creating PDL sentences for each transition in the circuit's corresponding automaton. Therefore Reo circuits can be seen as PDL models, which verification can rely on known techniques for validating PDL models, such as model checking [9], or even using PDL as a query language for Reo models, in an approach similar to the one presented by Tuominen [16].

Future works may be directed in tailoring a specific logic to reason about Reo circuits. It aims to lead to a logic that captures specific Reo properties, enabling to check whether Reo models are free of deadlock or properties regarding how data flow in a model's nodes. The implementation of a Graphical User Interface that enables the user to generate PDL models from Reo connectors and specify properties to be validated is also a direction that this work may be directed to.

References

[1] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical structures in computer science*, 14(3):329–366, 2004.

[2] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science,* 14(3):329â€“366, 2004.

[3] F. Arbab. Coordination for component composition. *Electronic Notes in Theoretical Computer Science*, 160:15 – 40, 2006. Proceedings of the International Workshop on Formal Aspects of Component Software (FACS 2005).

[4] C. Baier, M. Sirjani, F. Arbab, and J. Rutten. Modeling component connectors in reo by constraint automata. *Science of computer programming,* 61(2):75–113, 2006.

[5] M. Benevides, B. Lopes, and E. H. Haeusler. Towards reasoning about petri nets: A propositional dynamic logic based approach. *Theoretical Computer Science,* 774(5):22–36, 2018.

[6] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of computer and system sciences,* 18(2):194–211, 1979.

[7] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* Foundations of Computing Series. MIT Press, 2000.

[8] N. Kokash, C. Krause, and E. De Vink. Reo+ mcrl2: A framework for model-checking dataflow in service compositions. *Formal Aspects of Computing*, 24(2):187–216, 2012.

[9] M. Lange. Model checking propositional dynamic logic with all extras. *Journal of Applied Logic,* 4(1):39–49, 2006.

[10] B. Lopes, M. Benevides, and E. H. Haeusler. Extending propositional dynamic logic for Petri nets. *Electronic Notes in Theoretical Computer Science*, 305(11):67–83, 2014.

[11] B. Lopes, M. Benevides, and E. H. Haeusler. Propositional dynamic logic for Petri nets. *Logic Journal of the IGPL,* 22(5), 2014.

[12] M. Marx. Xpath and modal logics of finite dag‘s. In M. Cialdea Mayer and F. Pirri, editors, *Automated Reasoning with Analytic Tableaux and Related Methods,* pages 150–164, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[13] M. Marx. Xpath with conditional axis relations. In E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm, and E. Ferrari, editors, *Advances in Database Technology - EDBT 2004*, pages 477–494, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[14] S. Navidpour and M. Izadi. Linear temporal logic of constraint automata. In *Advances in Computer Science and Engineering*, pages 972–975. Springer, 2008.

[15] C. Sinz, W. Kuchlin, and T. Lumpp. Towards a verification of the rule-based expert system of the ibm sa for os/390 automation manager. In *Quality Software, 2001. Proceedings. Second Asia-Pacific Conference on*, pages 367–374. IEEE, 2001.

[16] H. Tuominen. Elementary net systems and dynamic logic. In *European Workshop on Applications and Theory in Petri Nets*, pages 453–466. Springer, 1988.

[17] H. Tuominen. Elementary net systems and Dynamic Logic. In G. Rozenberg, editor, *Advances in Petri Nets 1989*, Lecture Notes in Computer Science, pages 453–466. Springer Berlin Heidelberg, 1990.

[18] F. Wolter and M. Zakharyaschev. Dynamic Description Logics. In *Proceedings of AiML'98*, pages 290–300. CSLI Publications, 2000.

[19] J. Klein, S. Klüppelholz, A. Stam, and C. Baier. Hierarchical modeling and formal verification. an industrial case study using reo and vereofy. In *International Workshop on Formal Methods for Industrial Critical Systems*, pages 228–243. Springer, 2011

[20] N. Kokash and F. Arbab. Formal design and verification of long-running transactions with extensible coordination tools. *IEEE Transactions on Services Computing*, 6(2):186–200, 2011.

[21] N. Kokash, C. Krause, and E. De Vink. Reo+ mcrl2: A framework for model-checking dataflow in service compositions. *Formal Aspects of Computing*, 24(2):187–216, 2012.-21

[22] Y. Li, X. Zhang, Y. Ji, and M. Sun. A formal framework capturing real-time and stochastic behavior in connectors. *Science of Computer Programming*, 2019. -31

[23] M. R. Mousavi, M. Sirjani, and F. Arbab. Formal semantics and analysis of component connectors in reo. *Electronic Notes in Theoretical Computer Science*, 154(1):83–99, 2006.- 32

[24] B. Pourvatan, M. Sirjani, H. Hojjat, and F. Arbab. Automated analysis of reo circuits using symbolic execution. *Electronic Notes in Theoretical Computer Science*, 255:137–158, 2009. -37

[25] S.-S. T. Jongmans and F. Arbab. Overview of thirty semantic formalisms for reo. *Scientific Annals of Computer Science*, 22(1), 2012.- 24

[26] F. Arbab, C. Baier, F. de Boer, and J. Rutten. Models and temporal logical specifications for timed component connectors. *Software & Systems Modeling*, 6(1):59–82, 2007. -5

[27] E. Ardeshir-Larijani, A. Farhadi, and F. Arbab. Simulation of hybrid reo connectors. In *2020 CSI/CPSSI International Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*, pages 1–10. IEEE, 2020.- 8

[28] Y. Li, X. Zhang, Y. Ji, and M. Sun. 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,* pages 287–304, 2017.

[29] M. Sun and Y. Li. 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*, pages 506–507. ACM, 2014.

[30] X. Zhang, W. Hong, Y. Li, and M. Sun. Reasoning about connectors in coq. In *International Workshop on Formal Aspects of Component Software*, pages 172–190. Springer, 2016.

[31] X. Zhang, W. Hong, Y. Li, and M. Sun. Reasoning about connectors using coq and z3. *Science of Computer Programming*, 170:27–44, 2019.