Regulating Synchronous Data Exchange to Meet Control Flow and Data Specifications

Ashwin Bhaskar
Chennai Mathematical Institute, India

M. Praveen
Chennai Mathematical Institute, India
CNRS IRL ReLaX, Chennai, India

Abstract

When multiple software components interact via method calls, we may want to ensure that the order of invoked methods and the arguments provided adhere to some specification. The classic problem associated with interface automata checks for the existence of a mediator whose intention is to act as a buffer in between method invocations so that invocations do not go unanswered. We extend the base model underlying interface automata, enabling them to exchange integer values - one automaton generates an integer value and outputs it by firing a generating transition and another automaton receives the value by synchronously firing a receiving transition. Transitions in the automata can have guards with linear order constraints on the exchanged values, influencing which methods can or can not be invoked later. So the generated values influence the sequences of invocations that are enabled. We specify desirable properties of the sequence of method calls and the arguments passed to them using an extension of Linear Temporal Logic (LTL). We consider the interoperability problem, which is to check if it is possible to generate integer values in such a way that all enabled sequences satisfy the given specification.

We show that the interoperability problem is undecidable in general, even when there are only two participating automata. We show decidability in the case where guards on generating transitions can only have equality constraints on the exchanged value (but receiving transitions can continue to have linear order constraints). We model this problem as a game between two players, one trying to generate integer values such that violating sequences are disabled while the other player tries to dig out violating sequences that are enabled. Interoperability is equivalent to the first player having a winning strategy. We solve this game via a finite abstraction, which results in a symbolic game. We then show that winning strategies for the symbolic game can be translated to winning strategies for the original game over integers.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Modal and temporal logics; Theory of computation → Verification by model checking

Keywords and phrases Distributed Systems, Interface Automata, Registers, Parity Games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2025.14

Funding *M. Praveen*: This author is partially supported by the Infosys foundation.

1 Introduction

There are many models for formalizing multiple software components interacting with each other. E.g. in [11], smart contracts are modeled using interface automata [5]. The classic question for interface automata is to check if there exists a mediator (that can buffer method calls among the participants) such that there are no unanswered method calls [5, 10]. In [1], smart contracts are modeled in the Behavior Interaction Priority (BIP) framework, which are finite state machines (FSMs) interacting via ports. The smart contracts in [1] are part of a supply chain for crude oil, and the goal is to model check properties such as "once the point-of-sale receives the oil batches from a vendor, the payment must be triggered". This is a typical temporal property saying one event must be followed by another.

FSMs in the BIP framework can exchange data in the form of arguments passed to method calls. We work with a model that also allows exchange of data (for example, an integer indicating the amount to be paid), but using synchronized transitions as in interface automata. In [1], control flow can be influenced by conditions such as "if the production at a vendor site is more than a specified limit, send an alert". We model these using guards on transitions, which comprise of linear order constraints on the data exchanged. The value that is exchanged thus influences which branches can be taken by the control flow. The interoperability problem is to check if the values can be generated in such a way that the control flow remains within a specified set. The set is specified by Constraint Linear Temporal Logic (CLTL), an extension of LTL that can check linear order constraints on data values. In the above supply chain example, this can check if there exists a value for the daily production limit such that vendor and point-of-sale work together ensuring that there is always enough inventory.

We benefit from game theoretic foundations of interface automata. To tackle the interoperability problem, we design a game between two players, SYSTEM and SPOILER. SYSTEM tries to choose values of the exchanged data (restricted to integers here) so that branches of control flow that can potentially violate the specification are disabled. SPOILER tries to pick out branches in the enabled part of control flow that can potentially violate the specification. If SYSTEM has a winning strategy, it provides integer values that will ensure that the specification is always met in all enabled branches of the control flow.

One difficulty that needs to be overcome is that branches in which the game can proceed depends on choices SYSTEM makes from an infinite domain (integers). We handle this by designing an abstraction that reduces the game to a finite one, while still maintaining enough information about existence of winning strategies. The high level idea of using abstraction to reduce infinite domains to finite ones is a known technique [7, 3, 2, 8]. But designing the abstraction must be done carefully so that peculiarities of the original model are correctly reflected in the resulting finite symbolic model. Our first contribution is coming up with a model that is expressive enough to specify interesting properties. An example is given in Section 4. Our second contribution is designing an abstraction [definition 19] such that the resulting finite symbolic model meets two goals. First, it is equivalent to the original model wrt the existence of winning strategies [Lemma 24]. Second, it can be solved by adapting known techniques [[3], Lemma 23].

Suppose the participants do not know each other's configurations and we need to check if they can still generate values based on the imperfect information they have. It turns out that if each participant sends integer values regularly, they can communicate their configuration information by encoding it in the integers they generate, thus communicating their configuration information to other participants and reducing it to the perfect information case. This trick will fail in other cases. It would be interesting to identify interesting fragments of the other cases where the interoperability problem makes sense.

Related work. In [10, 14], interface automata are extended with registers to handle data like we do here. But there, data values can be compared only for equality and its negation while we have linear order. Also, [10, 14] only consider specific safety properties; we allow properties expressible in CLTL. In [8], synthesis problem for specifications given as register automata over linearly ordered data domains (including $\mathbb{Z}$) are studied. They prove decidability for the existence of a winning strategy for one-sided games, where one of the players operates over a finite alphabet and the other manipulates data. The concrete game which we use to model the interoperability problem here is similar to such a one-sided game as even over

there, one of the players can only choose from a domain while the other can choose integer values. Also, we use techniques similar to theirs in order to deal with abstraction of the register values by tracking constraints between them and checking for the feasibility of such constraint sequences. However, synchronization between register automata is the central theme of our model and specification is based on a logic. In [8], register automata are used as specification mechanisms. Our techniques involve a combination of qualititative and quantitative specifications. Energy games [6, 4] provide a foundation for many such models.

2 Preliminaries

Let $\mathbb{Z}$ be the set of integers and $\mathbb{N}$ be the set of non-negative integers. We denote by $\lceil i \rceil_k$ the number i ceiled at k: $\lceil i \rceil_k = i$ if $i \leq k$ and $\lceil i \rceil_k = k$ otherwise. For integers n_1, n_2, we denote by $[n_1, n_2]$ the set $\{n \in \mathbb{Z} \mid n_1 \leq n \leq n_2\}$.

We recall here the definitions of constraint systems and Constraint LTL (CLTL) from [7] with some slight modifications. A constraint system $\mathcal{D}$ is of the form $(D, R_1, \ldots, R_n, \mathcal{I})$, where D is a non-empty set called the domain. Each R_i is a predicate symbol of arity a_i, with $\mathcal{I}(R_i) \subseteq D^{a_i}$ being its interpretation. For the sake of this paper, it suffices to describe CLTL formulas over constraint systems of the form $(D, <, =)$ where D is an infinite domain, $<$ is a total linear order over D and $=$ is the equality relation.

Let V be a finite set of variables, and A be a finite set of actions. A term is of the form $X^{-i}x$, where x is a variable and $i \geq 0$. A constraint c is of the form $t_i < t_j$ or $(t_i = t_j)$, where t_i, t_j are terms. The syntax of CLTL is given by the following grammar, where $a \in A$ and c is a constraint as defined above.

$$\phi ::= c \mid a \mid \neg\phi \mid \phi \vee \phi \mid X\phi \mid \phi U \phi$$

Let F_D denote the set of all mappings of the form $f : V \to D$. The semantics of CLTL is defined with respect to infinite sequences over $A \times F_D$ (also called concrete models in the following).

Given a concrete model $\alpha = (a_0, f_0)(a_1, f_1) \ldots \in (A \times F_D)^\omega$, the i^{th} position of α satisfies the atomic formula $a \in A$ (written as $\alpha, i \models a$) if $a = a_i$. Also, given $x, y \in V$ and $i_1, i_2 \in \mathbb{N}$, the i^{th} position of a concrete model α satisfies the constraint $X^{-i_1}x < X^{-i_2}y$ (resp. $X^{-i_1}x = X^{-i_2}y$) (written as $\sigma, i \models X^{-i_1}x < X^{-i_2}y$ (resp. $X^{-i_1}x = X^{-i_2}y$)) if and only if $i \geq i_1, i \geq i_2$ and $f_{i-i_1}(x) < f_{i-i_2}(y)$ (resp. $f_{i-i_1}(x) = f_{i-i_2}(y)$) hold. The semantics is extended to the rest of the syntax similar to the usual propositional LTL. We use the standard abbreviations $F\phi$ (resp. $G\phi$) to mean that ϕ is true at some position (resp. all positions) in the future. The X-length of a term $X^{-i}x$ is i. We say that a formula is of X-length k if it uses terms of X-length at most k. The formula $G(Fa \wedge X(X^{-1}x < y))$ will be true in the first position of a concrete model if, in all positions, the value of x is less than the value of y in the next position and if the action a holds infinitely often in the concrete model. In the rest of this paper when we refer to CLTL formulas, we shall restrict ourselves to CLTL formulas over the constraint system $(\mathbb{Z}, <, =)$ using just one variable d. The models for such formulas with just one variable can simply be thought of as sequences of the form $(a_0, v_0)(a_1, v_1) \ldots$ where $v_i \in \mathbb{Z}$.

3 Interface Automata with Data

We work with a model based on interface automata [5, 10]. We extend it so that integer values can be exchanged and stored.

▶ **Definition 1** (Finite State Register Machines (FSRMs)). *A finite state register machine* $\mathcal{M} = \langle B, q^0, R, T^E, T^H \rangle$ *where:*

- B *is a finite set of states,* $q^0 \in B$ *is the initial state and* R *is a finite set of registers,*
- $T^E \subseteq B \times \text{Test} \times \text{Asgn} \times B$ *is a set of external transitions, where* Asgn *is the power set of* R *and* Test *is the set of all formulas (guards) of the form* $g := r^i \leq r^j | r^i \leq d | g \wedge g | \neg g$ *where* $r^i, r^j \in R$ *and* d *is a formal parameter.*
- $T^H \subseteq B \times B$ *is the set of hidden transitions.*

We write $B_{\mathcal{P}}, R_{\mathcal{P}}, T^E_{\mathcal{P}}, T^H_{\mathcal{P}}$ to refer to the set of states, registers, external and hidden transitions of a particular FSRM $\mathcal{P}$. An R-valuation is a mapping from R to $\mathbb{Z}$. Given an R-valuation σ, an integer value v and a guard g, $\sigma, v \models g$ denotes that the formula g with each occurrence of $r \in R$ replaced by $\sigma(r)$ and each occurrence of d replaced by v, evaluates to true.

▶ **Definition 2** (Semantics of Finite State Register Machines). *The semantics of a finite state register machine* $\mathcal{M}$ *is a transition system defined as follows.*

- $C = B \times \mathbb{Z}^R$ *is the set of configurations,*
- $c^0 = (q^0, \{r \mapsto 0 \mid r \in R\})$ *is the initial configuration,*
- $(q, \sigma) \xrightarrow{(t,v)} (q', \sigma')$ *iff* $t = (q, g, u, q') \in T^E$ *such that* $\sigma, v \models g$ *and for each* $r \in R$, $\sigma'(r) = v$ *if* $r \in u$ *and* $\sigma'(r) = \sigma(r)$ *otherwise.*
- $(q, \sigma) \xrightarrow{t,*} (q', \sigma)$ *iff* $t = (q, q') \in T^H$.

In words, an external transition $t = (q, g, u, q')$ goes from configuration (q, σ) to (q', σ') storing the integer value v in registers belonging to u, provided t is enabled in σ: σ and the new integer value v should satisfy the guard g. Hidden transitions change states without changing register valuations. We refer to the pair (t, v) or $(t, *)$ as a *step* s.

▶ **Definition 3** (Traces of Finite State Register Machines). *Given a finite state register machine* $\mathcal{M}$*, we define a trace* τ *of* $\mathcal{M}$ *to be a finite or infinite sequence of alternating configurations and steps:* $c_0 s_0 \cdots c_{m-1} s_{m-1} c_m \cdots$ *where* $m \in \mathbb{N}$, $c_0 = c^0$, $c_i \in C$, $c_i \xrightarrow{s_i} c_{i+1}$ *and* s_i *is a step for all* $i \geq 0$*. We denote the set of all traces of* $\mathcal{M}$ *as* Traces.

A set of FSRMs can be composed. An output transition of one FSRM can synchronize with an input transition of another FSRM. To control which transitions can synchronize, we label them with a letter from a finite set of *actions*, along with one of the symbols $\{!, ?\}$. Transitions labeled with ! (resp. ?) are intended to be output (input) transitions. Two transitions can synchronize if they are labelled with the same action, one of them is additionally labeled with ! and the other one is additionally labeled with ?.

▶ **Definition 4** (Interface Register Transition System). *An interface register transition system* $\mathcal{S} = \langle \overline{\mathcal{P}}, A, \text{ac} \rangle$ *where* $\overline{\mathcal{P}}$ *is a tuple of finite state register machines,* A *is a finite set of actions and* $\text{ac} : T^E_{\overline{\mathcal{P}}} \to \{!, ?\} \times A$ *is a labelling function. The notation* $T^E_{\overline{\mathcal{P}}}$ *stands for the union of the sets of external transitions of all the FSRMs in* $\overline{\mathcal{P}}$*.*

▶ **Definition 5** (Semantics of Interface Register Transition System). *The semantics of an interface register transition system* $\mathcal{S}$ *is a transition system defined as follows:*

1. *A configuration is a function* $c : \overline{\mathcal{P}} \to \cup_{\mathcal{P} \in \overline{\mathcal{P}}} B_{\mathcal{P}} \times \mathbb{Z}^{R_{\mathcal{P}}}$ *such that* $c(\mathcal{P}) \in B_{\mathcal{P}} \times \mathbb{Z}^{R_{\mathcal{P}}}$ *for all* $\mathcal{P} \in \overline{\mathcal{P}}$*.*
2. $c^0 = \{\mathcal{P} \mapsto (q^0_{\mathcal{P}}, \{r \mapsto 0 \mid r \in R_{\mathcal{P}}\}) \mid \mathcal{P} \in \overline{\mathcal{P}}\}$ *is the initial configuration mapping all FSRMs to their initial states and all registers to* 0*.*

3. $c \xrightarrow{(a,t_o,t_i,v)}_{\mathcal{S}} c'$ *iff the following conditions are met:* $t_o \in T^E_{\mathcal{P}_o}$, $t_i \in T^E_{\mathcal{P}_i}$, $\mathcal{P}_o \neq \mathcal{P}_i$, $ac(t_o) = (!, a)$, $ac(t_i) = (?, a)$, $v \in \mathbb{Z}$, $(q_i, \sigma_i) \xrightarrow{t_i,v} (q'_i, \sigma'_i)$, $(q_o, \sigma_o) \xrightarrow{t_o,v} (q'_o, \sigma'_o)$ *and* c' *is obtained from* c *by changing* $c(\mathcal{P}_i)$ *from* (q_i, σ_i) *to* (q'_i, σ'_i) *and changing* $c(\mathcal{P}_o)$ *from* (q_o, σ_o) *to* (q'_o, σ'_o).
4. $c \xrightarrow{(a,t_o,*,v)}_{\mathcal{S}} c'$ *iff the following conditions are met:* $t_o \in T^E_{\mathcal{P}_o}$, $ac(t_o) = (!, a)$, $v \in \mathbb{Z}$, $(q_o, \sigma_o) \xrightarrow{t_o,v} (q'_o, \sigma'_o)$ *and* c' *is obtained from* c *by changing* $c(\mathcal{P}_o)$ *from* (q_o, σ_o) *to* (q'_o, σ'_o). *Additionally, for any transition* t_i *with* $ac(t_i) = (?, a)$, t_i *can not be fired in the configuration* c *with the value* v.
5. $c \xrightarrow{*,t,*,*} c'$ *iff the following conditions are met:* $t \in T^H_{\mathcal{P}}$, $(q, \sigma) \xrightarrow{t,*} (q', \sigma)$ *and* c' *is obtained from* c *by changing* $c(\mathcal{P})$ *from* (q, σ) *to* (q', σ).

In the transition mentioned in point 3 above, output transition t_o and input transition t_i from different FSRMs synchronize by exchanging the integer value v. In the transition of point 4, only an output transition from an FSRM fires and we think of the environment as receiving the value, provided none of the transitions in $\mathcal{S}$ can fire to receive the value v. In the transition in point 5, a hidden transition in FSRM $\mathcal{P}$ fires. We refer to tuples $(a, t_o, t_i, v), (a, t_o, *, v)$ and $(*, t, *, *)$ as steps.

▶ **Definition 6** (Traces of Interface Register Transition System). *Given an interface register transition system* $\mathcal{S}$*, we define a trace (or a finite trace)* τ *of* $\mathcal{S}$ *to be a finite alternating sequence of configurations and steps:* $c_0 s_0 \dots c_{m-1} s_{m-1} c_m$ *where* $c_0 = c^0$, $m \in \mathbb{N}$, $c_i \in C$, $c_i \xrightarrow{s_i}_{\mathcal{S}} c_{i+1}$ *and* s_i *is a step for all* $0 \leq i \leq m$*. We denote the set of all finite traces of* $\mathcal{S}$ *as* Traces*. An infinite trace* $\overline{\tau}$ *of* $\mathcal{S}$ *is similarly defined to be an infinite alternating sequence of configurations and steps. We denote the set of all infinite traces of* $\mathcal{S}$ *as* $\overline{\text{Traces}}$.

For a step (a, t_o, t_i, v) or $(a, t_o, *, v)$, its projection to actions and values is $\pi_{av}((a, t_o, t_i, v)) = \pi_{av}((a, t_o, *, v)) = (a, v)$. For a step $(*, t, *, *)$ with a hidden transition t, $\pi_{av}((*, t, *, *)) = \epsilon$, the empty sequence. We extend π_{av} to traces as $\pi_{av}(c_0 s_0 c_1 s_1 \cdots) = \pi_{av}(s_0)\pi_{av}(s_1) \cdots$. We say that a trace $\overline{\tau}$ satisfies a CLTL formula ϕ (denoted as $\overline{\tau} \models \phi$) if there are infinitely many external transitions fired along $\overline{\tau}$ and $\pi_{av}(\overline{\tau}) \models \phi$.

Two traces that differ only in hidden transitions can not be distinguished by CLTL formulas. We formalize this next.

▶ **Definition 7** (Hidden transitions invariance). *Suppose a finite trace* τ *is of the form* $\tau_1 \cdot c \cdot (*, t, *, *) \cdot \tau_2$*. Bypassing the step* $(*, t, *, *)$ *based on the hidden transition* t *will result in the sequence* $\tau_1 \cdot \tau_2$*. Let* τ^* *be obtained from* τ *by bypassing all hidden transitions occurring in* τ*. We say two traces* τ_1, τ_2 *are equivalent wrt hidden transitions (written as* $\tau_1 \sim_I \tau_2$*) if* $\tau_1^* = \tau_2^*$.

A step (a, t_o, t_i, v) can only be taken if the current configuration along with the new value v satisfies the guards in the transitions t_o, t_i. Choosing certain values can thus help in avoiding traces that violate a given CLTL specification. We denote by $T^O_{\mathcal{P}}$ the set of transitions $T^O_{\mathcal{P}} = \{t \in T^E_{\mathcal{P}} \mid ac(t) \in \{!\} \times A\}$.

▶ **Definition 8** (Regulator). *Given an interface register transition system* $\mathcal{S}$*, a regulator is a function* $\text{reg} : Traces_{\mathcal{S}}/\sim_I \times T^O_{\mathcal{P}} \to \mathbb{Z}$.

An infinite trace $\overline{\tau}$ conforms to a regulator reg if the following is true: for every prefix of the form $\tau(a, t_o, t_i, v)c'$ or $\tau(a, t_o, *, v)c'$, $v = \text{reg}([\tau]_{\sim_I}, t_o)$. We say that an interface register transition system $\mathcal{S}$ is interoperable satisfying a CLTL formula ϕ if there is a regulator reg such that all traces $\overline{\tau}$ that conform to reg satisfy ϕ.

In order to handle maximal finite traces, we add gadgets to the interface register transition system. For every action a, we add a sink state s_a. For an existing state q, suppose $g_1, \ldots, g_r$ are the guards on the input transitions from q labelled with a. We add an input transition from q to s_a labelled with the action a and the guard $\neg(g_1 \vee \ldots \vee g_r)$. This new transition is enabled if and only if none of the existing input transitions labelled by a are enabled. On s_a, we have a self-loop with an output transition labelled with a new action a'. On every existing state in all the FSRMs, we add self-loops with an input transition labelled with a'. This way, a maximal finite play is artificially extended to an infinite one by padding $(a')^\omega$. This can be easily identified in the CLTL specification and handled accordingly (like disallowing such actions a' as needed for the undecidability proof in appendix A).

In general, the interoperability problem for interface register transition system is undecidable. Given a 2-counter machine, we construct an interface register transition system with just two FSRMs such that the runs of the 2-counter machine are simulated by the traces of $\mathcal{S}$. It is an adaptation of similar undecidability proofs for games involving numbers, in which one player simulates a two counter machine and the other player catches mistakes if any during the simulation. For example, if the player simulating an incrementing transition increments by more than 1, it can be caught by the other player by checking that there is a value between the previous and incremented value. The details of the proof of undecidability can be found in appendix A.

4 Modeling Example

In this section, we provide an example of modeling an e-commerce system with the aim of checking whether a client and a server can be made to work with each other by regulating the interaction appropriately. The example is for most parts copied from a similar example given in [10]; we add a feature to it motivating the introduction of linear order constraints in the guards of the FSRMs, which are absent in [10]. For modeling this example, we assume Interface Register Transition Systems can exchange multiple integer values in one transition. Our results can be easily extended to handle this extension.

The example models a typical online store; one FSRM for the client and one for the server. Suppose the client starts by sending a *StartOrder* message containing its *id* and it expects to receive an id of a new order. A schematic of the FSRM modeling the client is shown below in Figure 1.

In the figure, an edge label such as $!StartOrder(ClientId)$ means that it is a transition labeled with the action *StartOrder* and the symbol ! (to indicate that it is a output action) and sends an integer value stored in the register named *ClientId*. The client then orders a number of items via the *AddToOrder* action, at the end of which it provides the payment information via the *PlaceOrder* action. Some edge labels in the figure are a quite long, so we have used shorthands such as $stmt_1$. These shorthands are expanded as below.

$$
\begin{aligned}
stmt_i &= AddToOrder(OrderId, ItemId_i, Qty_i) \\
stmt &= PlaceOrder(OrderId, CCNo) \\
rjct_i &= OutOfRange(ItemId_i, Min, Max); Qty_i < Min \vee Max < Qty_i \\
cnfm_i &= Confirmed(OrderId, Item_i, Qty_i)
\end{aligned}
$$

$$stmt^i_a = Process_Item(OrderId, ItemId)$$
$$stmt^i_b = SetQty(OrderId, ItemId, Qty_i)$$
$$stmt^i_c = ConfirmItem(OrderId, ItemId, Qty_i)$$
$$stmt^i_d = OutOfRange(OrderId, ItemId, Qty_i, Min, Max)$$
$$stmt_e = CloseOrder(OrderId, CCNo)$$

Then the client expects all items together with the quantities to be confirmed by the customer service. The client also accepts rebuttals from the customer service in case the number of items ordered is outside the minimum and maximum range allowed, via the *OutOfRange* message. This is a feature we have added here beyond what is described in [10], in order to demonstrate the utility of linear order constraints. The edge label $rjct_i$ above implements this feature by using a guard to test that the quantity is indeed out of range. In practice, e-commerce systems would allow the client to revise the order, but for the sake of simplicity here, we just move ahead by dropping those items that are out of range. It blocks in case that it does not receive the right confirmation for the remaining items. Then it announces that it is ready to close the order and expects to receive the result of the payment transaction.

The client service expects to receive the client's id, then it sends a confirmation and sends an id of a new order, together with a client verification. It is then prepared to repeat a loop in which it 1) receives an order of an item, 2) receives a quantity, 3) confirms the quantity or rejects as out of range and sends minimum and maximum quantity allowed. After that, it receives payment information, arranges payment via a third party service (invisible to the client and not modeled here), and sends the result of the payment transaction.

The client and customer service are not directly interoperable, since the client sends all items before accepting responses but the customer service sends responses for each item before accepting the next item. They can however still interoperate via a mediator that buffers the item orders and communicates with the two parties as expected. As done in [10], we restrict ourselves to the case where the number of items per order is bounded by a constant n, which becomes a parameter of the interoperability problem. The specification will say that every item is either acknowledged or out of range and that the order is closed and the result of the payment transaction is provided to the client. This can be easily specified in CLTL.

5 Modelling interoperability as a game

In the simulation of the two counter machine briefly mentioned at the end of Section 4 (details of which can be found in appendix A), we check that counter increments and decrements are done correctly using guards on output transitions, which ensure that the generated value is strictly between two previously generated values. It turns out that if we restrict guards on output transitions to prevent such checks, interoperability problem is decidable. Specifically, let guards on output transitions be those of the form $g := r^i \leq r^j | r^i = d | g \wedge g | \neg g$. From now on, we shall call this the restricted syntax. The new value being generated (represented symbolically by d in guards) can only be tested for equality ($r_i = d$, as opposed to $r_i \leq d$ in general) with register values. In other words, the restricted guards can check if the generated value is actually there in one of the registers, which is a natural test one may want to do. With guards on output transitions obeying this restricted syntax, the interoperability problem is decidable. As a first step towards proving decidability, we informally describe a game formulation of the interoperability problem.

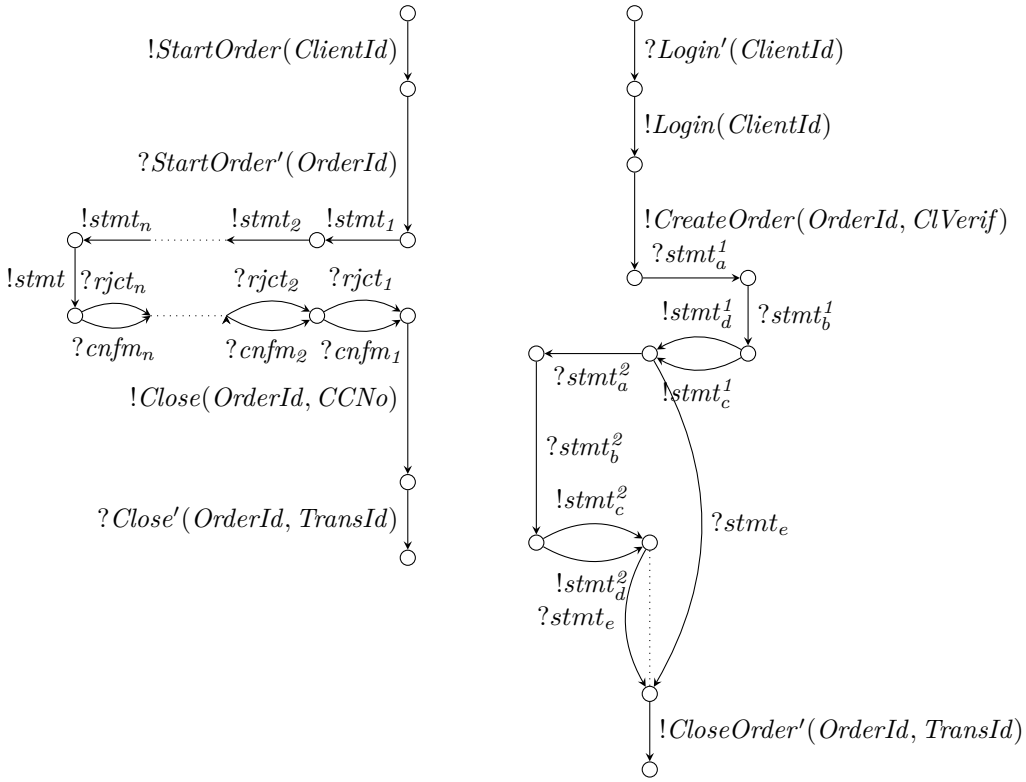

Figure 1 Modeling an e-commerce system. On the left is the client FSRM and on the right, customer service FSRM.

The game is played between SYSTEM and SPOILER, on the transition system which is the semantics of the given interface register transition system. SPOILER's goal is to pick out a trace $\overline{\tau}$ that violates the given CLTL specification. SYSTEM tries to prevent this by choosing values of the generated integers such that some transitions are disabled, thus restricting the choices available to SPOILER in the next move. The game begins at the initial configuration and SPOILER chooses an output transition t_o labeled with some action a enabled at the current configuration. SYSTEM chooses a value v based on the history of the play so far. SPOILER chooses a transition t_i enabled at some FSRM. The game moves to the updated configuration as per semantics. The game continues with the next round as before. SYSTEM wins the play if the infinite trace generated by the play satisfies the CLTL specification. The existence of a winning strategy for SYSTEM in this game is equivalent to the existence of a regulator.

To formally prove the above equivalence, we first formalize the game itself. A round ends with SPOILER choosing an input transition that is enabled at the current configuration. The next round begins wth SPOILER choosing an output transition that is enabled at the current configuration. We will combine these two moves of the spoiler and make it into a position of another game, which we call the *concrete game*. We denote by $T^I_{\overline{\mathcal{P}}}$ the set of transitions $T^I_{\overline{\mathcal{P}}} = \{t \in T^E_{\overline{\mathcal{P}}} \mid ac(t) \in \{?\} \times A\}$.

▶ **Definition 9** (Concrete game). *Given an interface register transition system $\mathcal{S}$, we associate with it a* concrete *game. The initial positions are of the form (c_0, t_o), where c_0 is any configuration that can be reached from the initial configuration c^0 with a finite sequence of hidden transitions and t_o is any output transition enabled at c_0. The set of other positions is $(T^I_{\overline{\mathcal{P}}} \cup \{*\}) \times C \times T^O_{\overline{\mathcal{P}}}$. The set of* SYSTEM *moves is $\mathbb{Z}$.*

The concrete game starts at an initial position (c_0, t_o). From any position (c_0, t_o), (t_i, c, t_o) or $(*, c, t_o)$, SYSTEM makes a move by choosing a value $v \in \mathbb{Z}$. SPOILER can respond by moving to any position (t_i', c_2, t_o') (resp. $(*, c_2, t_o')$) provided the following conditions are satisfied:

- Output transition t_o and input transition t_i' are labeled with the same action, say a (resp. t_o is labeled with a and there is no input transition enabled at the configuration c to receive the value v),
- $c \xrightarrow{a,t_o,t_i',v}_{\mathcal{S}} c_1$ (resp. $c \xrightarrow{a,t_o,*,v}_{\mathcal{S}} c_1$), c_2 is reachable from c_1 by a sequence of hidden transitions and t_o' is an output transition enabled in c_2.

The game then continues as before from the new position. A play is an infinite sequence of alternating positions and moves: $p_0 v_0 p_1 v_1 p_2 \cdots$ such that successive positions satisfy the conditions above. Histories are finite prefixes of plays, ending at positions. With every play π, we can naturally associate an infinite sequence of actions and values $\pi_{av}(\pi)$. SYSTEM wins a play π if $\pi_{av}(\pi)$ satisfies the CLTL formula ϕ specified in the interoperability problem. Let H denote the set of all histories.

▶ **Definition 10** (Strategies). *A SYSTEM strategy is a function $f : H \to \mathbb{Z}$.*

A play $p_0 v_0 p_1 v_1 p_2 \cdots$ conforms to the strategy f if for all $i \geq 0$, $v_i = f(p_0 v_0 \cdots p_i)$. We say that f is a winning strategy for SYSTEM if all plays conforming to it are won by SYSTEM.

▶ **Proposition 11** (Interoperability and concrete game). *Let $\mathcal{S}$ be an interface register transition system such that guards on all the output transitions obey the restricted syntax and let ϕ be a CLTL formula. $\mathcal{S}$ is interoperable satisfying ϕ if and only if SYSTEM has a winning strategy in the corresponding concrete game.*

Proof. A regulator is a function $\text{reg} : Traces_{\mathcal{S}}/\sim_I \times T^O_{\mathcal{P}} \to \mathbb{Z}$. A strategy is a function $f : H \to \mathbb{Z}$. The result follows from the observation that there is a natural one-to-one correspondence between $Traces_{\mathcal{S}}/\sim_I \times T^O_{\mathcal{P}}$ and H. ◀

6 Symbolic interoperability game

In this section we prove that the interoperability problem for CLTL specifications is decidable when the guards on output transitions obey the restricted syntax.

We shall prove this by checking if SYSTEM has a winning strategy in the associated concrete game. This is in turn checked by designing a finite parity game that is equivalent to the concrete game with respect to SYSTEM having a winning strategy. We use positional determinacy of finite parity games to show this equivalence. In the rest of this section we elaborate on this.

In the concrete game, data exchanged between FSRMs come from $\mathbb{Z}$, an infinite set. We will have to simulate this in a game that can only use letters from a finite alphabet, for which we shall construct a symbolic abstraction.

We introduce the notion of partial frames and frames. We use frames (resp. partial frames) to capture the total pre-order (resp. not necessarily total pre-order) induced by the configurations and the data values along s consecutive positions in the concrete game. While a partial s-frame captures the pre-order induced by the configurations and data values appearing in the last s rounds of the concrete game upto the latest position of the SPOILER, an s-frame extends this to a total pre-order which includes the integer value generated by SYSTEM in its successive move as well.

The models of CLTL are infinite sequences over infinite alphabets. Frames, introduced in [7], abstract them to finite alphabets. Conceptually, frames and symbolic models as we will define here are almost the same as introduced in [7], where the authors used these notions to

solve the satisfiability problem for CLTL. Also, in [3], the authors use the notion of frames and partial frames in CLTL games to solve the single-sided realizability problem and the notions we use here are again, quite similar.

Given an interface register transition system $\mathcal{S} = \langle \overline{\mathcal{P}}, A, \mathrm{ac} \rangle$ and a CLTL formula ϕ over A and a single variable d, we first define the notion of $\mathcal{S}$-variables and $\mathcal{S}$-terms.

The set of $\mathcal{S}$-variables $V_{\mathcal{S}} = \bigcup_{\mathcal{P} \in \overline{\mathcal{P}}} R_{\mathcal{P}} \cup \{d\}$. An $\mathcal{S}$-term is of the form $X^{-i}x$, where $x \in V_{\mathcal{S}}$ and $i \geq 0$. For $k \geq 0$, we denote by $T_{\mathcal{S}}[k]$ the set of all $\mathcal{S}$-terms of the form $X^{-i}x$, where $i \in [0, k]$.

▶ **Definition 12** (Frames, [7], [3, Definition 3]). *Given a number $s \geq 1$, an s-frame f is a pair $(a_f, \leq_f)$ where $a_f \in A$ and $\leq_f$ is a total pre-order [1] on the set of terms $T_{\mathcal{S}}[s-1]$.*

In the notation s-frame, s is intended to denote the size of the frame – the number of successive positions about which information is captured. The current position and the previous $s-1$ positions are considered, for which the terms in $T_{\mathcal{S}}[s-1]$ are needed. We denote by $<_f$ and $\equiv_f$ the strict order and equivalence relation induced by $\leq_f$: $x <_f y$ iff $x \leq_f y$ and $y \not\leq_f x$ and $x \equiv_f y$ iff $x \leq_f y$ and $y \leq_f x$.

We will deal with symbolic models that constitute sequences of frames. An s-frame will capture information about the first s positions of a model.

An s-frame meant for positions $i-s$ to $i-1$ may be followed by another s-frame meant for positions $i-s+1$ to i. The two frames must be consistent about the positions $i-s+1$ to $i-1$ that they share. The following definition formalizes this requirement.

▶ **Definition 13** (One-step compatibility, [7], [3, Definition 4]). *For $s \geq 1$, an s-frame f and an $(s+1)$-frame g, the pair (f, g) is one-step compatible if for all terms $t_1, t_2 \in T_{\mathcal{S}}[s-1] \setminus V_{\mathcal{S}}$, $t_1 \leq_f t_2$ iff $t_1 \leq_g t_2$. For $s \geq 2$ and s-frames f, g, the pair (f, g) is one-step compatible if for all terms $t_1, t_2 \in T_{\mathcal{S}}[s-2] \setminus V_{\mathcal{S}}$, $t_1 \leq_f t_2$ iff $X^{-1}t_1 \leq_g X^{-1}t_2$*

Fix a number $k \geq 0$ and consider formulas of X-length k. A symbolic model is a sequence ρ of frames such that for all $i \geq 0$, $\rho(i)$ is an $\lceil i+1 \rceil_{k+1}$-frame and $(\rho(i), \rho(i+1))$ is one-step compatible. CLTL formulas can be interpreted on symbolic models, using symbolic semantics $\models_s$ as explained next. To check if the i^{th} position of ρ symbolically satisfies the atomic formula a, we check whether $\rho(i)$ is of the form $(a, \leq)$ for some pre-order $\leq$, and to check if the i^{th} position of ρ symbolically satisfies the atomic constraint $t_1 < t_2$ (where t_1, t_2 are terms), we check whether $t_1 < t_2$ holds according to the total pre-order $\leq_{\rho(i)}$ in the i^{th} frame $\rho(i)$. In formal notation, this is written as $\rho, i \models_s t_1 < t_2$ if $t_1 <_{\rho(i)} t_2$ holds. The symbolic satisfaction relation $\models_s$ is extended to all CLTL formulas of X-length k by induction on the structure of the formula, as done for propositional LTL. To check whether $\rho, i \models_s t_1 < t_2$ in this symbolic semantics, we only need to check $\rho(i)$, the i^{th} frame in ρ, unlike the CLTL semantics, where we may need to check other positions also. In this sense, the symbolic semantics lets us treat CLTL formulas as if they were formulas in propositional LTL and employ techniques that have been developed for propositional LTL. But to complete that task, we need a way to go back and forth between symbolic and concrete models.

Given a concrete model α, we associate with it a symbolic model $\mu(\alpha)$ as follows. Imagine we are looking at the concrete model through a narrow aperture that only allows us to view $k+1$ positions of the concrete model, and we can slide the aperture to view different portions. The i^{th} frame of $\mu(\alpha)$ will capture information about the portion of the concrete model visible when the right tip of the aperture is at position i of the concrete model (so the left tip

[1] a reflexive and transitive relation such that for all x, y, either $x \leq_f y$ or $y \leq_f x$

will be at $i - \lceil i \rceil_k$). Formally, the total pre-order of the i^{th} frame is the one induced by the valuations along the positions $i - \lceil i \rceil_k$ to i of the concrete model. The action occurring in the first component of each position of the symbolic model $\mu(\alpha)$ will be the same as the action appearing in the first component of the corresponding position of the concrete model α.

▶ **Lemma 14** ([7, Lemma 3.1], [3, Lemma 5]). *Let ϕ be a CLTL formula of X-length k. Let α be a concrete model and let $\rho = \mu(\alpha)$. Then $\alpha, 0 \models \phi$ iff $\rho, 0 \models_s \phi$.*

The symbolic model ρ induces an order $\leq_\rho$ on $\mathbb{N} \times V_{\mathcal{S}}$ as follows. Consider $i \in \mathbb{N}$ such that $\rho(i)$ contains an s-frame for some $s \geq 1$ and let $j \in \mathbb{N}$ be such that $j \leq i$ and $i - j \leq (s-1)$. We say $(i, x) \preceq_\rho (j, y)$ (resp. $(j, y) \preceq_\rho (i, x)$) if $x \leq_{\rho(i)} X^{j-i} y$ (resp. $X^{j-i} y \leq_{\rho(i)} x$). In other words, for the variables and positions captured in the frame $\rho(i)$, $\preceq_\rho$ is same as the total pre-order $\leq_{\rho(i)}$ over that frame. Now we define $\leq_\rho$ to be the transitive closure of $\preceq_\rho$ and $<_\rho$ to be the transitive closure of $\prec_\rho$. $\leq_\rho$ is now a partial order over $\mathbb{N} \times V_{\mathcal{S}}$.

For every concrete model, there is an associated symbolic model, but the converse is not true. We say that a symbolic model ρ admits a concrete model if there exists a concrete model α such that $\rho = \mu(\alpha)$. In particular, it is easy to see that if there are infinite chains of the form that we shall define below in a symbolic model then it does not admit a concrete model.

▶ **Definition 15** (Infinite chains, [7, Condition $C_{\mathbb{Z}}$]). *A symbolic model ρ is said to have infinite chains if there exist two infinite sequences $i_1 \leq i_2 \leq \cdots$ and $j_1 \leq j_2 \leq \cdots$ and two variables $x, y \in V_{\mathcal{S}}$ satisfying one of the following conditions:*

$$\begin{array}{ccccccccc} (i_1,x) & <_\rho & (i_2,x) & <_\rho & (i_3,x) <_\rho \cdots & & (i_1,x) & >_\rho & (i_2,x) & >_\rho & (i_3,x) >_\rho \cdots \\ \downarrow\wedge & & \downarrow\wedge & & \downarrow\wedge & \text{or} & \vee\downarrow^{\rho} & & \vee\downarrow^{\rho} & & \vee\downarrow^{\rho} \\ (j_1,y) & \geq_\rho & (j_2,y) & \geq_\rho & (j_3,y) \geq_\rho \cdots & & (j_1,y) & \leq_\rho & (j_2,y) & \leq_\rho & (j_3,y) \leq_\rho \cdots \end{array}$$

If a symbolic model ρ has an infinite chain as above, the integer values associated with $(i_2, x), (i_3, x) \ldots$ will all be distinct from each other and will be inside the interval from (i_1, x) to (j_1, y), which is impossible. Hence, absence of infinite chains is a necessary condition for a symbolic model to admit concrete models. We shall see later that this is also a sufficient condition for a subset of symbolic models

We next define partial frames and compatibility between a partial frame and a frame.

▶ **Definition 16** (Partial frames and compatibility). *For $s \geq 1$, a partial s-frame pf is a pair $(a_{pf}, \leq_{pf})$ where $a_{pf} \in A$ and $\leq_{pf}$ is a total pre-order on the set of terms $T_{\mathcal{S}}[s-1] \setminus \{d\}$. For $s \geq 0$, an s-frame f and a partial $(s+1)$-frame pf, the pair (f, pf) is one-step compatible if for all $t_1, t_2 \in T_{\mathcal{S}}[s-1] \setminus V_{\mathcal{S}}$, $t_1 \leq_f t_2$ iff $t_1 \leq_{pf} t_2$. For $s \geq 2$, an s-frame $f = (a_f, \leq_f)$ and a partial s-frame $pf = (a_{pf}, \leq_{pf})$, the pair (f, pf) is one-step compatible if for all $t_1, t_2 \in T_{\mathcal{S}}[s-2] \setminus V_{\mathcal{S}}$, $X^{-1}t_1 \leq_{pf} X^{-1}t_2$ iff $t_1 \leq_f t_2$, and the pair (pf, f) is one-step compatible if $a_f = a_{pf}$ and for all $t_1, t_2 \in T_{\mathcal{S}}[s-1] \setminus \{d\}, t_1 \leq_{pf} t_2$ iff $t_1 \leq_f t_2$.*

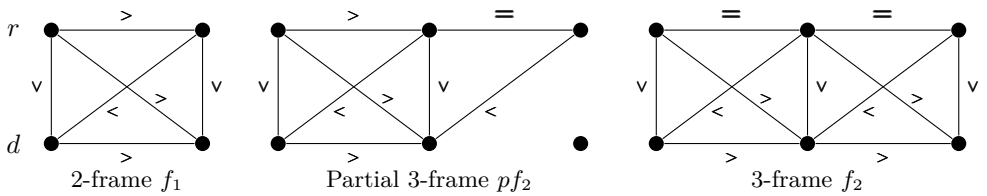

Figure 2 Frames, partial frames and one-step compatibility.

Figure 2 illustrates s-frames, partial s-frames and one-step compatibility graphically. Observe here that the pairs (f_1, pf_2) and (pf_2, f_2) are both one-step compatible. Note that the figure does not indicate some of the edges in the 3-frame and partial 3-frame which can be inferred using transitivity.

We now begin defining the symbolic game. We first define the notion of a symbolic configuration. A pair $(\overline{q_P}, f)$ where f is any $\lceil s \rceil_{k+1}$-frame for some $s \geq 1$, is called a symbolic configuration (denoted by sc). We call $(\overline{q_P}, pf)$ a partial symbolic configuration (denoted by psc) where pf is any partial $\lceil s \rceil_{k+1}$ -frame for some $s \geq 1$. Given a transition $t = (q, g, u, q')$, a tuple of states $\overline{q_P}$ and a $\lceil s \rceil_{k+1}$-frame f, we say t is enabled at $(\overline{q_P}, f)$ if the action label of t is same as the first component of f, $q \in \overline{q_P}$ and the frame f satisfies the guard g, or in other words, $r^i_{\mathcal{P}} \leq_f r^j_{\mathcal{P}}$ holds if $g = r^i_{\mathcal{P}} \leq r^j_{\mathcal{P}}$ and $r^i_{\mathcal{P}} \leq_f d$ holds if $g = r^i_{\mathcal{P}} \leq d$. We next define the notions of symbolic updates and symbolic steps which we shall use to design the rules of the finite parity game simulating the concrete game.

▶ **Definition 17** (Symbolic update). *Given an $\lceil s \rceil_{k+1}$-frame f, and an update $u_{\mathcal{P}}$ (or resp. a pair of updates $u_{\mathcal{P}_1}, u_{\mathcal{P}_2}$ with $\mathcal{P}_1 \neq \mathcal{P}_2$) we define the updated partial $\lceil s+1 \rceil_{k+1}$-frame pf to be such that (f, pf) is one-step compatible, $r_{\mathcal{P}} \equiv_{pf} X^{-1} r_{\mathcal{P}}$ for all $r_{\mathcal{P}} \notin u_{\mathcal{P}}$, $r'_{\mathcal{P}} \equiv_{pf} X^{-1} d$, for all $r'_{\mathcal{P}} \in u_{\mathcal{P}}$ and $r_{\mathcal{P}'} \equiv_{pf} X^{-1} r_{\mathcal{P}'}$ for all $r_{\mathcal{P}'} \in R_{\mathcal{P}'}$, for all $\mathcal{P}' \neq \mathcal{P}$ (resp. for a pair of updates, we update the register variables of both $\mathcal{P}_1$ and $\mathcal{P}_2$ and the remaining variables are equal to their previous values).*

▶ **Definition 18** (Symbolic step). *Given a symbolic configuration $sc = (\overline{q_P}, f)$, a partial symbolic configuration $psc' = (\overline{q_P}', pf')$, an action a and a pair of output and input transitions t_o, t'_i, where $\overline{q_P}, \overline{q_P}' \in \prod_{\mathcal{P} \in \overline{\mathcal{P}}} B_{\mathcal{P}}$, f is some $\lceil s \rceil_{k+1}$-frame and pf' is a partial $\lceil s+1 \rceil_{k+1}$-frame, we define a symbolic step $sc \xrightarrow{(a, t_o, t'_i)}_{\mathcal{S}} psc'$ iff the following conditions are met: $t_o = (q_{\mathcal{P}_o}, g_{\mathcal{P}_o}, u_{\mathcal{P}_o}, q'_{\mathcal{P}_o}) \in T^E_{\mathcal{P}_o}$, $t'_i = (q_{\mathcal{P}_i}, g_{\mathcal{P}_i}, u_{\mathcal{P}_i}, q'_{\mathcal{P}_i}) \in T^E_{\mathcal{P}_i}$, t_o, t'_i are enabled at sc, $\mathcal{P}_o \neq \mathcal{P}_i$, $ac(t_o) = (!, a)$, $ac(t'_i) = (?, a)$, $q'_{\mathcal{P}_o}, q'_{\mathcal{P}_i} \in \overline{q_P}'$ and pf' is obtained by updating f with the updates $u_{\mathcal{P}_o}$ and $u_{\mathcal{P}_i}$. We can define the symbolic step $sc \xrightarrow{(a, t_o, *)}_{\mathcal{S}} psc'$ in a similar manner.*

We know that any LTL formula ϕ can be converted to an equivalent non-deterministic Büchi automaton with an exponential number of states in the size of ϕ in EXPTIME [15]. Now, every non-deterministic Büchi automaton with n states can be converted to a deterministic parity automaton [9, Chapter 1] with number of states exponential in n and number of colours polynomial in n [13, Theorem 3.10]. Using these results, it is easy to see that given a CLTL formula ϕ, we can construct a deterministic parity automaton $\mathcal{A}_\phi$ with set of states Q and with number of colours l, accepting the set of all sequences of frames that symbolically satisfy ϕ, such that $|Q|$ is double exponential in the size of ϕ and l is exponential in the size of ϕ.

In order to construct a winning strategy for SYSTEM in the concrete game using a winning strategy for SYSTEM in the symbolic parity game that we shall construct, we also need to ensure that every sequence of frames resulting from a play of the game conforming to the winning strategy admits integer labellings for every variable in each of the frames. As done in [7] and [3], it is possible to construct a Büchi automaton of size polynomial in the size of the formula ϕ and hence a deterministic parity automaton $\mathcal{A}_{chain}$ of size exponential in the size of ϕ that checks for the absence of infinite chains in a symbolic model ρ satisfying ϕ. We shall use $\mathcal{A}_\phi$ and $\mathcal{A}_{chain}$ to define the parity winning condition of the symbolic game and we will see later that this ensures the existence of winning strategies with integer labellings for the sequence of frames along every play conforming to certain type of winning strategies.

▶ **Definition 19** (Symbolic game). *Given an interface register transition system $\mathcal{S} = \langle \overline{\mathcal{P}}, A, \mathrm{ac} \rangle$ and a CLTL formula ϕ over A and a single variable d with X-length k, we associate with it a* symbolic *game. Let $\mathcal{F}$ denote the set of all s-frames for $s \in [0, k+1]$ and $\mathcal{PF}$ denote the set of all partial s-frames for $s \in [0, k+1]$. Let $\mathcal{A} = \mathcal{A}_\phi \times \mathcal{A}_{chain}$ be a deterministic parity automaton accepting the set of all sequences of frames that symbolically satisfy ϕ and do not contain infinite chains, with Q being the set of states, $q_I \in Q$ being the initial state and l being the number of colours. We define a parity game with initial positions $\{(\overline{q^0_{\mathcal{P}}}, pf_0, t_o, q_I)\}$. Here, $\overline{q^0_{\mathcal{P}}}$ is the tuple of initial states of each of the FSRMs in $\mathcal{S}$, pf_0 is the partial 1-frame where all register variables are equal and t_o is some output transition whose source state is a current state in $\overline{q^0_{\mathcal{P}}}$. The set of other* SPOILER *positions is $\prod_{\mathcal{P} \in \overline{\mathcal{P}}} B_{\mathcal{P}} \times (T^I_{\overline{\mathcal{P}}} \cup \{*\}) \times \mathcal{PF} \times T^O_{\overline{\mathcal{P}}} \times Q$. The set of* SYSTEM *positions is $(\prod_{\mathcal{P} \in \overline{\mathcal{P}}} B_{\mathcal{P}} \times (T^I_{\overline{\mathcal{P}}} \cup \{*\}) \times \mathcal{F} \times T^O_{\overline{\mathcal{P}}} \times Q) \cup \{(\overline{q^0_{\mathcal{P}}}, f_0, t_o, q_I)\}$. Position $(\overline{q^0_{\mathcal{P}}}, pf_0, t_o, q_I)$ receives the same colour as that of the initial state q_I in $\mathcal{A}$ and positions $(t_i, \overline{q_{\mathcal{P}}}, pf, t_o, q)$, $(*, \overline{q_{\mathcal{P}}}, pf, t_o, q)$, $(t_i, \overline{q_{\mathcal{P}}}, f, t_o, q)$ and $(*, \overline{q_{\mathcal{P}}}, f, t_o, q)$ receive the same colour as that of state q in $\mathcal{A}$*

We denote the set of spoiler positions by P_{spo} and set of system positions by P_{sys} The symbolic game starts at an initial position $(\overline{q^0_{\mathcal{P}}}, pf_0, t_o, q_I)$. From any position $(\overline{q^0_{\mathcal{P}}}, pf_0, t_o, q_I)$, $(t_i, \overline{q_{\mathcal{P}}}, pf, t_o, q)$ or $(*, \overline{q_{\mathcal{P}}}, pf, t_o, q)$, SYSTEM makes a move by choosing a position $(\overline{q^0_{\mathcal{P}}}, f_0, t_o, q_I)$, $(t_i, \overline{q_{\mathcal{P}}}, f, t_o, q)$ or $(*, \overline{q_{\mathcal{P}}}, f, t_o, q)$ where f_0 is the 1-frame with all register variables and the variable d equal to each other and f is a frame such that (pf, f) is one-step compatible and is such that f satisfies the guard g appearing in t_o.

SPOILER can respond by moving to any position $(t'_i, \overline{q_{\mathcal{P}}}', pf', t'_o, q')$ (resp. $(*, \overline{q_{\mathcal{P}}}', pf', t'_o, q')$) provided the following conditions are satisfied:

- There is a symbolic step from $(\overline{q_{\mathcal{P}}}, f)$ to $(\overline{q_{\mathcal{P}}}'', pf')$ on (a, t_o, t'_i) or $(*, t_o, t'_i)$ as defined in 18 and $\overline{q_{\mathcal{P}}}'$ is reachable from $\overline{q_{\mathcal{P}}}''$ by a sequence of hidden transitions. Formally, $(\overline{q_{\mathcal{P}}}, f) \xrightarrow{(a, t_o, t'_i)}_{\mathcal{S}} (\overline{q_{\mathcal{P}}}'', pf')$ (resp. $(\overline{q_{\mathcal{P}}}, f) \xrightarrow{(a, t_o, *)}_{\mathcal{S}} (\overline{q_{\mathcal{P}}}'', pf')$).
- Transition t'_o is enabled at the partial symbolic configuration $(\overline{q_{\mathcal{P}}}', pf')$
- There is a transition from q to q' on reading the frame f in the deterministic parity automaton $\mathcal{A}$.

The game then continues as before from the new position. A play is an infinite sequence of alternating SPOILER and SYSTEM positions: $p_0 p'_0 p_1 p'_1 p_2 p'_2 \cdots$ such that successive positions satisfy the conditions above. Histories are finite prefixes of plays, ending at SPOILER positions. SYSTEM wins a play of the game if the parity condition is satisfied by the play. Let H denote the set of all histories. Given a play π, let $\pi_f(\pi)$ be the projection of the play to the underlying sequence of frames contained in each SYSTEM position.

▶ **Definition 20** (Strategies). *A strategy for* SYSTEM *in the symbolic game is a function $st : H \to P_{sys}$.*

▶ **Definition 21** (Positional Strategies). *A positional strategy for* SYSTEM *in the symbolic game is a function $st : P_{spo} \to P_{sys}$.*

A play $p_0 p'_0 p_1 p'_1 p_2 p'_2 \cdots$ conforms to the strategy (resp. positional strategy) st if for all $i \geq 0$, $p'_i = st(p_0 v_0 \cdots p_i)$ (resp. $p'_i = st(p_i)$). We say that a strategy/positional strategy st is a winning strategy for SYSTEM if all plays conforming to it are won by SYSTEM. We abuse notation and say $t_1 \leq_{p'} t_2$ for $t_1, t_2 \in V_{\mathcal{S}}$ and $p' \in P_{sys}$ to mean $t_1 \leq_f t_2$ where f is the frame contained in position p'.

We associate a finitely branching infinite tree with a positional strategy st. Let Π_{st} be the collection of all plays that conform to strategy st and let H_{st} be the collection of all finite prefixes of plays in Π_{st} ending in a SPOILER position. Now let $\pi_{spo}(H_{st})$ denote the

projection of such histories to just the sequence of SPOILER positions. Now we define a tree associated with positional strategy st as T: $\pi_{spo}(H_{st}) \to P_{sys}$ given by $T(p_0 p_1 \dots p_i) = st(p_i)$. For every node η in T, its set of children corresponds to the set of SPOILER moves possible for the SYSTEM move $T(\eta)$. For a node η, the subtree T_η rooted at η is such that for all η', $T_\eta(\eta') = T(\eta \cdot \eta')$. A tree T is called *regular* if the set $\{T_\eta \mid \eta \in \pi_{spo}(H_{st})\}$ is finite, i.e., there are only finitely many subtrees up to isomorphism. Two nodes η, η' are said to be isomorphic if $T_\eta = T_{\eta'}$. It is easy to verify that the tree T associated with a positional winning strategy is a regular tree.

Given an infinite path $\pi \in \pi_{spo}(\Pi_{st})$ of the tree T, we associate with it, a symbolic model ρ, which is the infinite sequence of frames obtained from the labels of the nodes along that path.

We now give an automata-theoretic characterization for trees that are labellable. Using the fact that the trees associated with positional strategies are regular and using results from [3], we get that such trees are labellable if and only if the symbolic model associated with each infinite path of the tree does not have any infinite chain. Recall that the automaton $\mathcal{A}_{chain}$ we defined earlier, checks for absence of infinite chains in a symbolic model and therefore, we get the following lemma:

▶ **Lemma 22** ([3, Lemma 23]). *The tree T associated with a positional strategy st is labellable iff the symbolic model associated with every infinite path of T is recognized by the deterministic parity automaton $\mathcal{A}_{chain}$.*

Now, we know that if a positional strategy is winning, then the sequence of frames along every play conforming to the strategy must not have any infinite chains, hence no path in the tree associated with this strategy should have any infinite chains. Using Lemma 22, we then get that:

▶ **Lemma 23.** *The tree T associated with a positional winning strategy st is labellable.*

We now extend the map μ to go from histories h_c (resp. plays π_c) of the concrete game to histories h_s (resp. plays π_s) of the symbolic game. We define a function μ: $H_c \to H_s$ as follows: $\mu((c_0, t_0)v_0(t_0', c_1, t_1)v_1 \dots (t_i', c_{i+1}, t_{i+1}))$ equals $(\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I)(\overline{q_{\mathcal{P}}^0}, f_0, t_0, q_I)(t_0', \overline{q_{\mathcal{P}}^1}, pf_1, t_1, q_1)(t_0', \overline{q_{\mathcal{P}}^1}, f_1, t_1, q_1) \dots (t_i', \overline{q_{\mathcal{P}}^{i+1}}, pf_{i+1}, t_{i+1}, q_{i+1})$ where pf_j is the partial $\lceil j+1 \rceil_{k+1}$-frame induced by the configurations $c_{j-\lceil j \rceil_{k+1}}, \dots c_j$ and the integer values $v_{j-\lceil j \rceil_{k+1}}, \dots v_{j-1}$ and f_j is the $\lceil j+1 \rceil_{k+1}$-frame induced by the configurations $c_{j-\lceil j \rceil_{k+1}}, \dots c_i$ and the integer values $v_{j-\lceil j \rceil_{k+1}}, \dots v_j$, $\overline{q_{\mathcal{P}}^j}$ is the tuple of states in configuration c_j and q_j is the state of DPA $\mathcal{A}$ on reading the sequence of frames $f_0 \dots f_{j-1}$ for all $0 \le j \le i$. μ is defined over plays of the concrete game in a similar manner. We now use all the above definitions and lemmas to establish the equivalence between the concrete and the symbolic games.

▶ **Lemma 24** (Equivalence between concrete and symbolic games). *SYSTEM has a winning strategy in the concrete game if and only if SYSTEM has a winning strategy in the symbolic game*

Proof. ($\Rightarrow$) Suppose SYSTEM has a winning strategy st in the CLTL game. We will construct a winning strategy st_p for SYSTEM in the symbolic game. Any history h_s of length 1 is of the form $(\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I)$ and it is clear that $\mu((c_0, t_0)) = (\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I))$. Therefore, we define $st_p((\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I)) =$ last position of $\mu((c_0, t_0) \cdot st((c_0, t_0)))$.

We assume inductively that the strategy st_p has been defined for all histories h_s in the symbolic game of length $j \le i$ and that there exists a history h_c of length j in the concrete game such that $\mu(h_c) = h_s$. Now, consider a history $h_s^{i+1} = (\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I)(\overline{q_{\mathcal{P}}^0}, f_0, t_0, q_I) \dots$

$(t'_i, \overline{q_{\mathcal{P}}^{i+1}}, pf_{i+1}, t_{i+1}, q_{i+1})$ generated after $i+1$ rounds of the game. By induction hypothesis, there exists a history h_c^i of length i of the form $(c_0,t_0)v_0(t'_0,c_1,t_1)v_1\dots(t'_{i-1},c_i,t_i)$ such that $\mu(h_c^i) = (\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I)(\overline{q_{\mathcal{P}}^0}, f_0, t_0, q_I)(t'_0, \overline{q_{\mathcal{P}}^1}, pf_1, t_1, q_1)(t'_0, \overline{q_{\mathcal{P}}^1}, f_1, t_1, q_1)\dots(t'_{i-1}, \overline{q_{\mathcal{P}}^i}, pf_i, t_i, q_i)$. Let $v_i = st((c_0,t_0)v_0(t'_0,c_1,t_1)v_1\dots(t'_{i-1},c_i,t_i))$ [The proof fails at this step if we allow the guards on the output transitions to use the full syntax instead of the restricted syntax. See note at the end of the proof]. Now, let $c_i \xrightarrow{a,t_i,t'_i,v_i}_{\mathcal{S}} c_{i+1}$. Thus, we now have $h_c^{i+1} = (c_0,t_0)v_0(t'_0,c_1,t_1)v_1\dots(t'_{i-1},c_i,t_i)(t'_i,c_{i+1},t_{i+1})$ such that $\mu(h_c^{i+1}) = h_s^{i+1}$ and therefore, $st_p(h_s^{i+1}) =$ last position of $\mu(h_c^{i+1}\cdot st(h_c^{i+1}))$.

Let π_s be any infinite play in the parity game that SYSTEM plays according to the strategy derived from st_p. By the inductive definition, there exists a play π_c in the concrete game played according to the strategy st such that $\pi_s = \mu(\pi_c)$. Now, since st is a winning strategy for the concrete game, π_c must be winning for SYSTEM in the concrete game, or in other words the concrete model underlying π_c must satisfy the CLTL formula ϕ. Therefore, by Lemma 14, the infinite sequence of frames along π_s must satisfy ϕ symbolically and since it admits an integer labelling it cannot have infinite chains. Therefore, π_s satisfies the parity winning condition of $\mathcal{A}$ or in other words, st_p is a winning strategy for SYSTEM in the symbolic game.

($\Leftarrow$) We know that if there is a winning strategy in a parity game then there must also be a positional winning strategy. Let st_p be such a positional winning strategy for SYSTEM in the parity (symbolic) game and let T_p be the tree associated with st_p. By Lemma 23, we know that T_p admits a labelling function L_p given by $L_p(\eta)\colon V_{\mathcal{S}} \to \mathbb{Z}$ for all $\eta \in \pi_{spo}(H_{st_p})$ such that $\hat{T}(\pi) = \mu(\hat{L}(\pi))$ for every infinite sequence $\pi \in \pi_{spo}(\Pi_{st_p})$. Now suppose $L_p((\overline{q_{\mathcal{P}}^0}, pf_0, t_0, q_I))(x) = r$ for all $x \in V_{\mathcal{S}}$ (all $x \in V_{\mathcal{S}}$ must have the same label as they are equal wrt pf_0). We construct a new label function L'_p given by $L'_p(\eta)(x) = L_p(\eta)(x) - r$ for all $\eta \in \pi_{spo}(H_{st_p})$ and all $x \in V_{\mathcal{S}}$. Clearly, L'_p is also a valid labelling function satisfying $\hat{T}(\pi) = \mu(\hat{L'}_p(\pi))$ for every infinite sequence $\pi \in \pi_{spo}(\Pi_{st_p})$. We will use this to inductively construct a winning strategy st for the SYSTEM in the concrete game.

Consider a history h_c^i of length $i \geq 1$ in the concrete game. We define $st(h_c^i) = L'_p(\pi_{spo}(\mu(h_c^i)))(d)$. This labelling also respects the fact that the inital values of all the register variables are 0 in the concrete game and assigns values accordingly. Now consider any play π_c played according to strategy st. Clearly, by Lemma 14, the underlying concrete model $\alpha \models \phi$ as the symbolic model $\mu(\alpha)$ satisfies the formula ϕ symbolically as $\mu(\alpha)$ is the infinite frame sequence along the play $\pi_s = \mu(\pi_c)$ which is winning for the symbolic game and therefore satisfies the parity winning condition. Thus, $\pi_{av}(\pi_c) \models \phi$ or in other words, π_c is winning for SYSTEM and therefore, st is indeed a winning strategy for SYSTEM in the concrete game. ◀

As noted in the proof, the last output transition t_i may not actually be enabled at configuration c_i of the concrete game if the guard on t_i necessitates the newly generated value to be strictly between two consecutive integer values in c_i (notice that such a guard is used in the undecidability proof for simulating a counter increment, see Appendix A). The guard on t_i can indeed impose this condition if the guards on output transitions are allowed to use the full syntax instead of the restricted syntax.

We know that checking for the existence of a winning strategy of a player in a finite parity game is decidable. Thus, using Proposition 11, Lemma 24 and the decidability of parity games, we have the following main result:

▶ **Theorem 25** (Decidability of Interoperability). *Let S be an interface register transition system such that guards on all the output transitions obey the restricted syntax and let ϕ be a CLTL formula. The problem of checking whether S is interoperable satisfying ϕ is decidable.*

7 Discussion and Future Work

The deterministic parity automaton $\mathcal{A}$ that checks for bounded chains and symbolic satisfiability has size doubly exponential with respect to the size of the formula as already discussed before. The number of frames and partial frames is exponential in the number of registers of transition system $\mathcal{S}$. Therefore, the size n of the parity game graph is polynomial in the number of states and transitions in $\mathcal{S}$ and exponential in the number of registers. As we saw earlier, the number of colours l is exponential in the size of the formula. The interoperability problem for the decidable fragment can therefore be decided in $O(n^{log(l)})$. This gives us a 2EXPTIME upper bound for the interoperability problem.

We want to point out here that while it may seem that technically many of the ideas are similar those used in solving the realizability problem for CLTL in [3], the main difference is that we are addressing a problem about a transition system whereas in CLTL realizability, there is only a formula and no transition system. It is actually possible to encode the transition system as a CLTL formula. Using such an encoding, it is indeed possible to reduce the interoperability problem to the single-sided realizability problem for CLTL. But we know that single-sided realizability for CLTL over the integers is 2EXPTIME-complete. Hence, the upper bound we get will be doubly exponential in the size of the transition system. Based on the discussion above, the approach we have taken gives us a better upper bound in terms of the size of the transition system.

In this paper, we allow only integer values to be exchanged between finite state register machines. But one can also consider models where rational or real values could be exchanged. It turns out that the interoperability problem for interface register transition systems which exchange values coming from some dense, open domain is in fact decidable for the general syntax. For dense and open domains it turns out that satisfiability of any guard is guaranteed by its symbolic satisfiability ([7], [3]) (unlike in the case of integers, where it is not possible to satisfy a guard that demands that the newly generated value be between two registers whose values happen to be consecutive). Therefore, the equivalence established by Lemma 14 would hold even for the general syntax. Also, we do not need to check for the absence of unbounded chains for dense, open domains and therefore, we get decidability for the general syntax by reducing it to a parity game with in fact, a simpler parity winning condition.

We defined frames and partial frames to be total orders over the set of terms. But there could be pairs of variables that might never be compared. So, in principle, it is possible to try to avoid total orders and check if frames can be defined as partial orders. It would be interesting to see if this leads to better efficiency.

References

1 Sarra Alqahtani, Xinchi He, Rose Gamble, and Mauricio Papa. Formal verification of functional requirements for smart contract compositions in supply chain management systems. In *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020. doi:10.24251/HICSS.2020.650.

2 Ashwin Bhaskar and M Praveen. Constraint ltl with remote access. In *43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2023)*, pages 41:1–41:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.FSTTCS.2023.41.

3 Ashwin Bhaskar and M Praveen. Realizability problem for constraint ltl. *Information and Computation*, 296:105126, 2024. doi:10.1016/J.IC.2023.105126.

4 Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theoretical Computer Science*, 458:49–60, 2012. doi:10.1016/J.TCS.2012.07.038.

5 Luca De Alfaro and Thomas A Henzinger. Interface automata. *ACM SIGSOFT Software Engineering Notes*, 26(5):109–120, 2001. doi:10.1145/503209.503226.

6 Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Toruńczyk. Energy and mean-payoff games with imperfect information. In *International Workshop on Computer Science Logic*, pages 260–274. Springer, 2010. doi:10.1007/978-3-642-15205-4_22.

7 Stéphane Demri and Deepak D'Souza. An automata-theoretic approach to constraint LTL. *Information and Computation*, 205(3):380–415, 2007. doi:10.1016/j.ic.2006.09.006.

8 Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. Church synthesis on register automata over linearly ordered data domains. *Formal Methods in System Design*, 61(2):290–337, 2022. doi:10.1007/S10703-023-00435-W.

9 Erich Gradel and Wolfgang Thomas. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media, 2002.

10 Lukáš Holík, Malte Isberner, and Bengt Jonsson. Mediator synthesis in a component algebra with data. In *Correct System Design: Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015, Proceedings*, pages 238–259. Springer, 2015. doi:10.1007/978-3-319-23506-6_16.

11 Gabor Madl, Luis Bathen, German Flores, and Divyesh Jadav. Formal verification of smart contracts using interface automata. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 556–563. IEEE, 2019.

12 Marvin L Minsky. Recursive unsolvability of post's problem of" tag" and other topics in theory of turing machines. *Annals of Mathematics*, 74(3):437–455, 1961.

13 Nir Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3, 2007. doi:10.2168/LMCS-3(3:5)2007.

14 Ayleen Schinko, Walter Vogler, Johannes Gareis, N Tri Nguyen, and Gerald Lüttgen. Interface automata for shared memory. *Acta Informatica*, 59(5):521–556, 2022. doi:10.1007/S00236-021-00408-8.

15 Moshe Y Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331. IEEE Computer Society, 1986.

A Proof of undecidability stated in Section 3

An n-counter machine is a tuple $(B, q_{\text{init}}, n, \delta)$ where B is a finite set of states, $q_{\text{init}} \in B$ is an initial state, $c_1, \ldots, c_n$ are n counters and δ is a finite set of instructions of the form "$(q : c_i := c_i + 1; \text{goto } q')$" or "$(q : \text{If } c_i = 0 \text{ then goto } q' \text{ else } c_i := c_i - 1; \text{ goto } q'')$" where $i \in [1, n]$ and $q, q', q'' \in B$. A configuration of the machine is described by a tuple $(q, m_1, \ldots, m_n)$ where $q \in B$ and $m_i \in \mathbb{N}$ is the content of the counter c_i. The possible computation steps are defined as follows:

1. $(q, m_1, \ldots, m_n) \to (q', m_1, \ldots, m_i + 1, \ldots, m_n)$ if there is an instruction $(q : c_i := c_i + 1; \text{goto } q')$. This is called an incrementing transition.
2. $(q, m_1, \ldots, m_n) \to (q', m_1, \ldots, m_n)$ if there is an instruction $(q : \text{If } c_i = 0 \text{ then goto } q' \text{ else } c_i := c_i - 1; \text{ goto } q'')$ and $m_i = 0$. This is called a zero testing transition.
3. $(q, m_1, \ldots, m_n) \to (q'', m_1, \ldots, m_i - 1, \ldots, m_n)$ if there is an instruction $(q : \text{If } c_i = 0 \text{ then goto } q' \text{ else } c_i := c_i - 1; \text{ goto } q'')$ and $m_i > 0$. This is called a decrementing transition.

A counter machine is *deterministic* if for every state q, there is at most one instruction of the form $(q : c_i := c_i + 1; \text{goto } q')$ or $(q : \text{If } c_i = 0 \text{ then goto } q' \text{ else } c_i := c_i - 1; \text{ goto } q'')$ where $i \in [1, n]$ and $q', q'' \in B$. This ensures that for every configuration $(q, m_1, \ldots, m_n)$ there exists at most one configuration $(q', m'_1, \ldots, m'_n)$ so that $(q, m_1, \ldots, m_n) \to (q', m'_1, \ldots, m'_n)$.

For our undecidability results we will use deterministic 2-counter machines (i.e., $n = 2$), henceforward just "counter machines". Given a counter machine $(B, q_0, 2, \delta)$ and two of its states $q_{\text{init}}, q_{\text{fin}} \in B$, the reachability problem is to determine if there is a sequence of transitions of the 2-counter machine starting from the configuration $(q_{\text{init}}, 0, 0)$ and ending at the configuration $(q_{\text{fin}}, n_1, n_2)$ for some $n_1, n_2 \in \mathbb{N}$. It is known that the reachability problem for deterministic 2-counter machines is undecidable [12]. To simplify our undecidability results we further assume, without any loss of generality, that there exists an instruction $\hat{t} = (q_{\text{fin}} : c_1 := c_1 + 1; \text{goto } q_{\text{fin}}) \in \delta$.

Given a counter machine, we design an instance of the interoperability problem for an interface register transition system $\mathcal{S}$ consisting of just two FSRMs $\mathcal{P}$ and $\mathcal{Q}$ such that the counter machine reaches the halting configuration iff the answer to the interoperability problem for $\mathcal{S}$ is a "yes". Let T be the set of transitions of the counter machine. We construct an interface register transition system $\mathcal{S}$ whose set of actions $A = T \cup \{D\} \cup \{E\}$ where D, E are special actions. The FSRM $\mathcal{P}$ consists of states B of the counter machine and has three registers $r^0_{\mathcal{P}}, r^1_{\mathcal{P}}, r^2_{\mathcal{P}}$. The first register is intended to carry the value 0 and the last two are intended to carry the two counter values.

For every incrementing transition t of the counter machine from q to q', $\mathcal{P}$ has an output transition labeled with action t from q to q'. It has the guard $r^i_{\mathcal{P}} < d$ (syntactic sugar for $(r^i_{\mathcal{P}} \leq d) \ \wedge \ \neg(d \leq r^i_{\mathcal{P}})$), where c_i is the counter incremented by t. The value generated is stored the register $r^i_{\mathcal{P}}$. This ensures that the value generated by $\mathcal{P}$ while executing this transition is strictly greater than the current value of $r^i_{\mathcal{P}}$. But we need more work to ensure that the value generated by $\mathcal{P}$ is exactly one more than the current value of $r^i_{\mathcal{P}}$. We will describe later how to achieve this using the other FSRM $\mathcal{Q}$.

For every decrementing or zero testing transition t of the counter machine from q to q', $\mathcal{P}$ has an output transition labeled with action t from q to q'. It has the guard $d \leq r^i_{\mathcal{P}}$, where c_i is the counter decremented or tested for zero by t. The generated value is stored in the register $r^i_{\mathcal{P}}$.

The FSRM $\mathcal{Q}$ consists of states $B \cup (B \times T) \cup \{q_e\}$, (where B is the set of states of the counter machine), and four registers $r^0_{\mathcal{Q}}, r^1_{\mathcal{Q}}, r^2_{\mathcal{Q}}, r^3_{\mathcal{Q}}$. The first register is intended to store the value 0, the next two are intended to store the two counter values and the last one is intended to store the value generated by $\mathcal{P}$ to be used to detecting simulation errors.

For every incrementing transition t of the counter machine from q to q', $\mathcal{Q}$ has an input transition labeled with action t from q to q'. The received value is stored in the register $r^i_{\mathcal{Q}}$, where c_i is the counter incremented by t. There is also an input transition labeled t from q to (q', t), which stores the received value in $r^3_{\mathcal{Q}}$. From (q', t), there is an output transition labeled with a special action E (for "error") to the state q_e, with the guard $r^i_{\mathcal{Q}} < d \ \wedge \ d < r^3_{\mathcal{Q}}$. This forces $\mathcal{Q}$ to generate a value that is strictly between the old value of c_i and the new value $\mathcal{P}$ generated for that counter. If this transition is enabled, it means $\mathcal{P}$ generated a value that is more than one plus the old value of the counter c_i, which is an error. To let the FSRMs continue to operate infinitely, we add an output transition from (q', t) to itself labeled with a special action D (for "dummy") and an output transition from q_e to itself labeled E.

For every zero testing transition t of the counter machine from q to q', $\mathcal{Q}$ has an input transition labeled with action t from q to q'. The received value is stored in $r^i_{\mathcal{Q}}$, where c_i is the counter that is tested for zero by t. There is also an input transition labeled t from q to (q', t), which stores the received value in $r^3_{\mathcal{Q}}$. From (q', t), there is an output transition labeled with the action E with the guard $r^3_{\mathcal{Q}} \neq r^0_{\mathcal{Q}} \ \vee \ r^i_{\mathcal{Q}} \neq r^0_{\mathcal{Q}}$. This means that either the new counter value generated by $\mathcal{P}$ is non-zero or the old value of c_i was non-zero, both of which are errors. We add an output transition from (q', t) to itself labeled with the action D.

For every decrementing transition t of the counter machine from q to q'', $\mathcal{Q}$ has an input transition labeled with action t from q to q''. The received value is stored in $r^i_{\mathcal{Q}}$, where c_i is the counter that is decremented by t. There is also an input transition labeled t from q to (q'', t), which stores the received value in $r^3_{\mathcal{Q}}$. From (q'', t), there is an output transition labeled with the action E with the guard $(r^3_{\mathcal{Q}} < d \wedge d < r^i_{\mathcal{Q}})$. This forces $\mathcal{Q}$ to check that the value generated by $\mathcal{P}$ was less than the old value of c_i minus one, which is an error. We add an output transition from (q'', t) to itself labeled with the action D. At every state in $\mathcal{P}$, we add a transition to itself labeled with the action D and another transition to itself labeled with the action E. Now, adding gadgets to both these FSRMs as described at the end of Section 3 (to avoid finite traces) gives us a set of extra actions A'.

We now construct a CLTL formula ϕ (basically just an LTL formula over the set of actions $A \cup A'$ in this case) that excludes all strings that either containing E or the suffix $(a')^\omega$ for some $a' \in A'$. Among the remaining, it includes those which have at least one occurrence of D or one occurrence of a transition t that leads to the halting state q_{fin}.

▶ **Lemma 26.** *The halting state is reachable in the counter machine iff the interface register transition system constructed above is interoperable satisfying ϕ.*

Proof. If the halting state is reachable, we can construct a distributed regulator that updates the counter values exactly as the counter machine does. The only traces that respect this regulator are those which faithfully simulate the counter machine and those which execute dummy transitions. Both of these satisfy the formula ϕ.

If the interface register transition system is interoperable, consider a distributed regulator that makes it interoperable. If this regulator provides wrong values of the counter at any point during simulation, some trace respecting the regulator will execute some error transition, which does not satisfy ϕ, a contradiction. Hence, the regulator provides correct counter values. The traces that respect such a regulator are those that execute dummy transitions and the trace that faithfully simulates the counter machine. For this trace to satisfy ϕ, some transition leading to the halting state q_{fin} needs to be included, so the counter machine halts. ◀