Decidability and complexity of checking simulation preorder between data-aware web services

Lakhdar Akroun\(^1\), Boualem Benatallah\(^2\), Lhouari Nourine\(^1\), and Farouk Toumani\(^1\)

\(^1\) LIMOS, CNRS, Blaise Pascal University, Clermont-Ferrand, France
akroun, nourine, ftoumani@isima.fr
\(^2\) CSE, UNSW, Sydney
boualem.benatallah@gmail.com

Abstract. This paper addresses the problem of analysing specifications of data-centric Web service interaction protocols (also called artefact-centric business level protocols). Specifications of such protocols includes data in addition to operation signatures and messages ordering constraints. Reasoning about data-aware service interactions is a complex task because of the inherently infinite states of underlying service execution instances. Our work focuses on characterising the problem of checking a refinement relation between service interaction protocol specifications. More specifically, we consider the problem of checking the simulation preorder \([43]\) when service business protocols are represented using data-aware state machines, namely, the Colombo model \([14]\). In this framework, a service (i) exchanges messages using variables; (ii) act on a shared database; (iii) has a transition based behavior. We show that both simulation and state reachability properties are undecidable in this model. Furthermore, these properties are also undecidable when services do not communicate, and only read from the database. We show also that simulation is decidable when the maximum number of tuple in any instance of database is bounded (denoted DB-bounded service). Although DB-bounded services are also infinite-state systems, a finite symbolic representation of such services can be obtained by partitioning the original infinite state space into a finite number of equivalence classes. Then, a simulation algorithm can be designed using a symbolic procedure that manipulates (finite) sets of states (i.e., the equivalence classes) instead of (infinite) individual states.

Keywords: Data-centric services, artefact-centric business processes, simulation preorder

1 Introduction

Web service technologies facilitate interoperation among autonomous software systems and applications \([6]\). In a nutshell, a Web service is platform-independent
and machine accessible software component that can be described, published, and invoked over the network by using standard communication protocols and description languages.

Main stream service description languages such as WSDL\(^3\) allow descriptions of low-level service operations. Semantic Web-based representation languages (e.g., OWL-S) investigated rich and machine-understandable descriptions of service properties and capabilities. Business protocol representation models and languages (e.g., state machines \([19,12,13]\), Petri-nets \([46,31,40]\)) are description models which are used for specifying external behavior of services. Business protocol representation models and languages play an important role, since they provide to developers information on how to write a client (a service) to correctly interact with a given service, they record the intended behavior of the service \([35]\) and open the possibility of formally analysing and synthesising services. For example, they allow to study business protocol compatibility and substitution \([13]\) and business protocol synthesis \([15,16]\). In the aforementioned research works, simulation preorder plays a fundamental role to solve the considered problems, since it is recognized as an appropriate means for comparing the structures of state-transitions systems. Simulation enables to formalize the idea that a given service is able to faithfully reproduce the externally visible behavior of another service. Simulation equivalence plays also a crucial role in model checking where it is often exploited to minimize the state space explored by verification algorithms \([41,22]\).

In this paper, we consider the problem of analyzing specifications of data-centric services using the relation of simulation preorder. The need of include data in business protocols or specification of web services is widely recognized, and there is increasing interest to model and study such models \([47,52,14,2,35]\). Formal models used to describe such specifications are essentially communicating guarded transitions systems in which transitions are used to model either messages exchanges between a service and its environment (i.e. a client), or actions (i.e., read, write) on a global database shared among existing services. A state in such models is the control state of transition system and a database, the interaction with the environment or actions modify the state of the service. The incorporation of data turns out to be very challenging since it makes service specifications infinite which leads, in most cases, to the undecidability of many verification problems.

We investigate the decidability issue of service simulation in the framework of the Colombo model \([14]\). It is a pioneer data-centric service model that has been used to investigate the service composition problem. This model is a guarded transition system, where a service uses a set of variables to send and receive messages, and it can read or modify a shared database using atomic process, atomic process describes action in terms of its inputs, outputs, preconditions and postconditions. Then a state, is characterised by the control state of the

\(^3\) Web Services Description Language.
transition system, the database instance, and the values of variables. The two sources of infiniteness that make the study of the simulation difficult are:

- services receive messages with variables, where the values comes from an infinite domain, so the number of states after receiving a message is infinite.
- an infinite number of possible initial databases for a given service which makes service specification an infinite state machine. Each execution of service, start with a fixed database, due to infinite domain, the number of initial database instances is infinite.

At first glance, the Colombo model appears to have a limited expressivity since:

- it restricts accesses to the database only through atomic processes, and
- it supports a very limited database ‘query’ language which consists in simple key-based access functions.

We show that even in this restricted framework, the simulation problem is already undecidable. Our proof is based on a reduction from the halting problem of a two counter machine (a Minsky machine) into the state reachability problem in Colombo services, knowing that this later problem can be recast as a simulation problem. Even worse, the way the proof is constructed enables to deduce that the reachability and simulation problems remain undecidable even in the case of non-communicating Colombo services with read-only accesses to the database (i.e., services that are not able to modify the world database or communicate with other services, in this cases, services can only read values from database and store them in variables), namely unbounded Colombo.

The main source of undecidability comes from the ability of a Colombo service to read an unbounded number of values from the database. This is a decidability border since, we show that simulation is decidable for Colombo services with bounded databases, it is the class of services where the number of tuples in any database is at most equal to a constant $k$ (namely DB-bounded services), then the number of values which can be read by the service is bounded. The proof is achieved in two steps:

- first we show that simulation is decidable for Colombo services without any access to the database (denoted DB-less service). DB-less services are also infinite-state systems, because they manipulate variables and values of variables come from an infinite domain, a finite symbolic representation of such services can be obtained by partitioning the original infinite state space (here a state is characterized with control state of transition system and a valuation of variables) into a finite number of equivalence classes. Then, a simulation algorithm can be designed using a symbolic procedure that manipulates (finite) sets of states (i.e., the equivalence classes) instead of (infinite) individual states,
- then we show that DB-bounded services can be rewritten into equivalent DB-less services while preserving the simulation preorder.
The rest of this paper is organized as follows: Section 2 presents the Colombo model and defines the associated simulation problem. Section 3 describes our results regarding undecidability of unbounded Colombo. Section 4 considers the case of DB-less services and show decidability of simulation in this context. Section 5 show how to transform DB-bounded service to DB-less service. Section ?? discusses related works and draw future research directions.

2 Overview on the Colombo Model

We present below a simplified version of the Colombo model which is sufficient to present our results. A detailed description of the Colombo model is given in [14]. We assume some familiarity with relational database concepts (e.g., see [4]). A world database schema, denoted \( W \), is a finite set of relation schemas having the form \( R_k(A_1, \ldots, A_k; B_1, \ldots, B_n) \), where \( A_i \)'s, \( B_j \)'s are attributes and the \( A_i \)'s form a key for \( R_k \). A world database is an instance over the schema \( W \). Let \( R(A_1, \ldots, A_k; B_1, \ldots, B_n) \) be a relation schema in \( W \), then \( f_j^R(A_1, \ldots, A_k) \) is an access function that returns the \( k+j \)-th element of the tuple \( t \) in \( R \) identified by the key \((A_1, \ldots, A_k)\) (i.e., the \( j \)-th element of the tuple \( t \) after the key). Given a set of constants \( C \) and variables \( V \), the set of accessible terms over \( C \) and \( V \) is defined recursively to include all the terms constructed using \( C, V \) and the \( f_j^R \) functions.

Example 1. Fig. 1(c) depicts an example of a world database schema while figure 2 shows an instance of such a schema. For example, access to the relation \( \text{Inventory}(\text{code}, \text{available}, \text{warehouse}, \text{price}) \) is only possible through the access function \( f_j^{\text{Inventory}}(\text{code}) \) with \( j \in [1, 3] \). For instance, using the world database depicted at figure 2, the function \( f_2^{\text{Inventory}}(\text{“HP15”}) \) returns the value “NGW”, corresponding to the value of the second attribute (i.e., the attribute \text{warehouse}) of the tuple identified by the code “HP15” in the relation \text{Inventory}.

2.1 atomic processes

In the Colombo model, services actions are achieved using the notion of atomic processes. An atomic process is a triplet \( p = (I, O, CE) \) where: \( I \) and \( O \) are respectively input and output signatures (i.e., sets of typed variables) and \( CE = \{((\theta, E))\} \), is a set of conditional effects, with:

- Condition \( \theta \) is a boolean expression over atoms over accessible terms over some family of constants and the input variables \( u_1, \ldots, u_n \) in \( I \),
- A set of effects \( E \) where each effect \( e \in E \) is a pair \((es, ev)\) with:
  - \( es \), effect on world state, is a set of modifications on the global database, i.e., expressions of the form

\footnote{In particular, we omit notions like QStore, linkage, ..., which are not relevant for our purposes.}
\textbf{Example 2.} Fig. 1(b) shows an example of a specification of an atomic process \texttt{CheckItem}. This process takes as input an item code \texttt{(item)} and a customer number \texttt{(cust)} and checks first if the requested item is available in the relation \texttt{inventory} (condition if $f^\text{inventory}_1(\text{item}) = T$). If the requested item is not available, the process \texttt{CheckItem} simply returns the output parameter \texttt{avail} = $F$. Otherwise, if the requested item is available, the process returns the warehouse the item is available in and the price. Moreover, if the price of the requested item is greater than 50 and the status of the current customer is defined (condition
if price > 50 and \( f^1_{\text{Customers}}(\text{cust}) \neq \omega \), then a new order id is created and inserted in the relation `Orders`. Otherwise, the process returns the output parameter `ord = 0`.

### 2.2 guarded automata

The behavior of a Colombo service is given by the notion of *guarded automata* as defined below.

**Definition 1.** A *guarded automaton* (GA) of a service \( S \) is a tuple \( \text{GA}(S) = \langle Q, \delta, q_0, F, L\text{Store}(S) \rangle \), where:

- \( Q \) is a finite set of control states with \( q_0 \in Q \) the initial state,
- \( F \subseteq Q \) is a set of final states,
- \( L\text{Store}(S) \) is a finite set of typed variables,
- the transition relation \( \delta \) contains tuples \( (q, \theta, \mu, q') \) where \( q, q' \in Q \), \( \theta \) is a condition over \( L\text{Store}(S) \) (no access to world instance), and \( \mu \) has one of the following forms:
  - *(incoming message)* \( \mu = ?m(v_1, \ldots, v_n) \) where \( m \) is a message having as signature \( m(p_1, \ldots, p_n) \), and \( v_i \in L\text{Store}(S), \forall i \in [1, n] \), or
  - *(send message)* \( \mu = !m(b_1, \ldots, b_n) \) where \( m \) is a message having as signature \( m(p_1, \ldots, p_n) \), and \( \forall i \in [1, n] \), each \( b_i \) is either a variable of \( L\text{Store}(S) \) or a constant, or
  - *(atomic process invocation)* \( \mu = p(u_1, \ldots, u_n; v_1, \ldots, v_m, CE) \) with \( p \) an atomic process having \( n \) inputs, \( m \) outputs and \( CE \) as conditional effects, and \( \forall i \in [1, n] \), each \( u_i \) (respectively, \( v_i \)) is either a variable of \( L\text{Store}(S) \) or a constant.

A message type has the form \( m(p_1, \ldots, p_n) \) where \( m \) is the message name and \( p_1, \ldots, p_n \) are message parameters. Each parameter \( p_i \) is defined over a domain \( D \).

\( L\text{Store}(S) \) can be viewed as a working area of a service. The variables of \( L\text{Store}(S) \) are used to (i) capture the values of incoming messages, (ii) capture the output values of atomic processes, (iii) populate the parameters of outgoing messages, and (iv) populate the input parameters of atomic processes.

**Example 3.** Fig. 1(a), inspired from [14], shows the guarded automata of a *Warehouse* service. The states of the automata represent the different phases that the service may go through during its execution. Transitions are associated with a send or a receive message or with an atomic process. The *Warehouse* service is initially at its initial state (i.e., the state indicated in the figure by an unlabeled entering arrow). The service starts its execution upon receiving a `requestOrder` message. Then, depending on the requested payment mode and the price, respectively given by the values of the received message parameters `payBy` and `price`, the service can make two possible moves: (i) if the payment mode is `CC` (credit card) or the price > 10, the service sends a `requestCCCheck` message, for example to a bank, in order the check whether the credit card can be used...
to make the payment, or (ii) if the payment mode is PREPAID and the price \( \leq 10 \), the service will execute the atomic process charge in order to achieve the payment. The service ends its execution at a final state, depicted in the figure by double-circled states.

If a given guarded automaton \( GA(S) \) uses only transitions of the form \((q, \theta, \mu, q')\) with \( \mu \) is an atomic process, in this case the corresponding service \( S \) is called a non-communicating service (since \( S \) cannot exchange messages with its environment). Moreover, if all the atomic processes used in a guarded automaton \( GA(S) \) have no effects on world states (i.e., the set \( es \) of each atomic process is empty), in this case the service \( S \) is called a read-only Colombo service.

### 2.3 Service runs

**Semantics** We use the notion of an extended automata to define the semantics of a Colombo service. At every point in time, the behavior of an instance of a Colombo service \( S \) is determined by its instantaneous description (or simply, configuration). A configuration of a service is given by a triplet \( id = (l, I, \alpha) \) where \( l \) is its current control state, \( I \) a world database instance and \( \alpha \) is a valuation over the variables of \( LStore \).

**Definition 2. (service runs)**

Let \( GA(S) = \langle Q, \delta, l_0, F, LStore(S) \rangle \) be a guarded automata of a service \( S \).

A run \( r \) of \( S \) is a finite sequence \( r = id_0 \xrightarrow{\mu_0} id_1 \xrightarrow{\mu_1} \ldots \xrightarrow{\mu_{n-1}} id_n \) which satisfy the following conditions:

- (Initiation) \( id_0 = (l_0, I_0, \alpha_0) \) is an initial configuration of the run with \( I_0 \) is an arbitrary database over \( W \) and \( \alpha_0(x) = \omega, \forall x \in LStore(S) \).

- (Consecution) \( \forall i \in [1, n], id_i = (l_i, I_i, \alpha_i) \) and there is a transition \((l_i, \theta, \mu, l_{i+1}) \in \delta \) such that \( \alpha_i(\theta) \equiv true \) and one of the following conditions holds:
  
  - \( \mu = ?m(v_1, \ldots, v_n) \) and \( \mu_i = ?m(c_1, \ldots, c_n) \), with \( c_j \) a constant \( \forall j \in [1, n] \), then \( I_{i+1} = I_i \) and \( \alpha_{i+1}(v_j) = c_j \) and \( \forall v \in LStore(S) \setminus \{v_1, \ldots, v_n\}, \alpha_{i+1}(v) = \alpha_i(v) \).
  
  - \( \mu = \mu m(b_1, \ldots, b_n) \) and \( \mu_i = \mu m(\alpha_i(b_1), \ldots, (\alpha_i(b_n))) \) then \( I_{i+1} = I_i \) and \( \forall v \in LStore(S), \alpha_{i+1}(v) = \alpha_i(v) \) and
    
    - \( \mu = p(u_1, \ldots, u_n; v_1, \ldots, v_m, CE) \) and \( \mu_i = p(\alpha_i(u_1), \ldots, \alpha_i(u_n); \alpha_{i+1}(v_1), \ldots, \alpha_{i+1}(v_m), CE) \) then
      
      * if there is no \((c, E) \in CE\) s.t. \( \alpha_i(c) \equiv true \) (or there is more than one such \((c, E)\) then \( I_{i+1} = I_i \) and \( \forall v \in LStore(S), \alpha_{i+1}(v) = \alpha_i(v) \), or
        
        * let \((c, E)\) be the unique conditional effects in \( CE \) s.t. \( \alpha_i(c) \equiv true \), and let \((es, ev)\) be a non-deterministicall chosen element of \( E \), then
          
          - for each statement insert \( R(t_1, \ldots, t_k, s_1, \ldots, s_l) \), delete \( R(t_1, \ldots, t_k) \), or modify \( R(t_1, \ldots, t_k, s_1, \ldots, s_l) \) in \( es \), apply the corresponding modification obtained by replacing \( t_i \) (respectively, \( s_i) \) by \( \alpha_i(t_i) \) (respectively, \( \alpha_i(s_i) \) on the instance \( I_i \). The obtained instance is the database \( I_{i+1} \).
\[ \forall v_j := t \in ev, \alpha_{i+1}(v_j) = \alpha_i(t) \text{ and } \alpha_{i+1}(v) = \alpha_i(v) \text{ for all } \text{ other variables } v \text{ of } LStore(S). \]

A execution of a service \( S \) starts at an initial configuration \( id_0 = (l_0, I_0, \alpha_0) \), with \( l_0 \) the initial control state of \( GA(S) \), \( I_0 \) an arbitrary database over \( \mathcal{W} \) and \( \alpha_0(x) = \omega, \forall x \in LStore(S) \). Then, a service moves from an \( id_i \) to \( id_j \) according to the mechanics defined by the set of transitions of \( GA(S) \). If \( id_i \xrightarrow{\mu_i} id_j \) satisfies the conduction condition above, we say that \( \mu_i \) is allowed from \( id_i \).

\[
\text{Fig. 2. Example of an initial configuration } id_0 = (l_0, I_0, \alpha_0).
\]

**Example 4.** We illustrate in this example a run of our sample Warehouse service \( S_1 \) depicted at figure [1]. Figure [2] shows a possible initial configuration of the Warehouse service \( S_1 \). This configuration is made of: (i) the initial state \( q_0 \) of the guarded automaton of \( S_1 \), (ii) an initial world database over the relation schemas Inventory, Customers and Orders, and (iii) the local store \( LStore(S_1) \) having all its variables set to \( \omega \) (i.e., the variables are initially undefined). Upon the reception of the message requestOrder(cust := '1', payBy := ‘cc’, item := “HP15”, addr := “NW”) the service \( S_1 \) moves from configuration \( id_0 = (l_0, I_0, \alpha_0) \) to the configuration \( id_1 = (l_1, I_1, \alpha_1) \) depicted at figure [3]. Note that at configuration \( id_1 \), the world database is left unchanged while the values conveyed by the message requestOrder are stored in the corresponding variables in \( LStore(S_1) \). Then, upon the execution of the atomic process CheckItem, the service moves from configuration \( id_1 \) to the configuration \( id_2 = (l_2, I_2, \alpha_2) \) depicted at figure [4]. As explained in the previous example, the atomic process CheckItem (c.f., figure [1](b)), takes as input parameter the variable \( \text{item} \) whose value at configuration \( id_1 \) is \( \alpha_1(\text{item}) = \text{"HP15"} \). Hence, the condition (if \( f^1_{\text{Inventory}}(\text{item}) = T \) in the specification of the effects of the CheckItem process is evaluated to true. Therefore, the output parameters \( \text{avail} \), \( \text{wh} \) and \( \text{price} \) of the CheckItem process are updated as follows: \( \text{avail} := T \), \( \text{wh} := f^2_{\text{Inventory}}(\text{"HP15"}) = \text{"NW"} \) and \( \text{price} := f^3_{\text{Inventory}}(\text{"HP15"}) = \text{"65"} \). Moreover, the condition (if \( \text{price} > 50 \) and \( f^4_{\text{Customers}}(\text{cust}) \neq \omega \)) is also evaluated to true.
at configuration \(id_1\). Hence, a new order id is generated (e.g., the order L021) and inserted in the relation Orders.

![Fig. 3. The configuration \(id_1 = (l_1, I_1, \alpha_1)\) after reception of the message requestOrder.](image)

![Fig. 4. The configuration \(id_2 = (l_2, I_2, \alpha_2)\) after the execution of the checkItem process.](image)

### 2.4 extended state machine

The semantics of a Colombo service can be captured by the following notion of an extended infinite state machine.

**Definition 3.** (extended state machine) Let \(GA(S) = (Q, \delta, l_0, F, LStore(S))\) be a guarded automata of a service \(S\). The associated infinite state machine, noted \(E(S)\), is a tuple \(E(S) = (Q, \mathcal{Q}_0, \mathcal{F}, \Delta)\) where:
- \( Q = \{(l, I, \alpha)\} \) with \( l \in Q \), \( I \) a database over \( W \) and \( \alpha \) a valuation over the variables of \( LStore \). The set \( Q \) contains all the possible configurations of \( E(S) \).

- \( Q_0 = \{(l_0, I_0, \alpha_0)\} \), with \( I_0 \) an arbitrary database over \( W \) and \( \alpha_0(x) = \omega \), \( \forall x \in LStore(S) \). \( Q_0 \) is the infinite set of initial configurations of \( E(S) \).

- \( F = \{(l_f, I, \alpha) \mid l_f \in F\} \). \( F \) is the set of final configurations of \( E(S) \).

- \( \Delta \) is an (infinite) set of transitions of the form \( \tau = (l_i, I_i, \alpha_i) \xrightarrow{\mu} (l_j, I_j, \alpha_j) \) such that \( \mu \) is allowed from \( (l_i, I_i, \alpha_i) \) (i.e., \( \tau \) satisfies the consecution condition of definition 3).

Any configuration of the extended state machine belongs in a path from an initial configuration to a final configuration. A run of \( E(S) \) is any finite path from an initial configuration of \( E(S) \) to one of its final configurations. Given an initial configuration \( id_0 \) of \( E(S) \), all the possible runs of \( E(S) \) starting from \( id_0 \) form an (infinite) execution tree having \( id_0 \) as its root. Hence, due to the infinite number of initial databases, all the runs of service \( S \) are captured in an (infinite) forest, that contains all possible execution trees of \( E(S) \) (i.e., the set of trees having as a root an initial configuration \( id \) with \( id \in Q_0 \)).

### 2.5 Simulation relation

We define now the notion of simulation between two Colombo services.

**Definition 4. (Simulation)** Let \( S \) and \( S' \) be two Colombo services and let \( E(S) = (Q, Q_0, F, \Delta) \) and \( E(S') = (Q', Q'_0, F', \Delta') \) be respectively their associated extended state machines.

- Let \( (id, id') \in Q \times Q' \). The configuration \( id = (l, I, \alpha) \) is simulated by \( id' = (l', I', \alpha') \), noted \( id \preceq id' \), iff:
  - \( I = I' \), and
  - \( \forall id \xrightarrow{\mu} id_j \in \Delta, \) there exists \( id' \xrightarrow{\mu'} id'_j \in \Delta' \) such that \( \mu = \mu' \) and \( id_j \preceq id'_j \)
- The extended state machine \( E(S) \) is simulated by the extended state machine \( E(S') \), noted \( E(S) \preceq E(S') \), iff \( \forall id_0 \in Q_0, \exists id'_0 \in Q'_0 \) such that \( id_0 \preceq id'_0 \)
- A Colombo service \( S \) is simulated by a Colombo service \( S' \), noted \( S \preceq S' \), iff \( E(S) \preceq E(S') \).

Informally, if \( S \preceq S' \), this means that \( S' \) is able to faithfully reproduce the external visible behavior of \( S \). The external visible behavior of a service is defined here with respect to the content of the world database as well as the exchanged concrete messages (i.e., message name together with the values of the message parameters). The existence of a simulation relation ensures that each execution tree of \( S \) is also an execution tree of \( S' \) (in fact, a subtree of \( S' \)), modulo a relabeling of control states.
Example 5. Consider the Colombo services $S_2$ and $S_3$ depicted at figure 5. We assume that these services use the same world database schema as the service $S_1$ of figure 1. An interesting question is to compare the three services with respect to their external visible behaviours. For example, although the automata of the services $S_1$ and $S_2$ look different, service $S_1$ is in fact simulated by service $S_2$ (i.e., $S_1 \preceq S_2$) which means that any behaviour of $S_1$ can be reproduced by $S_2$.

On another side, even if service $S_3$ looks more general than $S_1$, the two services are in fact not comparable w.r.t. simulation relation (i.e., $S_1 \not\preceq S_3$ and $S_3 \not\preceq S_1$). One can see that $S_1$ does not simulate $S_3$ because $S_3$ allows the PREPAID payment mode for any item while $S_1$ accepts the PREPAID payment mode only for items having a price less than 100. The service $S_3$ does not simulate $S_1$ because if a payment by credit card (payment mode CC) is approved, the service $S_1$ sends a message $\text{!requestShip}(wh,addr)$ before terminating the execution while service $S_3$ never sends such a message.

3 Undecidability of simulation in unbounded Colombo

We shall show that the simulation problem is undecidable for Colombo services.

Problem 1. Let $S$ and $S'$ be two Colombo services. The simulation problem, noted $\text{CheckSim}(S,S')$, is the problem of deciding whether $S \preceq S'$. 

We start by establishing a connection between the problems of state reachability and checking simulation between services. We exploit then this connection to establish undecidability of simulation.

Let us first define the state reachability problem for Colombo services.

**Problem 2.** Let $S$ be a Colombo service and $E(S) = (Q, Q_0, F, \Delta)$ its extended state machine. Let $l \in Q$ be a control state in $GA(S)$. The reachability problem, noted $reach(E(S), l)$, is the following: Is there a database $J$ over the scheme $W$ and a valuation $\alpha$ over $LStore(S)$ such that the configuration $(l, J, \alpha)$ appears in a run of $E(S)$?

**Example 6.** An example of a reachability problem is to ask whether the configuration $id_2 = (l_2, I_2, \alpha_2)$ depicted at figure 4 is reachable by our sample Warehouse service $S_1$. The answer in this case is yes since, as illustrated in the previous example, the configuration $id_2$ can be reached from the initial configuration $id_0$ shown at figure 2.

We exhibit the following straightforward link between simulation and reachability.

**Theorem 1.** If the reachability problem for a given class of Colombo service is undecidable so is simulation in that class.

**Proof.** (sketch)

Let $S$ be a Colombo service and $l$ be a state in $GA(S)$. W.l.o.g., we assume that for any transition $(l', c, \mu, l)$ of $GA(S)$, the label $\mu$ is unique (i.e., $\mu$ do not appear in any another transition of $GA(S)$). Then, given the reachability problem $reach(E(S), l)$, we build a new service $S'$, such that $GA(S')$ is obtained from $GA(S)$ by deleting the state $l$. Consider now the simulation problem $CheckSim(S, S')$. Hence in this case, it is easy to prove that $S \preceq S'$ iff $l$ is not reachable in $E(S)$.

Let us consider now the reachability problem in Colombo.

**Lemma 1.** The reachability problem in Colombo is undecidable.

The proof of this lemma is achieved by a reduction from halting problem of a Minsky machine [44]. A Minsky machine $M$ consists of two nonnegative counters, $\text{cpt}_1$ and $\text{cpt}_2$, and a sequence of labelled instructions:

$L_0 : \text{instr}_0; L_1 : \text{instr}_1; \ldots ; L_{n-1} : \text{instr}_{n-1}; L_n : \text{halt}$

where each of the first $n$ instructions has one of the following forms:

1. $L_i : \text{cpt}_k := \text{cpt}_k + 1; \text{goto } L_j$, or
2. $L_i : \text{if } \text{cpt}_k = 0 \text{ then goto } L_j \text{ else } \text{cpt}_k := \text{cpt}_k - 1; \text{goto } L_l$.

with $k \in \{1, 2\}$, $i \in [0, n-1]$ and $j, l \in [0, n]$. 

12
A machine $M$ starts its execution with counters $cpt_1 = cpt_2 = 0$ and the control at label $L_0$. Then, when the control is at a label $L_i$, $i \in [0, n-1]$, the machine executes the instruction $instr_i$ and jump to the appropriate label as specified in this instruction. The machine $M$ halts if the control reaches the halt instruction at label $L_n$.

**Example 7.** Figure 6(a) shows an example of a Minsky machine $M_1$ which computes the difference operation $5 - 2$ at the counter $cpt_2$. The seven first lines $L_0$ to $L_6$ of $M_1$ are used to initialize the counters $cpt_1 := 2$ and $cpt_2 = 5$. Then, the machine $M_1$ loops on the lines $L_7$ and $L_8$ to compute the difference $cpt_2 - cpt_1$ and halts. Figure 6(b) shows a Minsky machine $M_2$ that never halts. An execution of such a machine leads to an infinite sequence: $(L_0, cpt_1 = 0, cpt_2 = 0), (L_1, cpt_1 = 1, cpt_2 = 0), (L_0, cpt_1 = 1, cpt_2 = 1), \ldots$

It is known that the halting problem of Minsky machines, i.e., whether the execution of a given machine halts, is undecidable even in the case when the two counters are initialized to zero.

Given a Minsky machine $M$, we construct a Colombo service $S_M$ that captures the execution of $M$. $S_M$ uses a world database schema containing a single binary relation schema (i.e., $W = \{R(A; B)\}$). The main idea to simulate a machine $M$ is to make $S_M$ working only on parts of instances of $R$ that form a *chain* having the constant 0 as a root. A chain of length $k$ is any set $T_k = \{(c_0, c_1), \ldots, (c_{k-1}, c_k)\}$ where $i \in [0, k-1], c_i$ is a constant. The constant $c_0$ is called the root of $T_k$. For a pair $(c_{i-1}, c_i) \in T_k$, we note by $d(c_i) = l$ the distance of $c_i$ with respect to the root $c_0$ in the chain $T_k$. An instance $I$ of $R$ is said $k$-standard if there exists a chain $T_k$ such that $T_k \subseteq I$ and $c_0 = 0$. Hence, a $k$-standard instance contains a chain of length $k$ that starts with pair $(0, c_1)$.

**Example 8.** Figures 10(a) and (b) show examples of two 3-standard databases. Each of the relations of these figures contains a chain of length 3 starting from...
the root 0: the chain $\mathcal{T}_3 = \{(0, 1), (1, 2), (2, 3)\}$ of figure 10(a) and the chain $\mathcal{T}_3 = \{(0, a), (a, f), (f, k)\}$ of figure 10(b). Note that, the relation $R$ of figure 10(b) contains two additional tuples $(l, 10)$ and $(10, 20)$ that do not belong to the chain $\mathcal{T}_3 = \{(0, a), (a, f), (f, k)\}$. These two tuples will never be accessed by the constructed Colombo services (i.e., the constructed Colombo services can see only the elements of a chain rooted at the constant 0). Figures 10(c) and (d) show examples of non-standard databases. The database at figure 10(c) is non-standard because it does not contain the constant 0 while the database at figure 10(d) is non-standard because it includes a chain with a cycle (i.e., the chain $\mathcal{T}_3 = \{(0, a), (a, b), (b, c), (c, b)\}$).

To simulate the counters $\text{cpt}_1$ and $\text{cpt}_2$ during an execution of $M$, $S_M$ uses respectively two variables, namely $x_1$ and $x_2$ (hereafter called counter variables), of its LStore. The variables $x_1$ and $x_2$ are initially set to 0. Intuitively, a value of a counter $\text{cpt}_j$, with $j \in \{1, 2\}$, is captured by the distance between the current value of the variable $x_j$ w.r.t. to the root 0 of the chain (i.e., $\text{cpt}_j = d(x_j)$). Hence, a given counter $\text{cpt}_j$ of a minsky machine $M$ is equal to 0 iff its corresponding counter variable $x_j$ is equal to 0 (with $j \in \{1, 2\}$). Incrementing a counter $\text{cpt}_j$ is captured in $S_M$ by moving forward the corresponding variable $x_j$ in the chain $\mathcal{T}_k$ while decreasing a counter amounts to moving $x_k$ backward in the chain.

**Example 9.** Figure 8 shows some configurations of a Colombo service $S_{M1}$ used to simulate the Minsky machine $M1$ of figure 6(a). The local store of $S_{M1}$ includes among others the variables $x_1$ and $x_2$ which are respectively used to simulate the counters $\text{cpt}_1$ and $\text{cpt}_2$ of $M1$. The initial state of $M1$, i.e., $\text{cpt}_1 = \text{cpt}_2 = 0$, corresponds to the configuration of $S_{M1}$ depicted at figure 8(a). In this configuration, both $x_1$ and $x_2$ are set to 0. Figure 8(b) shows the configuration of $S_{M1}$.
after the incrementation of the counter \( \text{cpt}_1 \) of \( M_1 \) while figure 8 (c) shows a configuration of \( S_{M_1} \) corresponding to a state of \( M_1 \) where \( \text{cpt}_1 = 1 \) and \( \text{cpt}_2 = 5 \).

Moreover, to be able to simulate correctly an execution of a Minsky machine \( M \), a service \( S_M \) requires an input database which is at least \( k_{\text{max}} \)-standard where \( k_{\text{max}} \) is the maximum value reached by the counters \( \text{cpt}_1 \) and \( \text{cpt}_2 \) of \( M \) in the considered execution. Hence, during its execution a service \( S_M \) needs to continuously check that the current database is \( k_{\text{max}} \)-standard. Due to the limited expressivity of the Colombo model, the implementation of such verification operations as well as the incrementation and decrementation of the counter variables \( x_1 \) and \( x_2 \) are not straightforward. We explain below in more details how the service \( S_M \) is constructed.

Let \( M \) be a Minsky machine defined as above. We associate to \( M \), a Colombo service \( S_M \), called the corresponding service of \( M \), with the guarded automata \( GA(S_M) = (Q, \delta, q_{\text{start}}, F, L\text{Store}(S)) \). The set of states \( Q \) contains among other states, a state \( q_i \) for each label \( L_i \) in \( M \), with \( i \in [1, n-1] \), the initial state \( q_{\text{start}} \) and two final states \( q_{\text{fail}} \) and \( q_{\text{halt}} \). The state \( q_{\text{halt}} \) corresponds to the label \( L_n \) of the halt instruction of \( M \). An execution of \( S_M \) ends at the final state \( q_{\text{halt}} \) if the corresponding Minsky machine execution halts. An execution of \( S_M \) reaches the final state \( q_{\text{fail}} \) every time it is given as input an initial database which is not \( k_{\text{max}} \)-standard. To achieve this task, the service \( S_M \) uses a boolean variable noted \( x_{\text{flag}} \) to control the conformity of the current database: \( x_{\text{flag}} \) is initialized to \( \text{true} \) and then it is set to \( \text{false} \) if during a given execution the service finds

![Fig. 8. Examples of configurations of a service \( S_{M_1} \) which simulates the Minsky Machine \( M_1 \).](image)
out that the current database is not \( k_{\text{max}} \)-standard. Setting the boolean variable \( x_{\text{flag}} \) to \text{false}, will make the execution moving to the final state \( q_{\text{fail}} \).

![Diagram](https://example.com/diagram.png)

**Fig. 9.** Sub-processes of \( S_M \).

Fig. 9 shows fragment of a Colombo service used to model the two kinds of instructions used by Minsky machines while Fig. 10 describes the associated atomic processes. Fig. 9 (a) depicts the initialisation of a service \( S_M \). An execution of such a service starts by executing the atomic process \( \text{init} \) and moves to the state \( q_{\text{temp}} \). The \( \text{init} \) process checks that the initial database is 1-standard (i.e., it contains a tuple \((0, c_1)\)) and in this case sets the counter variables to 0 and the boolean variable \( x_{\text{flag}} \) to \text{true}. In case the initial database is not 1-standard, the variable \( x_{\text{flag}} \) is set to \text{false} which will make the execution moving from state \( q_{\text{temp}} \) to the final state \( q_{\text{fail}} \).

**Example 10.** Consider again a Colombo service \( S_{M_1} \) which simulates the Minsky Machine \( M_1 \). By construction, the guarded automaton of such a service includes the initialisation part depicted at figure 9(a). Therefore, if the service \( S_{M_1} \) is given as initial database the non 1-standard database of figure (c), the service starts by executing the atomic process \( \text{Init} \) of figure 10 and moves to the state \( q_{\text{temp}} \). As an effect of the execution of the atomic process \( \text{Init} \), the variable \( V_{\text{flag}} \) is set...
Effects: Output: \( V \)
Input: \( V \)

\[ f(V) = 0 \]

Else
\[ V_{temp1} := 0; \]
\[ V_{temp1} := 0; \]
\[ \ldots := V_{temp1}; \]
\[ V_{check} := true; \]

\[ \text{Else} \]
\[ V_{temp2} := V_{temp1}; \]

Copy
Input: \( V_{temp3} \)
Output: \( V_{temp2} \)

Effects:
\[ V_{temp2} := V_{temp3}; \]

\textbf{Fig. 10.} Atomic processes of the Colombo service \( S_M \).

to false during this transition. Indeed, when evaluated over the non 1-standard database of (10c), the condition \( (f^R_1(0) \neq \omega) \) of the Init process returns false and hence the effect \( V_{flag} := \text{false} \) specified in the Else branch is applied. At state \( q_{temp} \), the only possible transition for service \( S_M \), is to move to the final state \( q_{fail} \) and terminate the execution. Hence, when given any non 1-standard initial database, the service \( S_M \) always terminates at state \( q_{fail} \) (and can never reach the state \( q_{halt} \)).

\textbf{Fig. 11(b)} depicts part of a service that implements Minsky machine instructions of type 1: \( L_1 : cpl_k := cpl_k + 1; \text{goto} L_j \) (i.e., incrementation of a counter \( cpl_k \), with \( k \in \{1, 2\} \)). As explained above, incrementation amounts to moving forward in the chain the corresponding counter variable \( x_k \). Assume that the current value of the variable \( x_k \) is \( x_k = c_l \), with \( c_l \) a constant. The incrementation of \( x_k \) requires to: (i) first check that \( f^R_1(x_k) \neq \omega \) (i.e., the chain is long enough to handle the new value of the counter), and (ii) check that \( f^R_1(c_l) \) is a new value which has not already appeared in the chain. These two conditions ensure that the considered database is \( k \)-standard (with \( k = d(c_l)+1 \)). The first condition is easy to check (c.f., atomic process \textbf{INCr}) while the second one is handled by reading the chain starting from the root until the tuple \((c_{l-1}, c_l)\) and checking at each step whether the value \( f^R_1(c_l) \) has already appeared or not. To achieve this task, an execution of \( S_M \) enters the state \textit{checkL}_j and then recursively calls the atomic process \textbf{CheckValue} starting from the root \((0, c_l)\) of the chain (c.f., loop between the states \textit{CheckL}_j and \textit{LoopL}_j in \textbf{Fig. 11(b)}). The execution exits from the loop in two cases: (i) either it reaches to tuple \((c_{l-1}, c_l)\), which means that the current database is \( k \)-standard (with \( k = d(c_l)+1 \)) and hence the service moves to the state \( q_{L_j} \) and continue the execution, or (ii) it reaches a tuple...
\((c_i, f^R_i(c_i))\) in the chain which means that the database is not \(k\)-standard (with \(k = d(c_i) + 1\)) and hence the service moves to the final state \(q_{fail}\).

**Example 11.** Let us illustrate the incrementation of a counter on the non-standard database of figure 10(d). Consider the state of the Minsky machine \(M1\) of figure 9 after the execution of the lines L0 to L4: the current values of the counters are \(cpt_1 = 2\) and \(cpt_2 = 3\) and the current line is L5. This state corresponds to a configuration of the \(S_{M1}\) service with a current state \(q_{L5}\) and the counter variables having as values: \(x_1 = b\) and \(x_2 = c\). Note that, in the considered database, the distance of the constant \(b\) to the root is equal to 2 while the distance of \(c\) to the root is equal to 3 (i.e., \(d(b) = 2\) and \(d(c) = 3\)). Hence, such a configuration corresponds to a state of the Minsky machine \(M1\) with the counter \(cpt_1\) equal to 2 and the counter \(cpt_2\) equal to 3. The line L5 of \(M1\) increments \(cpt_2\) and moves to line L6. Let us see how such an incrementation is implemented by the Colombo service \(S_{M1}\). Following the automaton of figure 9(b), \(S_{M1}\) calls the atomic process \(\text{INCr}\) and moves from state \(q_{L5}\) to state \(\text{check}_{L6}\). The execution of the atomic process \(\text{INCr}\) checks that the current chain is long enough to handle the new value of the counter. This is the case in the considered database since the condition \((f^R_1(x_2) \neq \omega) \land (f^R_1(x_2) \neq 0)\) evaluates to \(\text{true}\) over the non-standard database of figure 10(d) (indeed, we have \((f^R_1(x_2) = f^R_1(c) = b)\). But before assigning the constant \(b\) to the variable \(x_2\), the service \(S_{M1}\) enters the state \(\text{check}_{L6}\) and checks whether or not \(b\) is a new constant in the chain (i.e., \(b\) does not already appear in the chain). This verification is achieved by iterating on the chain from the root 0 to the current value \(x_2\) (i.e., the constant \(c\)) and checking at each iteration that the constant \(b\) do not belong to the chain (loop between the states \(\text{check}_{L6}\) and \(\text{loop}_{L4}\) and call to the atomic process \(\text{CheckValue}\) in the automaton of figure 9(b)). In the considered database, the first iteration reads the tuple \((0, a)\) of the chain while the second iteration reads the tuple \((a, b)\). The service \(S_{M1}\) is then able to detect that there is cycle in the chain because the constant \(b\) appears twice and hence the considered database is not standard. Hence, the service will move to state \(q_{fail}\) and terminates the execution.

We consider now the implementation of instructions of type 2:

\[ L_1 : \text{if} \ cpt_k = 0 \ \text{then goto} \ L_3 \ \text{else} \ cpt_k := \text{cpt}_k - 1 \ \text{then goto} \ L_4 \ \text{(c.f., Fig.9(c))}. \]

The main difficulty here lies in the implementation of the decrementation operation (which amounts to moving back the counter \(x_k\) in the chain). Assume that the current value of \(x_k\) is \(c_l\). Decrementing \(x_k\) amounts to assigning to \(x_k\) the constant \(c\) such that \(f^R_1(c) = c_l\). To find the constant \(c\) one needs to read again the chain starting from the root. In the service \(S_M\) this is implemented by first entering the state \(D_{kL_1}\), by executing the Init-Decr process, and then recursively calling the atomic process \(\text{DECR}\) (c.f., loop between the states \(D_{kL_1}\) and \(B_{kL_1}\) of Fig.9(c)) to explore the chain starting from the root and stopping at the tuple \((c, c_l)\) (we are sure that such a tuple exist because during the incrementation step to reach the value \(c_l\), the database has been checked to be at least \(d(c_l)\)-standard).
Example 12. Consider again a configuration of the $S_{M1}$ service with the database of figure 10(d) and the counter variable $x_2 = c$ (i.e., corresponding to the counter $cpt_2 = 3$). To decrement $x_2$, the service $S_{M1}$ reads the chain from the root 0 and stops the tuple $(b, c)$ (third tuple of the database) because we have $f_R^1(b) = c$ (and hence $d(b) = d(c) - 1 = 2$). The constant $b$ is then assigned as the new value for the variable $x_1$ (which corresponds to a counter $cpt_2 = 2$).

We give now the main property of the proposed construction that enables to prove lemma [1]

Lemma 2. Let $M$ be a Minsky machine and $S_M$ the corresponding Colombo service, then: $M$ halts iff $reach(E(S_M), q_{halt})$

This result is obtained from the connection that exists between executions of $M$ and the executions of $S_M$ that use as input a $k$-standard databases. In particular, the different values taken by the counter $cpt_1$ and $cpt_2$ during an execution of $M$ are captured by the distances of the counter variables $x_1$ and $x_2$ during the execution of $S_M$. Hence, it is possible to map any execution of $M$ into an execution of $S_M$ on a $k$-standard database and conversely. Moreover, it is easy to show that if there exists an execution of $M$ that halts and in which $k_{max}$ is the maximum value reached by the counters of $M$, then the execution of the corresponding service $S_M$ using a $k_{max}$-standard initial database terminates at the final state $q_{halt}$. On the other side, by construction, $S_M$ terminates at the final state $q_{halt}$ iff it takes as initial database a $k$-standard database (which hence can be mapped into an execution of $M$ that halts).

From theorem [1] and lemma [1] we obtain the following main result regarding simulation in the Colombo model.

Theorem 2. Let $S$ and $S'$ be two Colombo services, then $CheckSim(S, S')$ is undecidable.

Finally, the following theorem can be straightforwardly derived from the previous proof since the constructed service $S_M$ is a non-communicating read-only Colombo service.

Theorem 3. Let $S$ and $S'$ be two non-communicating services with read-only accesses to the world database and let $l$ be a control state in $GA(S)$, then both $CheckSim(S, S')$ and $reach(E(S), l)$ are undecidable.

4 Decidability of simulation in DB-less Colombo

We investigate in this section the simulation problem in the setting of a Colombo model without a global database (i.e., we assume the world schema $W = \emptyset$). We denote such a model $Colombo_{\emptyset}$. Before going into details of this study, it is worth mentioning the following observation:
While we consider services without global databases, the corresponding specifications are still infinite-state systems since the variables are ranging over an infinite domain. More precisely, the value of a variable is taken from an infinite domain when the service performs a reception of message.

Let $S$ be a Colombo$^{db=0}$ service. The associated state machine is a tuple $E(S) = (Q, Q_0, P, \Delta)$. A configuration of $E(S)$ has the form $id = (l, \emptyset, \alpha)$ while there is only one initial configuration $id_0 = (l_0, \emptyset, \alpha_0)$ with $\alpha_0(x) = \omega, \forall x \in LStore(S)$. Moreover, in Colombo$^{db=0}$ services, atomic processes can only assign constants to variables of $LStore(S)$ or affect value of a variable to another. Note that $E(S)$ is still an infinite state system. This is due to the presence of input messages with parameters taking their values from a possibly infinite domain. We describe below a symbolization technique that allows to abstract from concrete values and hence turns extended machines associated with Colombo$^{db=0}$ services into finite state machines.

**Notation and basic notions.** Let $X$ be a set of variables taking their values from an infinite domain $\mathcal{D} \cup \{\omega\}$. Let $\theta$ be a condition on a set of variables $X$ and let $\alpha$ be a valuation over $X$. Than $\theta(\alpha)$ is the condition obtained by replacing each variable $x$ appearing in $\theta$ by $\alpha(x)$. We say that $\alpha$ satisfies $\theta$, noted $\alpha \models \theta$, if $\theta(\alpha) = true$. A valuation $\alpha$ satisfies a set $\Theta$ of conditions, noted $\alpha \models \Theta$, if $\alpha \models \theta, \forall \theta \in \Theta$.

Let $K = \{c_1, \ldots, c_k\}$ with $c_1 < \ldots < c_k$ be a set of constants in $\mathcal{D}$. We define the set $I_K$ of elementary intervals over $K$ as $I_K = \{[\omega, \omega], -\infty, c_1[, c_k, +\infty[) \cup \{[c_l, c_l], l \in [1, k]\} \cup \{[c_l, c_{l+1}], l \in [1, k-1]\}$. Note that, a set of intervals $I_K$ forms a partition of the domain $\mathcal{D} \cup \{\omega\}$ (i.e., intervals in $I_K$ are pairwise disjoint).

**Example 13.** For $K = \{4, 10\}$, the associated set of elementary intervals is $I_K = \{[\omega, \omega], -\infty, 4[, 4, 10[, 10, +\infty[\}$

Let $X$ be a set of variables and $op \in \{=, <\}$. We denote by $\psi$ a set of conditions defined as follows: $\forall x, y \in X$, we have $\{x = \omega, y = \omega, x op y\} \cap \psi \neq \emptyset$. $\psi$ is called a $v$-order over $X$. A $v$-order $\psi$ is said consistent if it exists at least one valuation $\alpha$ over the variables of $X$ such that $\alpha \models \psi$. We note by $vo(X)$ the set of all $v$-orders on $X$.

**Example 14.** Let $X = \{x, y\}$, then examples of $v$-orders over $X$ are:

- $\psi_0 = \{x = \omega, y = \omega\}$
- $\psi_1 = \{x = y\}$
- $\psi_2 = \{x = \omega, y = y\}$
- $\psi_3 = \{x = \omega, y < y\}$ (inconsistent)
- $\psi_4 = \{x < y, y < x\}$ (inconsistent)

The $v$-orders $\psi_3$ and $\psi_4$ are indeed inconsistent while the other $v$-orders are consistent.

We use below the notion of regions to extend intervals to a set of variables.
Definition 5. (Regions) Let $X = \{x_1, \ldots, x_n\}$ be a set of variables and $K$ a set of constants. We assume variables in $X$ ordered according to the lexicographic order. A region of $X$ w.r.t $K$ is a tuple $r = (\tau_{x_1}, \ldots, \tau_{x_n}, \psi)$ with $\psi \in \text{vo}(X)$ and $\tau_{x_i} \in I_K, \forall i \in [1, n].$

The set of all possible regions of $X$ w.r.t. $K$ is denoted $R_g(X, K).$

Hence a region $r = (\tau_{x_1}, \ldots, \tau_{x_n}, \psi)$ associates an elementary interval $\tau_{x_i}$, with each variable $x_i \in X.$

Example 15. Let us consider the set of constants $K = \{4, 10\}$ and the set of variables $X = \{x, y\}$ of the previous examples, with their associated elementary intervals and v-orders. The set $R_g(X, K)$ includes the following regions:

- $r_\omega = ([\omega, \omega], [\omega, \omega], \psi_0)$
- $r_1 = ([4, 10], [4, 10], \psi_1)$
- $r_2 = ([\omega, \omega], -\infty, 4[\psi_2)$
- $r_3 = ([10, 10], [\omega, \omega], \psi_2)$

In the sequel, we abuse of notation and write $r \land \theta$ instead of $(\tau_{x_1}, \ldots, \tau_{x_n}, \psi \land \theta)$ where $\theta$ is a condition. We introduce below some notation regarding regions.

- A valuation $\alpha$ over $X$ belongs to a region $r = (\tau_{x_1}, \ldots, \tau_{x_n}, \psi)$ of $X$, denoted $\alpha \in r$, if $\alpha(x_i) \in \tau_{x_i}, \forall i \in [1, n]$ and $\alpha \models \psi.$ The set of valuations that belong to a region $r$ is noted $\text{val}(r).$
- A region $r$ is inconsistent, noted $r \models \bot,$ if $\text{val}(r) = \emptyset.$ In the previous example, the region $r_3$ is inconsistent.
- Let $r = (\tau_{x_1}, \ldots, \tau_{x_n}, \psi)$ be a region of $X$. A projection of $r$ on a set $\{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$, noted $\pi_{\{x_{i_1}, \ldots, x_{i_k}\}}(r)$, is the region $\pi_{\{x_{i_1}, \ldots, x_{i_k}\}}(r) = (\tau_{x_{i_1}}, \ldots, \tau_{x_{i_k}}, \psi|_{x_{i_1}, \ldots, x_{i_k}})$, where $\psi|_{x_{i_1}, \ldots, x_{i_k}}$ is the subset of $\psi$ that contains only the conditions over $\{x_{i_1}, \ldots, x_{i_k}\}$.
- Let $r = (\tau_{x_1}, \ldots, \tau_{x_n}, \psi)$ and $r' = (\tau_{x_1}, \ldots, \tau_{x_n}, \psi')$ be two regions of $X$. We say that $r$ coincides with $r'$ on a set of variables $\{x_1, \ldots, x_k\} \subseteq X$, noted $r \equiv_{\{x_1, \ldots, x_k\}} r'$, if $\pi_{\{x_1, \ldots, x_k\}}(r) = \pi_{\{x_1, \ldots, x_k\}}(r').$

In the sequel, w.o.l.g., we assume that the set $R_g(X, K)$ contains only consistent regions. Note that, if $X$ and $K$ are both finite sets so $R_g(X, K)$ is also a finite set.

Lemma 3. Let $X$ be a finite set of variables and $K$ a finite set of constants in $\Theta$. Let $r \in R_g(X, K)$, then: $\forall \alpha_1, \alpha_2 \in r, \forall \Theta' \subseteq \Theta, \text{if } \alpha_1 \models \Theta' \text{ then } \alpha_2 \models \Theta'.$

Proof. Suppose $\alpha_1 \models \Theta'$ with $\Theta' = \theta_1 \land \theta_2 \ldots \theta_k$. It is sufficient to prove the property for a condition $\theta \in \Theta'$. Let $\psi$ be the v-order of $r$. We distinguish 4 cases:

1. $\theta$ is the condition $x = y$, then $\alpha_1 \models x = y$ implies $(x = y) \in \psi$, otherwise $r$ is not consistent (by construction of $r$). Thus $\alpha_2 \models (x = y)$ since $\alpha_2 \models \psi$.
2. $\theta$ is the condition $x > y$. Similar to case 1.
3. \( \theta \) is the condition \( x = c \), with \( c \in K \), then \( \alpha_1 \vdash x = c \) implies \( \tau_x = [c, c] \) in \( r \) (by construction of the elementary intervals). Thus \( \alpha_2 \vdash (x = c) \) since \( \alpha_2 \in r \).

4. \( \theta \) is the condition \( x > c \), with \( c \in K \), then \( \alpha_1 \vdash x > c \) implies that \( \forall c' \in \tau_x \), \( c' > c \) (by construction of the elementary intervals). Thus \( \alpha_2 \vdash (x > c) \) since \( \alpha_2(x) \in \tau_x \).

**Canonic representation of Colombo\(^{db=\emptyset} \) services.** Given a Colombo service \( S \), the main idea is to use the notion of regions to group together extended states of \( E(S) \). Interestingly, the obtained representation, called a Colombo region automaton (defined below), is a finite state machine. We define below such state machines and then we show how they can be used to test simulation between Colombo\(^{db=\emptyset} \) services.

W.l.o.g., we consider in the sequel only Colombo services with atomic processes having:

- disjoint input and output variables (i.e., services \( S \) that use atomic processes of the form \( p(u_1, \ldots, u_n; v_1, \ldots, v_m) \) with \( \{u_1, \ldots, u_n\} \cap \{v_1, \ldots, v_m\} \cap L\text{Store}(S) = \emptyset \), and
- a unique conditional effects \( (c, E) \) with \( E = \{(es, ev)\} \) s.t. \( es = \emptyset \) (since there are no modification on the database) and \( ev = \{v_i := t, \text{ with } t \text{ is either a constant or } \omega \text{ or an input variable}\} \).

**Definition 6. (Colombo\(^{db=\emptyset} \) region automata)** Let \( GA(S) = \langle Q, \delta, q_0, F, L\text{Store}(S) \rangle \) be a guarded automata of a Colombo\(^{db=\emptyset} \) service \( S \) with \( X = L\text{Store}(S) = \{x_1, \ldots, x_n\} \), and let \( \Theta \) be a set of atomic conditions in \( GA(S) \). Let \( K \) be a set of constants appearing in \( \Theta \). The associated Colombo\(^{db=\emptyset} \) region automaton is a finite state machine \( R^S = (Q^S, q_0^S, F^S, \delta^S, R_g(X,K)) \) defined as follows:

- \( Q^S \subseteq Q \times R_g(X,K) \), the set of states of \( R^S \),
- \( q_0^S = (q_0, r_\omega) \), the initial state, where \( r_\omega = ([w, w], [w, w], \{(x_i = \omega), i \in [1,n]\}) \).
- \( F^S \subseteq F \times R_g(X,K) \), the set of final states,
- Let \( r \in R_g(X,K) \). For each state \( (q, r) \) of \( R^S \) and for each transition \( (q, \theta, \mu, q') \in \delta \) such that \( r \land \theta \) is consistent then:
  
  (a) if \( \mu = \text{im}(v_1, \ldots, v_m) \), we have \( ((q, r), \mu, (q', r)) \in \delta^S \).
  
  (b) if \( \mu = \text{?im}(v_1, \ldots, v_m) \) we have for each \( r' \in R_g(X,K) \) which coincides with \( r \) on the variables \( L\text{Store}(S) \setminus \{v_1, \ldots, v_m\} \), we have \( ((q, r), \mu, (q', r')) \in \delta^S \).
  
  (c) If \( \mu = p(u_1, \ldots, u_n; v_1, \ldots, v_m, \{c, E\}) \), we have two cases:
    
    (c-1) if \( r \land \theta \land c \) is consistent then \( ((q, r), p(u_1, \ldots, u_n; v_1, \ldots, v_m), (q', r')) \in \delta^S \) where \( r' \) coincides with \( r \) on the variables \( L\text{Store}(S) \setminus \{v_1, \ldots, v_m\} \) and:
    
    for each \( i \in [1,m] \)
    
    * If \( v_i := c \in E \), then \( r' \) includes \( \tau_{v_i} = [c, c] \).
If \( v_i := u_j \in E \) then \( r' \) includes \( \tau_{v_i} = \tau_{u_j} \) and \( \psi' \) of \( r' \) includes \( v_i = u_j \).

* If \( v_i := \omega \in E \) then \( r' \) includes \( \tau_{v_i} = [\omega, \omega] \).

(c-2) If \( r \land \theta \land \neg c \) is consistent, we have

\[
((q, r), p(u_1, \ldots, u_n; v_1, \ldots, v_m), (q', r)) \in \delta^S.
\]

It is worth noting that a region automaton constructed according to definition 6 must be cleaned to remove states that are not included in a path from the initial state to a final state. We illustrate the construction of a region automaton on the simple Colombo service depicted at Fig.11.

![Fig. 11. A Colombo service S.](image)

**Example 16.** Service \( S \) of Fig.11 uses:

- A set of variables \( X = LStore(S) = \{x, y\} \),
- A set of conditions \( \Theta = \{(x > 5), (y > 5)\} \) used as guards in transitions or condition in the atomic process \( \text{Perm} \),
- A set \( K = \{5\} \) of constants that appear in \( \Theta \).

Hence, the set of elementary intervals over \( K \) is:

\[
I_K = \{[\omega, \omega], -\infty, 5[, [5, 5], ]5, +\infty]\}
\]

while the set \( R_\theta(X, K) \) includes, among others, the following regions:

- \( r_\omega = ([\omega], [\omega], \{x = \omega, y = \omega\}) \)
- \( r_1 = ([\omega], -\infty, 5[, \{x = \omega, y = y\}) \)
- \( r_2 = ([5, +\infty[, 5, +\infty[, \{y < x\}) \)
- \( r_3 = ([5, +\infty[, 5, +\infty[, \{y = x\}) \)
- \( r_4 = ([5, +\infty[, 5, +\infty[, \{x < y\}) \)

23
\[ r_5 = ([5, +\infty[, [5, 5], \{ y < x \} \]
\[ r_6 = ([5, +\infty[, -\infty, 5], \{ y < x \} \]
\[ \ldots \]

The corresponding region automaton \( R^S \) is depicted at Fig.12. The initial state of \( R^S \) is made of the pair \((q_0, r_\omega)\). We illustrate below the cases (a), (b) and (c) of definition 6 on this region automaton.

- the transition \((q_0, ?m1(x, y), q_1)\) of \( GA(S) \) (c.f., Fig.11), is translated into a set of transitions \(((q_0, r_\omega), ?m1(x, y), (q, r))\) with \( r \in R_g(X, K) \) (case (b) of definition 6). This captures the fact that on a reception of a message \( ?m1(x, y) \), any new values may be associated to the variables \( x \) and \( y \).
- the transition \((q_1, x > 5 | Perm(y; x), q_2)\) of \( GA(S) \), enables to create new transition from the state \((q_1, r_2)\) of \( R^S \) as illustrated below:
  - \(((q_1, r_2), Perm(y; x), (q_2, r_3))\), this is because the region \( r_2 \) satisfies both the guard \( x > 5 \) of the transition and the condition \( u_1 > 5 \) of the atomic process (case (c-1) of definition 6). Hence, in this case the atomic process \( Perm \) is executed. The atomic process \( Perm \) assigns variable \( y \) to the variable \( x \), hence the region automata moves to a region where \( \tau_x := \tau y \) and requires to have \( x = y \) in the associated v-order. In our example, region \( r_3 \) satisfies both conditions.
  - \(((q_1, r_5), Perm(y; x), (q_2, r_5))\), this is because the region \( r_5 \) satisfies the guard \( x > 5 \) of the transition but does not satisfy the condition \( u_1 > 5 \) of the atomic process \( Perm \) (case (c-2) of definition 6). According to the Colombo semantics, the transition is fired but the atomic process \( Perm \) execute a no-op operation (no operation). As a consequence, the region automata moves to state \( q_2 \) while staying in the same region \( r_5 \).
- the transition \(((q_2, r_3), !m2(x,), (q_3, r_5))\) (case (a) of definition 6). A send of a message does not modify values of the variables, hence upon sending the message \( !m2(x) \), the region automaton \( R^S \) moves into a new state \((q_3, r_5)\) while staying in the same region \( r_5 \).

In the following we show that the region automata \( R^S \) constitutes a compact representation of the extended state machine of \( E(S) \) and hence it faithfully abstracts the original Colombo service \( S \). To do so, we define the notion of unfolding of a region automaton \( U_{\text{fold}}(R^S) \) as given below.

**Definition 7.** (unfolding of region automata) Let \( R^S = (Q^S, q_0^S, F^S, \delta^S, R_g(X, K)) \) be a region automata of a service \( S \). The associated extended state machine, noted \( U_{\text{fold}}(R^S) \), is a tuple \( U_{\text{fold}}(R^S) = (Q^g, Q_0^g, F^g, \Delta^g) \) where:

- \( Q^g = \bigcup_{r \in R_g(X, K)} \{(q, \alpha) \ s.t \ (q, r) \in Q^S, \alpha \in r \} \)
- \( Q_0^g = \{(q_0, \alpha_\omega)\}, \text{ with } \alpha_\omega(x) = \omega, \forall x \in LStore(S) \)
- \( F^g = \bigcup_{r \in R_g(X, K)} \{(q, \alpha) \ s.t \ (q, r) \in F^S, \alpha \in r \} \).
∀(q,r) \xrightarrow{m_i} (q',r') \in \delta^S, a new transition \( (q,\alpha) \xrightarrow{m_i} (q',\alpha') \) is added to \( \Delta \) such that \( \alpha \in r, \alpha' \in r' \) and:

(a) if \( \mu = \ul m(v_1, \ldots, v_m) \), then \( \alpha' = \alpha \).

(b) if \( \mu = \ul m(v_1, \ldots, v_m) \) then \( \forall x \in L\text{Store}(S) \setminus \{v_1, \ldots, v_m\} \), we have \( \alpha'(x) = \alpha(x) \).

(c) If \( \mu = p(u_1, \ldots, u_n; v_1, \ldots, v_m, \{c, E\}) \), we have two cases:

(c-1) if \( r \land \theta \land c \) is consistent then \( \forall x \in L\text{Store}(S) \setminus \{v_1, \ldots, v_m\} \), we have \( \alpha'(x) = \alpha(x) \) and for each \( i \in [1,m] \), we have:

* If \( v_i := k \in E, \) with \( k \in D \cup \{\omega\} \), then \( \alpha'(v_i) = k \)
* If \( v_i := u_j \in E \) then \( \alpha'(v_i) = \alpha(u_i) \)

(c-2) if \( r \land \theta \land \neg c \) is consistent, then \( \alpha' = \alpha \).

A run of \( U_{\text{unf}}(RS) \) is any finite path from an initial configuration of \( E(RS) \) to one of its final configurations.

Example 17. Fig[13(b)] depicts part of the extended automata obtained by unfolding the region automata of Fig[13(a)] which corresponds to a fragment of the region automata of Fig[12].
The following lemma states that $R^S$ preserves the semantics of the original Colombo service $S$ in the sense that an unfolding of a region automaton coincides with the extended automaton of the original Colombo service.

**Lemma 4.** Let $E(S) = (Q, Q_0, F, Δ)$ and $R^S = (Q^S, Q_0^S, F^S, δ^S, R_g(X, K))$ with $X$ and $K$ defined as in definition 6. Then $E(S) = U_{nfold}(R^S)$.

**Proof.** It is sufficient to show that a transition $(q, θ, α) \xrightarrow{μ} (q', θ, α') ∈ Δ^g$ iff $(q, α) \xrightarrow{μ} (q', α') ∈ Δ^g$. By construction we have $(q_0, θ, α_0) ∈ Q_0$ and $q_0^S = (q_0, r_w)$. Now, take $(q, θ, α) \xrightarrow{μ} (q', θ, α') ∈ Δ^g$. Hence, there exists $(q, θ, μ, q') ∈ GA(S)$ s.t. $α = G_α$ by definition 6. Let $r, r' ∈ R_g(X, K)$ such that $α ∈ r$ and $α' ∈ r'$. We show that $(q, r) \xrightarrow{μ} (q', r') ∈ δ^S$ which implies that $(q, α) \xrightarrow{μ} (q', α') ∈ Δ^g$. We distinguish the following three cases:

1. $μ = ?m(c_1, . . . , c_n)$. We have $α = α'$ and therefore $(q, r) \xrightarrow{μ} (q', r) ∈ δ^S$ by definition 6.
2. $μ = ?m(c_1, . . . , c_n)$. By definition 6, we have:
   
   $α'(x) = \begin{cases} 
   α(x) & \text{if } x ∈ LStore(S) \setminus \{x_1, . . . , x_n\} \\
   c_i & \text{if } x = x_i \text{ with } i ∈ [1, n]
   \end{cases}$

   $⇒ r'$ coincides with $r$ on $LStore(S) \setminus \{x_1, . . . , x_n\}$

   Moreover, we have $r ⊨ θ$ since $α ⊨ θ$. So $(q, r) \xrightarrow{μ} (q', r') ∈ δ^S$.
3. $μ = p(α(u_1), . . . , α(u_n); α'(v_1), . . . , α'(v_m), \{c, E\})$. We consider two cases.
   
   (a) $α ⊨ r ∧ θ ∧ ¬c$: By lemma 6 all $α_1 ∈ r, α_1 ⊨ r ∧ θ ∧ ¬c$ and thus $r ⊨ r ∧ θ ∧ ¬c$. So $α = α'$ and therefore $(q, r) \xrightarrow{μ} (q', r) ∈ δ^S$ by Definition 6.

Fig. 13. Unfolding a region automaton.
(b) \( \alpha \models r \land \theta \land c \): By definition, we have
- \( \alpha (u_i) = \alpha(u_i) \)
- \( \alpha (v_i) = \alpha(u_i) \) or \( \alpha (v_i) = c \), where \( c \in K \).

So, \( r' \) coincides with \( r \) on \( LStore(S) \setminus \{v_1, \ldots, v_m\} \). Moreover \( \tau_{v_i} = \tau_{u_j} \) or \( \tau_{v_i} = [c, c] \). Thus, By definition, we have \( (q, r) \xrightarrow{\mu} (q', r') \in \delta^S \).

The other direction of the proof can be derived using a similar scheme.

### 4.1 Simulation between regions automata

In this section, we define a simulation relation between region automata and then we show how such a relation can be used to check simulation between two Colombo services.

**Definition 8. (Simulation of Colombo regions) automata** Let \( S \) and \( S' \) be two Colombo regions services, \( X = LStore(S) \), \( X' = LStore(S') \) and let \( \Theta_S \) (resp. \( \Theta_{S'} \)) be the set of atomic conditions used in \( GA(S) \) (resp. \( GA(S') \)). Let \( K \) be the set of all constants appearing in \( \Theta_S \cup \Theta_{S'} \) and let \( R^S = (Q^S, q_0^S, F^S, \delta^S, R_g(X, K)) \) and \( R^{S'} = (L^S, L^S, F^{S'}, \delta^{S'}, R_g(X', K)) \) be, respectively, the region automata associated with \( S \) and \( S' \).

- Let \( ((q, r_1), (l, r_2)) \in Q^S \times L^{S'} \) and \( \beta \) is a subset of the set of equalities of variables in \( S \) and \( S' \), i.e. \( \{ x = x' \ s.t. \ x \in X, x \in X' \} \). The configuration \( ((q, r_1), \beta) \) is simulated by \( ((l, r_2), \beta) \) noted \( ((q, r_1), \beta) \preceq_g ((l, r_2), \beta) \) iff:
  1. \( \forall (q, r_1) \xrightarrow{\mu} (q', r_1') \in \delta^S \), there exists \( (l, r_2) \xrightarrow{\mu} (l', r_2') \in \delta^{S'} \) such that
     1. if \( \mu = \lambda m(x_1, \ldots, x_n) \) then \( \mu' = \lambda m(y_1, \ldots, y_n) \) and \( r_1 \land r_2 \land \beta \Rightarrow x_i = y_i \), where \( i \in [1, n] \) and \( ((q', r_1'), \beta) \preceq_g ((l', r_2'), \beta) \).
     2. If \( \mu = \lambda m(x_1, \ldots, x_n) \) then \( \mu' = \lambda m(y_1, \ldots, y_n) \) and \( r_1 \land r_2 \land \beta' \) is consistent and \( ((q', r_1'), \beta') \preceq_g ((l', r_2'), \beta') \) where \( \beta' = \{ x_i = y_i, i \in [1, n] \} \cup \{ z = t \in \beta \ s.t. z \neq x_i, z \neq y_i, t \neq x_i, t \neq y_i \} \).
     3. If \( \mu = p(x_1, \ldots, x_n; y_1, \ldots, y_m, \{ c, E \}) \), then \( \mu' = p(x_1, \ldots, x_n; y_1, \ldots, y_m, \{ c, E \}) \) and \( r_1 \land r_2 \land \beta \Rightarrow x_i = u_i \) and
       * if \( r_1 \land c \) is consistent then \( r_1' \land r_2' \land \beta' \Rightarrow y_i = v_i \) and \( ((q', r_1'), \beta') \preceq_g ((l', r_2'), \beta') \) where \( \beta' = \{ z = t \in \beta \ s.t. z = v_i, t = v_i \} \).
       * if \( r_1 \land \neg c \) is consistent then \( ((q', r_1'), \beta') \preceq_g ((l', r_2'), \beta) \)
  2. \( R^S \preceq_g R^{S'} \) iff \( ((q_0, r_0), \emptyset) \preceq_g ((l_0, r_0), \emptyset) \)

The following lemma ensures that the relation \( \preceq_g \) captures correctly the simulation preorder on Colombo services.

**Lemma 5.** Let \( S \) and \( S' \) be two Colombo regions services and \( X = LStore(S) \cup LStore(S') \) and \( \Theta_S \) (resp. \( \Theta_{S'} \)) be the set of atomic conditions appearing in the guards of \( GA(S) \) (resp. \( GA(S') \)), and \( K \) be the sets of all constants appearing in \( \Theta_S \cup \Theta_{S'} \). Let \( R^S = (Q^S, q_0^S, F^S, \delta^S, R_g(X, K)) \) and \( R^{S'} = (L^S, L^S, F^{S'}, \delta^{S'}, R_g(X', K)) \), then:

\[ U_{nfold}(R^S) \preceq U_{nfold}(R^{S'}) \text{ iff } R^S \preceq_g R^{S'} \]
Proof. Let \( U_{nfold}(R^S) = (\mathcal{Q}^g, \mathcal{Q}_0^g, F^g, \Delta^g) \) and \( U_{nfold}(R^{S'}) = (\mathcal{Q}^g, \mathcal{Q}_0^g, F^g, \Delta^g) \).

\( \Leftarrow \) Assume that \( R^S \succeq_g R^{S'} \). Take \( \succeq = \{(q, \alpha_1), (l, \alpha_2)\} \) s.t. \((q, r_1), (l, r_2), \beta) \in \succeq_g\), with \( \alpha_1 \in r_1, \alpha_2 \in r_2 \) and \( (\alpha_1, \alpha_2) \models \beta \). We show that \( \succeq \) is a simulation relation (i.e., \( U_{nfold}(R^S) \preceq U_{nfold}(R^{S'}) \)).

Clearly \( (q, \alpha_w), (l, \alpha_w) \) \( \in \succeq \) since \((q, r_w), (l, r_w), \emptyset) \in \succeq_g\). Now, suppose that \((q, \alpha_1), (l, \alpha_2) \) \( \in \succeq \). We show that for any transition \((q, \alpha_1), \mu, (q', \alpha'_1)\) \( \in \Delta^g\), there exists a transition \((l, \alpha_2), \mu', (l', \alpha'_2)\) \( \in \Delta^g\) such that \((q', \alpha'_1), (l', \alpha'_2) \) \( \in \succeq \).

Let \((q, \alpha_1), \mu, (q', \alpha'_1)\) \( \in \Delta^g\). Then, by lemma\(^4\) there exists a transition \((q, r_1), \mu, (q', r'_1)\) \( \in \delta^g\) such that \( \alpha_1 \in r_1 \) and \( \alpha'_1 \in r'_1\).

By construction of \( \succeq \), there exists \( \beta \) and \( (l, r_2) \in Q^g\) s.t. \((q, r_1), (l, r_2), \beta) \in \succeq_g\). Thus there exists a transition \((l, r_2), \mu', (l', r'_2)\) \( \in \delta^g\) such that \((q', r'_1), (l', r'_2), \beta') \( \in \succeq_g\), since \( R^S \succeq_g R^{S'}\). It suffices to take \( \alpha'_2 \) in \( r'_2\) s.t. \((\alpha'_1, \alpha'_2) \models \beta'\) (this is always possible since \( r'_1 \wedge r'_2 \wedge \beta' \) is consistent). This implies that \((q', \alpha'_1), (r'_1, \alpha'_2) \) \( \in \succeq \).

\( \Rightarrow \) Assume that \( U_{nfold}(R(S)) \preceq U_{nfold}(R(S')) \). Following the same schema as previously, one can show that it is possible to derive from the relation \( \succeq \) a relation \( \succeq_g \) which can be used as a witness to deduce that \( R^S \) is simulated by \( R^{S'}\). The relation \( \succeq_g \) is constructed inductively, starting with \( \succeq_g = \{(q, r_w), (l, r_w), \emptyset\} \) and then recursively augmenting it with new elements by exploiting the relation \( \succeq \) to identify target states and regions at each step and carefully defining the \( \beta \) conditions in order to cope with conditions of definition\(^8\).

The construction of \( \succeq_g \) stops when a fix point is reached.

We provide below the main result of this section by showing that simulation between Colombo\(^db=\emptyset\) services can be reduced to simulation between the corresponding region automata. This ensures the decidability of simulation in Colombo\(^db=\emptyset\) setting since Colombo\(^db=\emptyset\) region automata are finite state machines and hence exhaustive exploration of the state-space of such machines is possible.

**Theorem 4.** Let \( S \) and \( S' \) be two Colombo\(^db=\emptyset\) services then \( S \preceq S' \) iff \( R^S \preceq_g R^{S'} \).

Proof. By definition\(^4\) we have \( S \preceq S' \) iff \( E(S) \preceq_g E(S') \) and by lemma\(^5\) we have \( U_{nfold}(R(S)) \preceq U_{nfold}(R(S')) \) iff \( R^S \preceq_g R^{S'} \). The results follows from lemma\(^4\).

### 4.2 Complexity of simulation in DB-less services

This section is devoted to complexity analysis of simulation in DB-less Colombo model, which is EXPTIME-complete. We show first that the problem is in EXPTIME then show the completeness by a reduction inspired from the work of
The reduction is from the existence of an infinite execution of an alternating Turing machine $M$ working on space polynomially bounded by the input size. That is given a word as input, $M$ can make choices of existential transition such that whatever the transitions chosen by universal states the machine continues the execution.

EXPTIME membership

**Proposition 1.** Let $S_1$ and $S_2$ be two $Colombo^{db=∅}$ services. The region automata associated to $S_1$ and $S_2$ have exponential size.

**Proof.** Let $S_1$ and $S_2$ be two $Colombo^{db=∅}$ services with $K$ is the set of constants and $X$ the set of variables in $S_1$ and $S_2$. We suppose that $n$ is the number of constants and $m$ the number of variables. A region is a set of intervals and a $v$-order on $X ∪ \{ω\}$. The number of intervals (resp. $v$-orders) is bounded by $O(n)$ (resp. $O(m)$). The number of regions is bounded by $O(|Q| \times n^m \times |m|) = O(2^{log |Q| + log m + log n})$. We conclude that the size of the region automata associated to $S_1$ and $S_2$ are respectively $O(2^{log |Q| + log m + log n})$ and $O(2^{log |Q| + log m + log n})$.

EXPTIME hardness We start by giving the definition of an alternating Turing machine $M$, then we give the construction of the service $S_{defender}$, and show the correspondence between execution of $S_{defender}$ and execution of $M$ on $w$, finally we give the construction of test of simulation $S_{spoiler} ⪯ S_{defender}$, and prove that $M$ has an infinite execution on $w$ iff $S_{spoiler} ⪯ S_{defender}$. Note that, we do not need any accepting state in our definition of $M$, and we suppose there is no transition from rejecting state. In fact, one can construct from the word acceptance problem (which is EXPTIME-hard), the problem of existence of infinite execution, by adding a set of transition from accepting state to initial state s.t we reinitialize the tape to $w$.

**Alternating Turing machine** $M$ An alternating Turing machine $M$ [21] is a tuple $(Q, q_0, Γ, δ, mode)$ where:

- $Q$ is the set of control states.
- $q_0$ the initial state.
- $Γ$ tape symbols.
- $mode: Q → \{∀, ∃\}$ labelling function of control state.
- $δ: Q × Γ → P(Q × Γ × \{L, R\})$.

A configuration $C$ of $M$ is of the form $y_1, ..., qy_j, ..., y_n$, where $q$ is a state of the machine, and the head points actually on the $j$'th letter on the tape. The transition $qa → bRq'$ is applicable from a configuration $C$ if $y_j=a$; then the successor $C'$ of $C$ is equal to $y_1, ..., y_j, q y_{j+1}, ..., y_n$ s.t $y_k = y_k$ for $k ∈ [1, n]$ and
k \neq j and y_j' = b, we note this step $C \xrightarrow{qa/bRq^{'}} C'^{'}$ or $(y_1, ..., qy_j, ..., y_n)^{qa/bRq^{'}}$

$(y_1', ..., y_j', q', y_{j+1}', ..., y_n')$, the machine $M$ starts on $C_0 = qy_1, ..., y_n$, where $y_i = w_i$, the $i$'th letter of the input word.

The definition of acceptance of an alternating Turing machine is recursive:

- If the configuration $C$ is in an accepting control state $q$, then $C$ is accepting.
- If the configuration $C$ is in an rejecting control state $q$, then $C$ is rejecting.
- If the configuration $C$ is in a universal control state $q$, then $C$ is accepted if all configuration reachable in one step are accepting, and rejecting if some configuration reachable in one step is rejecting.
- If the configuration $C$ is in a existential control state $q$, then $C$ is accepting if some configuration reachable in one step are accepting, and rejecting when all configurations reachable in one step are rejecting (this is the case of classical NTM).

$M$ is said to accept an input word $w$ if the initial configuration of $M$ is accepting, and to reject if the initial configuration is rejecting.

\[\text{Fig. 14. Alternating Turing machine } M\]

We do not look for the problem of acceptance but to the problem of the existence of an infinite execution of an alternating Turing machine $M$ on input word $w$. That is given a word as input, $M$ can make choices of existential transition such that whatever the transitions chosen by universal states the machine continues the execution. We suppose the rejected states are states without outgoing transition. Also we look for machine work on space bounded by the size of the input $n$, then if the head points on $y_1$ the machine is not allowed to execute a transition labelled with L, and if the head points on $y_n$ the machine is not allowed to execute a transition labelled with R.
Example 18. Figure 14(a) depicts an alternating Turing machine $M$, where the initial state $q_0$ is universal, and $q_1$, $q_2$ existential states. Suppose $w = ab$, then starting from initial configuration $C_0$, the machine has two successor: $C_0 \xrightarrow{q_0/aRq_1} C_1$, and $C_0 \xrightarrow{q_0/aRq_2} C_2$. Take $C_1$, it has two successor, one read $b$ and replace it by itself, the other replace $b$ by $a$. The first successor back to the first cell and the tape contains $ab$, so it back to $C_0$, the same observation is made for $C_2$. For all choices of the universal state, there is a successor of the existential state s.t the machine continue the execution. So the machine $M$ has an infinite execution on the input word $ab$.

Given an alternating Turing machine $M$ and a word $w$ of size $n$, we construct the DB-less Colombo service $S_{defender}$, that captures the execution of $M$ bounded by $n$ on $w$. $S_{defender}$ will use $n$ variables to simulates the $n$ cells. The position of the head is encoded in the variable $head$. A state $q$ of $M$ is encoded as a state $l_q$ in $S_{defender}$, a transition $q \xrightarrow{a/bR} q'$ of $M$, is encoded in a transition $(l_q, x_i = a \land head = i, qabq R(\emptyset; x_i, head), l_q')$ which means: if the actual value of the $i$th variable is equal to $a$ and the the variable $head$ contains $i$, then executes the atomic process $qabq R_i(\emptyset; x_i, head)$ which increment the variable $head$ and modify the value of $x_i$ to $b$. Because during the execution of the machine we do not control which cell is read, we create $n - 1$ transition in $S_{defender}$ from $l_q$ to $l_q'$ (the head can not move on the right of the last variable this why $i$ is range in $[1,n-1]$). If the direction is $L$, the same construction is made, with the difference that the atomic process decrements the head, and $i$ is range in $[2,n]$, in the two cases, for one transition of $M$, we create $n - 1$ transitions in $S_{defender}$.

![Diagram](image)

Fig. 15. transition in $S_{defender}$ corresponding to a transition of $M$
Example 19. Figure 15(a) depicts the transition in $S_{\text{defender}}$ which encode the transition $q_0 \xrightarrow{a/aR} q_1$ of the machine $M$ in example 18. The atomic process $qabq'_{R_1}(\emptyset; x_1, \text{head})$ is depicted in Figure 15(b). First note that there is only one transition, because in this example the size of $w=2$, and we can not move right from the second cell. Also, because we do not have the incrementation in the language of the atomic process, we must encode $n-1$ atomic process, in this example: $qabq'_{R_1}(\emptyset; x_1, \text{head})$, where in the effects, $x_1:=2$ and not 1+1.

$S_{\text{defender}}$ starts with a part initialize the variables $x_1, \ldots, x_n$ to the input word $w$, and affects 1 to the variable head.

![Diagram](image)

(a) initialization of the variables

(b) atomic process init

Fig. 16. initialization of variables in $S_{\text{defender}}$

Example 20. Figure 16 depict the initialization of the service $S_{\text{defender}}$ corresponding to the machine $M$ in example 18 where $x_1=a$, $x_2=b$ and $\text{head}=1$.

Now we give formal definition of atomic processes, and $S_{\text{defender}}$.

**Definition 9.** $\mathcal{P}$ is the set of all atomic processes used to encode actions of the machine $M$:

- $\{qabq'_{R_i}(\emptyset; x_i, \text{head}) \mid q \xrightarrow{a/bR} q' \text{ in } M \text{ and } i \in [1, n-1]\}$
- $\{qabq'_{L_i}(\emptyset; x_i, \text{head}) \mid q \xrightarrow{a/bL} q' \text{ in } M \text{ and } i \in [2, n]\}$
- $\{!m()\}$

Where:
$qabq^' Ri(∅; x_i, head)$ is an atomic process with the conditional effect:
\begin{itemize}
  \item $\theta = True.$
  \item $ev$: 
    \begin{itemize}
      \item $x_i := b.$
      \item $head := i+1.$
    \end{itemize}
\end{itemize}

$qabq^' Li(∅; x_i, head)$ is an atomic process with the conditional effect:
\begin{itemize}
  \item $\theta = True.$
  \item $ev$: 
    \begin{itemize}
      \item $x_i := b.$
      \item $head := i-1.$
    \end{itemize}
\end{itemize}

And $g^a_i$ is a condition of the form $x_i = a \land head = i$.

Definition 10. $P_{qa}$ is a subset of $P$ restricted to atomic processes modifying the variable $x_i$ and representing only the transitions of the machine $M$ from the state $q$ and reading the letter $a$.

Construction of $S_{defender}$: The defender encodes exactly the machine $M$ with $w$ as input. That means during an execution of $M$, the actual configuration of the machine $M$ has a successor if the corresponding id of an execution of $S_{defender}$ has a successor, and if the execution of $M$ blocks on a configuration, then the service also blocks. We will use a set of additional transitions to force the spoiler to follow the actions chosen by the defender during an execution.

Let $GA(S_{defender}) = \langle Q_{defender}, \delta_{defender}, q_{start}, Lstore(S_{defender}) \rangle$ where:

- the set of states of $S_{defender}$ are following:
  \begin{itemize}
    \item For each state $q$ in $M$, a state $l_q$, where the initial state of $M$ is a final state $S_{defender}$.
    \item a state $l_{copy}$, which is also a final state.
    \item $\{choice_{qbdq'} | q \xrightarrow{a/bd'} q \text{ and } q \text{ an existential state and } d = R/L\}$
    \item $q_{start}$ is the initial state.
  \end{itemize}

- $Lstore(S_{defender}) = \{x_1, ..., x_n\} \cup \{head\}$.

- $\delta_{defender}$ is composed of the following sets of transitions:
  \begin{itemize}
    \item $(l_{start}, True, init(∅; head, x_1, ..., x_n), l_q)$, where $q$ the initial state of $M$.
    \item $(l_{copy}, True, P \cup \{lm()\}, l_{copy})$, where $P$ is the set of all atomic process in $S_{spoiler}$.
    \item For each transition $q \xrightarrow{a/bd'} q'$ in $M$, where $q$ is a universal state:
      \begin{itemize}
        \item if $d = R$ then:
          \begin{itemize}
            \item $\{(l_q, g^a_i, qabq^' Ri(∅; x_i, head), l_q') | i \in [1, n-1]\}$
            \item $\{(l_q, True, lm(), l_{copy})\}$
            \item $\{(l_q, g^a_i, P \setminus \mathcal{P}_{qa}, l_{copy}) | i \in [1, n-1]\}$
          \end{itemize}
      \end{itemize}
    \item if $d = L$ then:
      \begin{itemize}
        \item $\{(l_q, g^a_i, qabq^' L_i(∅; x_i, head), l_q') | i \in [2, n]\}$
        \item $\{(l_q, True, lm(), l_{copy})\}$
        \item $\{(l_q, g^a_i, P \setminus \mathcal{P}_{qa}, l_{copy}) | i \in [2, n]\}$
      \end{itemize}
    \end{itemize}
  \end{itemize}

- For each transition $q \xrightarrow{a/bd'} q'$ in $M$, where $q$ is a existential state:
Suppose \( \alpha \) is universal. After the part of initialization \( q_0 \).

Recall that a configuration of \( M \) is a set of self loop labelled with condition/action: if \( x_i = a \) and the head point on \( i \) then we can execute the atomic process which modifies \( x_i \) to \( b \) and increments the head. So \( S_{\text{defender}} \) can only execute the atomic process representing the transition \( q \xrightarrow{a/bR} q' \) iff the actual value of \( x_i = a \) and the head points on \( i \). Note that for any actual valuation of variables, there is only one transition from the \( n-1 \) which can be executed, this is due to the guards: several \( x_i \) can verify the condition but the head equal one number from 1 to \( n \). If \( q \) is existential, then \( S_{\text{defender}} \) sends a message \( m \) before executes the atomic process. The state \( l_{\text{copy}} \) contain a set of self loop labelled with all atomic processes \( \mathcal{P} \) and \( \{l_m\} \) (if \( S_{\text{defender}} \) reaches this state, it wins the simulation). All transitions which reach the state \( l_{\text{copy}} \) are used to prevent cheating of \( S_{\text{spoiler}} \) during test of simulation.

The next Lemma asserts that the execution of \( S_{\text{defender}} \) captures exactly the execution of \( M \). The proof is made by induction.

**Lemma 6.** Given an alternating Turing machine \( M \) work on space polynomially bounded by the input \( w \), \( M \) has an infinite execution on \( w \) iff \( S_{\text{defender}} \) has an infinite execution without passing through the state \( l_{\text{copy}} \).

**Proof.** Recall that a configuration of \( M \) is \( C = y_1, ..., y_j, ..., y_n \), where the head points on the \( j \)th cell. The initial configuration of \( M \) is \( C_0 = y_1, ..., y_n \). Suppose \( q \) is universal. After the part of initialization \( E(S_{\text{defender}}) \) is in an \( id = (l_q, \alpha(L_{\text{store}})) \) where \( \alpha(x_i) = w_i \), the \( i \)th letter of the input word \( w \). \( \alpha(\text{head}) = 1 \).

Suppose \( C_0 \xrightarrow{q_a/bRq} C' \), that means \( y_1 = a \), so \( \alpha(x_1) = a \). From construction of \( S_{\text{defender}} \) there exists a transition \( l_q \xrightarrow{g_i^* \mid qabq' \overset{R_i(\emptyset; x_i, \text{head})}} l_{q'} \) where \( g_i^* \)
\[x_1 = a \land \text{head} = 1, \text{so } (l_q, \alpha(Lstore)) \xrightarrow{qa'b' R_i(\emptyset; \alpha'(x_1), \alpha'(\text{head}))} (l'_{q}, \alpha'(Lstore))\] where: (i) \(\alpha'(x_1)= y'_1=b\), and \(\alpha'(\text{head})=2\).

Now suppose \(M\) is in configuration \(C=y_1, ..., qy_j, ..., y_n\), and there exists in \(E(S_{\text{defender}})\) an \(id=(l_q, \alpha(Lstore))\), where \(y_i=\alpha(x_i)\) and \(\alpha(\text{head})=j\). If \(C \xrightarrow{qa'b'R_j(\emptyset; x_j, head)} C'\) exists, then \(y_j=a\), \(y'_j=b\) and \(\alpha(x_j)=a\). From construction of \(S_{\text{defender}}\), we know there is a transition \(l_q\xrightarrow{a_i} l'_q\) where \(g_1: x_1 = a \land \text{head} = j\), so there exists in \(E(S_{\text{defender}})\) a transition \((l_q, \alpha(Lstore)) \xrightarrow{qa'b'R_j(\emptyset; \alpha'(x_1), \alpha'(\text{head}))} (l'_{q}, \alpha'(Lstore))\) where: (i) \(\alpha'(x_j)= y'_j=b\), and \(\alpha'(\text{head})=j+1\).

The same reasoning is used if \(M\) is in a configuration where the state is existential, the only difference is \(S_{\text{defender}}\) starts by sending the message \(m()\), then executes the atomic process \(qa'b'R_j(\emptyset; x_j, head)\), this additional transition does not change values of variables.

From construction of \(S_{\text{defender}}\) after the part of initialization, the execution of the service reaches an \(id=l_q, \alpha(Lstore)\), where \(q\) is the initial state of \(M\), suppose it is universal. \(\alpha(x_i)=w_i\), the \(i\)’th letter of the input word \(w\). \(\alpha(\text{head})=1\). If in \(E(S_{\text{defender}})\), there exists a transition \((l_q, \alpha(Lstore)) \xrightarrow{qa'b'R_i(\emptyset; \alpha'(x_1), \alpha'(\text{head}))} (l'_{q}, \alpha'(Lstore))\), then \(\alpha'(x_1)=b\) and \(\alpha'(\text{head})=2\). From construction of \(S_{\text{defender}}\), there exists \(C \xrightarrow{qa'b'R_j(\emptyset; x_j, head)} C'\) where \(y_1=a\) and \(y'_1=b\).

Suppose in \(E(S_{\text{defender}})\) there is an \(id=(l_q, \alpha(Lstore))\), where \(q\) is an universal state of \(M\). \(\alpha(x_i)=w_i\), the \(i\)’th letter of word in the tape. Suppose \(\alpha(\text{head})=j\) and there is a configuration of \(M\), \(C=y_1, ..., qy_j, ..., y_n\), where \(\alpha(x_i)=y_i\). If in \(E(S_{\text{defender}})\), there exists a transition \((l_q, \alpha(Lstore)) \xrightarrow{qa'b'R_j(\emptyset; \alpha'(x_1), \alpha'(\text{head}))} (l'_{q}, \alpha'(Lstore))\), then \(\alpha'(x_j)=b\) and \(\alpha'(\text{head})=j+1\). From construction of \(S_{\text{defender}}\), there exists \(C \xrightarrow{qa'b'R_j(\emptyset; x_j, head)} C'\) where \(y_j=a\) and \(y'_j=b\).

The same idea is used to prove the correspondence when \(q\) is existential. Also when the direction is left we just replace \(R\) by \(L\) and decrements the head.

**Example 21.** The Figure [17] depicts the part of service \(S_{\text{defender}}\) corresponding to the transition \(q_0 \xrightarrow{a/L} q_1\) where \(q_0\) is universal, and the two transition from the existential state \(q_1: q_1 \xrightarrow{b/L} q_0, q_1 \xrightarrow{b/aL} q_0\) of the machine \(M\) of example [18].

**Construction of \(S_{\text{spoiler}}\)** The spoiler also uses variables \(x_1, ..., x_n\) and head. And starts as \(S_{\text{defender}}\) by initializing the variables to \(w\), and the head to 1. It encodes all transitions that the machine \(M\) can do, the service can repeat
them infinitely often. If $M$ has a transition $q \xrightarrow{a/R} q'$ and $q$ universal, then the service has $(n-1)$ self-loop on state $q_{univ}$ labelled with $qabq_{Ri}R_i(\emptyset; x_i, head)$. If $q$ an existential state, first the service goes to an intermediate state $q_{exist}$ with sending the message $m()$, then has $n-1$ transition labelled with $qabq_{Ri}R_i(\emptyset; x_i, head)$, these transitions back to $q_{univ}$. As we will see in its formal definition, there is no guard before executing any atomic process. Then $S_{spoiler}$ can choose to execute any actions without constraints on actual values of variables.

Let $GA(S_{spoiler}) = (Q_{spoiler}, \delta_{spoiler}, q_{start}, q_{univ}, L_{store}(S_{spoiler}))$ where:

- $Q_{spoiler} = \{q_{start}, q_{univ}, q_{exist}\}$, where $q_{start}$ is the initial state and $q_{univ}$ final state.
- $L_{store}(S_{spoiler}) = \{x_1, ..., x_n\} \cup \{head\}$.
- $\delta_{spoiler}$ is composed of the following sets of transitions:
  - $(q_{start}, True, init(\emptyset; head, x_1, ..., x_n), q_{univ})$.
    Where $init$ initializes $head$ to 1 and $x_i$ to $w_i$ (the $i$th) letter of $w$.
  - $(q_{univ}, True, \{m()\}, q_{exist})$.
  - For each transition $q \xrightarrow{a/bd} q'$ in $M$, where $q$ is a universal state:
    * if $d = R$ then :
      - $\{ (q_{univ}, True, qabq_Ri(\emptyset; x_i, head), q_{univ}) \mid i \in [1, n-1] \}$.
      Where $x_i$ receives $b$ and $head$ receives $i + 1$ (moves on the right).
    * if $d = L$ then :
      - $\{ (q_{univ}, True, qabq_Li(\emptyset; x_i, head), q_{univ}) \mid i \in [2, n] \}$.
      Where $x_i$ receives $b$ and $head$ receives $i - 1$ (moves on the left).
  - For each transition $q \xrightarrow{a/\ldots} q'$ where $q$ is a existential state:
* if \( d = R \) then :
  \[ \{(q_{\text{exist}}, \text{True}, qabq' R_i(\emptyset; x_i, \text{head}), q_{\text{univ}}) \mid i \in [1, n-1]\} \]

* if \( d = L \) then :
  \[ \{(q_{\text{exist}}, \text{True}, qab L_i(\emptyset; x_i, \text{head}), q_{\text{univ}}) \mid i \in [2, n]\} \]

Example 22. The Figure 18 depicts the part of service \( S_{\text{spoiler}} \) corresponding to the transition \( q_0 \xrightarrow{a/R} q_1 \) where \( q_0 \) is universal, and the two transition from the existential state \( q_1 \):

\[ q_1 \xrightarrow{b/L} q_0, \quad q_1 \xrightarrow{b/aL} q_0 \]

of \( M \) in example 18.

**Fig. 18.** part of \( S_{\text{spoiler}} \)

**Lemma 7.** Given an alternating Turing machine \( M \) working in space bounded by the size of the input \( w \), \( M \) has an infinite computation on \( w \) iff \( S_{\text{spoiler}} \preceq S_{\text{defender}} \).

**Proof.** \( S_{\text{spoiler}}, S_{\text{defender}} \) start by initializing the variables representing the cells with the input word. If \( M \) has a transition \( q \xrightarrow{a/R} q' \) and \( q \) universal, then \( S_{\text{spoiler}} \) has \( n-1 \) loops:

\[ q_{\text{universal}} \xrightarrow{qabq' R_i(\emptyset; x_i, \text{head})} q_{\text{universal}} \] and the service \( S_{\text{defender}} \) contains \( n-1 \) transitions from \( l_q \xrightarrow{g_7} q' \mid qabq' R_i(\emptyset; x_i, \text{head}) \) \( l_q' \). So the difference with \( S_{\text{spoiler}} \) is that, \( S_{\text{defender}} \) can only execute one transition representing the action \( q \xrightarrow{a/R} q' \) if the actual value of \( x_i = a \) and the head points on \( i \). Suppose the condition is verified, then if \( S_{\text{spoiler}} \) choose to execute any transition different from \( qabq' R_i(\emptyset; x_i, \text{head}) \), the \( S_{\text{defender}} \) wins the game by choosing the transition which reaches the state \( l_{\text{copy}} \); if \( S_{\text{spoiler}} \) chooses to execute \( qabq' R_i(\emptyset; x_i, \text{head}) \), then \( S_{\text{defender}} \) execute \( qabq' R_i(\emptyset; x_i, \text{head}) \) and the game continue. Now, suppose the condition is not verified, the service \( S_{\text{defender}} \) is blocked, then \( S_{\text{spoiler}} \) wins the game by executing \( qabq' R_i(\emptyset; x_i, \text{head}) \).
If $q$ is existential, $S_{\text{spoiler}}$ has a transition $q_{\text{exist}} \xrightarrow{!m()} q_{\text{exist}}$ and $n - 1$ transitions from $q_{\text{exist}} \xrightarrow{\text{qabq} \ x, \text{head}} q_{\text{univ}}$. $S_{\text{defender}}$ contains a transition from $l_q \xrightarrow{!m()} l_{\text{qb}}$ and $n - 1$ transitions $l_{\text{qb}} \xrightarrow{g} l_{\text{qab}} \xrightarrow{\text{qabq} \ x, \text{head}} l_q'$. Suppose the actual values of variables verify the condition, if $S_{\text{spoiler}}$ chooses another action different from sending the message, it looses, if it send the message it reaches the state $q_{\text{exist}}$, then $S_{\text{defender}}$ reaches an intermediate state $l_{\text{qab}}$ by sending the message, $S_{\text{spoiler}}$ can do any action, but if it chooses an action different from $\text{qabq} \ x, \text{head}$ it looses the game, if it choose the action $q \xrightarrow{a/b} q'$, then the game continues.

Note that, after sending the message $m()$, $S_{\text{spoiler}}$ reaches $l_{\text{exist}}$, this transition can be simulated by many transition of $S_{\text{defender}}$, because $S_{\text{defender}}$ may have several transition labelled with sending $m()$ at $l_q$, so there is no simulation if all test of simulation are false, that means $S_{\text{defender}}$ is blocked whatever it chooses. If one choice is not blocking the game continue and hence there is simulation if the defender can always chooses a non blocking blocking state for existential transitions of $M$, and does not block for all universal transition of $M$. Which means $M$ during its execution has always a successor, so there exists an infinite computation.

**Theorem 5.** Given two DB-less Colombo services $S$, $S'$, checking whether $S \preceq S'$ is EXPTIME-complete.

## 5 Decidability of simulation in DB-bound Colombo

We study in this section the simulation problem in the setting of a Colombo model with a bounded global database (i.e., the size of database $W$ is at most equal to a constant $k$). Given two services $S$ and $S'$, $S$ is $k$-bounded simulated by $S'$ means $S'$ does not mimic $S$ on all executions but only executions where the size of database is at most equal to $k$. We will prove that the simulation is decidable in this setting by transforming a test of $k$-bounded simulation between Colombo services to a test of simulation between two DB-less Colombo services. This is done by coding the database in a set of variables. First we start by giving the definition of $k$-bounded extended state machines, which is used to captures the notion of simulation only on executions where database is bounded. Then we give the construction of the DB-less service and prove the equivalence of the two tests.

### 5.1 $k$-bounded extended state machine $E^k(S)$

The $k$-bounded extend state machine $E^k(S)$ of a Colombo service $S$ is extracted from the extended state machine $E(S)$ of $S$.

**Definition 11.** ($E^k(S)$) Let $S$ be a Colombo service and $E(S) = (Q, Q_0, \Delta)$ the associated extended state machine, then $E^k(S) = (Q^k, Q^k_0, \Delta^k)$ is the $k$-bounded extend state machine of $S$ where:
Q^k = \{(l, I, \alpha) \mid (l, I, \alpha) \in Q \text{ and } |I| \leq k\}.

The k-bounded extended state machine of S is the sub transition system of E(S) where in all configuration the database is k-bounded. Like E(S), a run \(\sigma\) of \(E^k(S)\) is a finite sequence \(\sigma = id_0 \xrightarrow{\mu_0} id_1 \xrightarrow{\mu_1} \ldots \xrightarrow{\mu_{n-1}} id_n\) where \(id_0\) is an initial configuration and \(id_n\) final configuration but \(|I_i| \leq k\) for \(i \in [0, n]\) where \(id_i = (l_i, I_i, \alpha_i)\). Due to infinite number of k-bounded initial databases, all runs of \(E^k(S)\) form a forest which is included in the forest of all runs of \(E(S)\). The only difference with the original extended state machine is the constraint on the size of the database.

Example 23. Figure 19 depicts a simple Colombo service \(S\) which receive two variables \(x, y\) and then uses the atomic process \(add\) to insert the tuple \((x, y)\) in the global database \(R\). The service can make an infinite loop during an execution, and inserts an unbounded number of tuples in \(R\).

The figure 20 depicts two path of \(E(S)\). The path depicts in Fig 20(a) starts with an instance of \(R\) which contains one tuple \((6, h)\) then insert the tuple \((7, 2)\), this path of \(E(S)\) is also a path in \(E^2(S)\), unlike the second path (Fig 20(b)) which start with an instance containing only the tuple \((8, 1)\) then insert the tuple \((9, 3)\), and finally insert the tuple \((1, 2)\). The second path is not a part of \(E^2(S)\) because the database of the last configuration does not satisfies the condition of \(|R| \leq 2\).

5.2 Bounded simulation \(\preceq_k\)

Definition 12. A Colombo service \(S\) is k-bounded simulated by a Colombo service \(S'\), noted \(S \preceq_k S'\), iff \(E^k(S) \preceq E^k(S')\).

\(S \preceq_k S'\), means \(S'\) mimic executions of \(S\) not on all databases, but only on \(k\) - bounded databases. Of course if \(S \preceq S'\) then \(S \preceq_k S'\) but the inverse is not true.
5.3 Mapping of Colombo service to Colombo DB-less service

We will prove the decidability of bounded simulation by proving that for any two colombo services \( S, S' \), test of k-bounded simulation \( S \preceq_k S' \) is equivalent to test \( \mathcal{M}(S) \preceq \mathcal{M}(S') \), where \( \mathcal{M}(S) \) and \( \mathcal{M}(S') \) are DB-less services where a set of variables is used to simulate the world database instance using the mapping \( \mathcal{M} \).

Suppose \( W = R \) and \( \text{arity}(R) = n \), the maximum number of elements in a k-bounded database is \( n \times k \). Then all k-bounded instance can be encoded with a finite number of variables \((n \times k)\). We will explain in the following, how transform a Colombo service into a DB-less service to test the k-bounded simulation.

We will use the following example to explain the transformation from a Colombo service to a DB-less service.

**Example 24.** Figure [21](a) depicts Colombo service Search. Search retrieve a product for a client in the global relation Inventory (Figure [21](c)) and send its price if the quantity requested is available using the atomic process check-item depicted in Figure [21](b). If the quantity of the product is equal to zero then the product is deleted from the stock.

**Database variables DV** As said earlier, the number of variables added to encode the database depends on the arity of database schema and \( k \). To simplify the explanation and the proof without loose of generality, from now we suppose \( W \) contains only one relation \( R(A_1; B_1, \ldots, B_m) \). We will denote the set of database variables \( DV \).
**Definition 13.** Let \( R(A_1; B_1, \ldots, B_m) \) the world database schema, and \( k \) a constant, then \( DV = \{ dv_{ij} \mid i \in [1,k] \text{ and } j \in [1,m+1] \} \).

![Diagram of Colombo services](image)

**Fig. 21.** Colombo services Search

Note that the variables \( dv_{11} \) represents the possible values of the keys (the attribute \( A_1 \)). Figure 22 depicts two instance of relation Inventory, and the set \( DV \) corresponding to 2-bounded databases. The elements of the tuple \( (HP5, 31, 200) \) of the first instance (Figure 22(a)), are mapped respectively to the variables \( dv_{11}, dv_{12}, dv_{13} \) and the those of the tuple \( (XS3, 48, 159) \) to \( dv_{21}, dv_{22}, dv_{23} \) of the first valuation of \( DV \) (Figure 22(d)). The tuple \( (HS7, 23, 120) \) of the second instance (Figure 22(b)), is mapped to \( dv_{11}, dv_{12}, dv_{13} \) of the second valuation of \( DV \) (Figure 22(e)).

**Initialization of DV** All executions of Colombo services start with initializing variables to null (\( \omega \)), to simulate the database, we will add an additional state...
and transition at the beginning of the DB-less service. If \( S \) is Colombo service, the corresponding DB-less service will start with a transition labelled with ?database\((dv_{1j},...,dv_{1m+1},...,dv_{km+1})\).

The Figure 23 depicts the initialization part added to the DB-less service corresponding to the service Search. Here the bound is equal to 2.

Note that because in the original model suppose only databases which respect the key constraint, during reception of message of \( DV \), we suppose all \( dv_{i1} \) are different (in fact we can encode the test of the key constraint).

**Atomic process transformation** Recall that a Colombo service access to the database only using *atomic process*, either to retrieve information or modify it. An atomic process is a triplet \( p = (I,O,CE) \) where: \( I= u_1,...,u_n \) and \( O= v_1,...,v_m \) are respectively input and output and \( CE = \{ (\theta,es,ev) \} \) is a set of conditional effects where \( \theta \) is a condition \( ev \) effects on outputs and \( es \) effects on database. We will explain how transforming an atomic process \( p \) acts on database \( R \) to a corresponding atomic process \( p_v \) which acts on \( DV \) modulo the bound \( k \).

An atomic process access to the values of a database using the access function \( f^R_j \) through the condition \( \theta \) or in \( ev \) by affecting a value to an output variable. It can also access the database by modify it using the state effects \( es(Insert,Delete,Modify) \). In follow, we will explain how to transform the access function \( f^R_j \) appearing in \( \theta \) and \( ev \) into condition \( \theta_v \) and outputs effects \( ev_v \) which involve only variables of \( Lstore \cup DV \). We will also explain how to transform \( es \) into \( ev_v \).

1. \( f^R_j \)

   Let "\( f^R_j(t) \ op t'' \)" be a condition in a \( \theta \) of an atomic process \( p \). The corresponding \( p_v \) will contain \( k \) conditions \( \theta_{v_i} \) with \( i \in [1,k] \), where
As example the atomic process check-item depicted in Figure 21(b), contains a condition \( f^2_{\text{Inventory}}(\text{item}) \geq qty \), this test will be transformed to \( \theta_{v1}, \theta_{v2} \) where:

- \( \theta_{v1} : dv_{v1} = \text{item} \wedge dv_{v2} \geq qty \).
- \( \theta_{v2} : dv_{v2} = \text{item} \wedge dv_{v3} \geq qty \).

Let \( "v_i := f^R_j(t)" \). Here the output variable \( v_i \) receives the value of \( f^R_j(t) \), then as for the condition \( \theta \), we need first to retrieve the key variable \( dv_{v1} \) equal to \( t \), then affect \( dv_{v1} \) to \( v_i \). The test and the affectation is made \( k \) times. The corresponding atomic process will contains a set of pair \((\theta_i, ev_i)\) where \( \theta_i = "dv_{v1} = t" \), and \( ev_i = "v_i := dv_{v1+1}" \).

Continuing with the atomic process check - item, there is an affectation : \( price := f^2_{\text{Inventory}}(\text{item}) \), this will be transformed to two pairs \((\theta_i, ev_i)\):

- \( (\theta_1, ev_1) : if \ dv_{v1} = \text{item} \ then \ price := \ dv_{v2} \).
- \( (\theta_2, ev_2) : if \ dv_{v2} = \text{item} \ then \ price := \ dv_{v3} \).

3. \( es \)

The atomic process can modify the database with the set \( es \), which can contains one of following expressions:

- insert \( R(t_1, s_1, \ldots, s_m) \). In \( p_v \), the insertion is encoded by retrieving a variable \( dv_{v1} = \omega \), then affect respectively \( t_1, s_1, \ldots, s_m \) to \( dv_{v1}, dv_{v2}, \ldots, dv_{v3} \). That means we will add k pair of the form \((\theta_i, ev_i)\) where
  - \( \theta_i = "dv_{v1} = \omega" \).
  - \( ev_i = \{ dv_{v1+j} = s_j \ | \ j \in [2,m+1] \ \text{and} \ l \in [1,m] \} \cup \{ dv_{v1} := t_1 \} \).

- delete \( R(t_1) \). In \( p_v \), the suppression is made by first retrieve the key variable \( dv_{v1} \) equal to \( t_1 \) then affect to the variable \( dv_{v1} \) the value \( \omega \). Again the new atomic process \( p_v \) will contain k pair of the form \((\theta_i, ev_i)\) where
  - \( \theta_i = "dv_{v1} = t_1" \).
  - \( ev_i = dv_{v1} := \omega \).

- modify \( R(t_1, r_1, \ldots, r_m) \). In \( p_v \), to simulate the modification first we need to find the key variable \( dv_{v1} \) equal to \( t_1 \), then affect to \( dv_{v1} \) the corresponding \( r_i \) if \( r_i \) is different from \( "\_" \). As for the previous cases, we add k pairs of the form \((\theta_i, ev_i)\) where
  - \( \theta_i = "dv_{v1} = t_1" \).
  - \( ev_i = \{ dv_{v1+j} := r_j \ | \ r_i \neq "\_" \ \text{and} \ l = j + 1 \} \).

The atomic process check - item Delete a product if its quantity is equal to zero : Delete \ Inventory(item). This action on database will be transformed into two pairs \((\theta_i, ev_i)\):
- $(\theta_1, ev_1):$ \[ \text{if } dv_{11} = 0 \text{ then } dv_{11} := \omega. \]
- $(\theta_2, ev_2):$ \[ \text{if } dv_{21} = 0 \text{ then } dv_{21} := \omega. \]

**Definition 14.** Let \( p = (I, O, CE) \) an atomic process acts on \( R(A_1; B_1, \ldots, B_m) \), then \( p_v = (I_v, O_v, CE_v) \) is constructed as follow:

- \( I_v = I \cup DV. \)
- \( O_v = O \cup DV. \)
- The set \( CE_v \) is obtained by applying the rules defined before.

The Figure 24(b) depicts the atomic process check-item\(v\).

Now, we will give the definition of the mapping from a service Colombo \( S \), to a corresponding DB-less service \( M(S) \).

**Definition 15.** Let \( GA(S) = \langle Q, \delta, l_0, F, L_{\text{Store}}(S) \rangle \) be a guarded automata of a service \( S \) and \( k \) a constant, then \( GA(M(S)) = \langle Q_{M(S)}, \delta_{M(S)}, l_{\text{init}}, F_{M(S)}, L_{\text{Store}}(M(S)) \rangle \) where:

- \( Q_{M(S)} = Q \cup l_{\text{init}} \), all state of \( S \) are included in \( M(S) \). \( M(S) \) contains an additional state corresponding to the initialization of \( DV \).
- \( l_{\text{init}} \) is the initial state of \( M(S) \).
- \( F_{M(S)} = F \), the set of final states.
- \( L_{\text{Store}}(M(S)) = L_{\text{Store}}(S) \cup DV. \)
- \( \delta_{M(S)} \) is constructed as follow:
  - A transition from \( l_{\text{init}} \) to \( l_0 \) labelled with the reception of the message \( ? \text{database} (v_{11}, \ldots, v_{ij}) \), where \( i \in [1, k] \) and \( j \in [1, m + 1] \).
  - If \( (l, \theta, \mu, l') \in \delta \) and \( \mu \neq p(u_1, \ldots, u_i; v_1, \ldots, v_j, (\psi, E)) \), then \( (l, \theta, \mu, l') \in \delta_{M(S)} \).
  - If \( (l, \theta, \mu, l') \in \delta \) and \( \mu = p(u_1, \ldots, u_i; v_1, \ldots, v_j, (\psi, E)) \), then \( (l, \theta, p_v, l') \in \delta_{M(S)} \).

---

**Fig. 24.** \( M(\text{Search}) \)
Figure 24 depicts the DB-less service $\mathcal{M}(\text{Search})$, the service starts with receiving a message containing the values of $DV$, the second difference is the atomic process $\text{check \ – \ item}_v$ which acts on the set $DV$.

Now we will proof the equivalence between testing k-bounded simulation between two Colombo service and the test of simulation between the corresponding DB-Less services.

**Lemma 8.** Let $S$ a Colombo service, $E^k(S) = (Q^k, Q^k_0, F^k, \Delta^k)$ its k-bounded extended state machine and $E(\mathcal{M}(S))$ the extend state machine of DB-less $\mathcal{M}(S)$, then

- If $(q_i, I_i, \alpha_i) \in Q^k$ then $\exists (q_i, \alpha'_i) \in Q_{\mathcal{M}(S)}$ s.t $\alpha'_i|_{L\text{store}} = \alpha_i$ and $\alpha'_i|_{DV} = I_i$ and
- $\forall (q_i, I_i, \alpha_i) \xrightarrow{\mu_i} (q_j, I_j, \alpha_j), \exists (q_i, \alpha'_i) \xrightarrow{\mu_i} (q_j, \alpha'_j)$ s.t $\alpha'_j|_{L\text{store}} = \alpha_j$ and $\alpha'_j|_{DV} = I_j$.

Lemma 8 asserts that for each state in the k-bounded state machine of $S$ there exists a corresponding state in the extended state machine of DB-less $\mathcal{M}(S)$ s.t the valuation of $DV$ is equal to database $I$ and the valuation of variables of $L\text{store}$ in the two states are equal. We will prove that in the following by induction.

**Proof.** The base:

1. Suppose $(q_0, I_0, \alpha_0) \in Q^k$, from the construction of $\mathcal{M}(S)$ there exists a state $(q_0, \alpha'_0) \in Q_{\mathcal{M}(S)}$ s.t (i) $\alpha'_0|_{L\text{store}} = \alpha_0 = \emptyset$ and (ii) $\alpha'_0|_{DV} = I_0$. The first point is easy to see, from the definition of an execution of a service, all variables start with no values. The second point come from the construction of $\mathcal{M}(S)$, where there is a transition from $q_{\text{init}}$ to $q_0$ labelled with reception of messages over all variables of $DV$, because the variables and the database are range over the same infinite domain, necessarily there exists a valuation of the variables of the message $\alpha'_0(DV)=I_0$.

2. Let $(q_0, I_0, \alpha_0) \xrightarrow{\mu_0} (q_j, I_j, \alpha_j)$ a transition in $\Delta^k$, then from the construction of $\mathcal{M}(S)$ and 1, we know there exists a transition $(q_0, \alpha'_0) \xrightarrow{\mu'_0} (q'_j, \alpha'_j)$ where:
   - if $\mu_0=?m(\alpha_0(u_1), \ldots, \alpha_0(u_n))$, then $\mu'_0=?m(\alpha'_0(u_1), \ldots, \alpha'_0(u_n))$ where:
     - $\alpha'_j|_{L\text{store}} - \{v_1, \ldots, v_n\} = \alpha_j|_{L\text{store}} - \{v_1, \ldots, v_n\}$.
     - $\alpha'_j(v_k) = \alpha_j(v_k)$ for $k \in [1, \ldots, n]$. This is due to the fact the two substitution have the same infinite co-domain.
   - if $\mu_0=lm(\alpha_0(u_1), \ldots, \alpha_0(u_n))$, then $\mu'_0=lm(\alpha'_0(u_1), \ldots, \alpha'_0(u_n))$ where:
     - $\alpha'_j|_{L\text{store}} = \alpha_j$; because $\alpha_0 = \alpha_j$ and $\alpha_0 = \alpha_0|_{L\text{store}}$ and $\alpha'_0 = \alpha'_j$.
   - if $\mu_0=p(\alpha_0(u_1), \ldots, \alpha_0(u_n), \alpha_0(DV); \alpha_j(v_1), \ldots, \alpha_j(v_m), \alpha_j(DV))$, then $\mu'_0=p(\alpha'_0(u_1), \ldots, \alpha'_0(u_n), \alpha'_0(DV); \alpha'_j(v_1), \ldots, \alpha'_j(v_m), \alpha'_j(DV))$ where:
• \( a'_j|DV = I_j \): from construction of \( M(S) \) and \( a'_0|Lstore = \alpha_0 \) and \( a'_0|DV = I_0 \), \( DV \) is modified regarding to updates made by \( p \), where the values of variables depends on inputs and the database.

• \( a'_j|Lstore-\{v_1,...,v_m\} = a'_j|Lstore-\{v_1,...,v_m\} \) because:
  \( a'_j|Lstore-\{v_1,...,v_m\} = a'_j|Lstore-\{v_1,...,v_m\} \) and \( a'_j|Lstore-\{v_1,...,v_m\} = a'_j|Lstore-\{v_1,...,v_m\} \) and \( a'_j|Lstore-\{v_1,...,v_m\} = a'_j|Lstore-\{v_1,...,v_m\} \).

• \( a'_j|\{v_1,...,v_m\} = a'_j|\{v_1,...,v_m\} \): From the construction of \( p_k \) and \( \alpha_i = \alpha'_i \) and \( \alpha'_i|DV = I_0 \).

The iteration i:

1. Suppose \( (q_i, I_i, \alpha_i) \in \mathbb{Q}^k \) and there exists \( (q_i, \alpha_i') \in \Delta_M(S) \) where \( I_i = \alpha'_i|DV \) and \( \alpha_i = \alpha'_i|Lstore \), then for each transition \( (q_i, I_i, \alpha_i) \xrightarrow{\mu_i} (q'_i, I'_j, \alpha'_j) \) in \( \Delta^k \),

we know there exists a transition \( (q_0, \alpha'_0) \xrightarrow{\mu'_0} (q'_0, \alpha'_0) \) from the construction of \( M(S) \) and 1 where:

- if \( \mu_i = m(\alpha_j(u_1), ..., \alpha_j(u_n)) \), then \( \mu_i' = m(\alpha'_j(u_1), ..., \alpha'_j(u_n)) \) where:
  • \( \alpha'_j|Lstore=\alpha'_j|Lstore-\{v_1,...,v_m\} \); because: \( \alpha'_j|Lstore-\{v_1,...,v_m\} = \alpha'_j|Lstore-\{v_1,...,v_m\} \) and \( \alpha'_j|Lstore-\{v_1,...,v_m\} = \alpha'_j|Lstore-\{v_1,...,v_m\} \) and \( \alpha'_j|Lstore-\{v_1,...,v_m\} = \alpha'_j|Lstore-\{v_1,...,v_m\} \).
  • \( \alpha'_i|DV = I_j \) because: \( I_i = I_j, \alpha'_i|DV = I_i \) and \( \alpha'_i|DV = \alpha'_i|DV \).

- if \( \mu_i = m(\alpha_i(u_1), ..., \alpha_i(u_n)) \), then \( \mu_i' = m(\alpha'_i(u_1), ..., \alpha'_i(u_n)) \) where:
  • \( \alpha'_i|Lstore=\alpha'_i|Lstore \); because: \( \alpha_i = \alpha_j \) and \( \alpha_i = \alpha'_i|Lstore \) and \( \alpha_i = \alpha'_j \).

- if \( \mu_i = p(\alpha_i(u_1), ..., \alpha_i(u_n), \alpha_i(DV); \alpha_j(v_1), ..., \alpha_j(v_m), \alpha'_j(DV)) \), then

  \( \mu_i' = p(\alpha'_i(u_1), ..., \alpha'_i(u_n), \alpha'_i(DV); \alpha'_i(v_1), ..., \alpha'_i(v_m), \alpha'_j(DV)) \) where:

  • \( \alpha'_i|DV = I_j \) from construction of \( M(S) \) and \( \alpha'_i|Lstore = \alpha_i \) and

  • \( \alpha'_i|DV = I_i, DV \) are modified regarding to updates made by \( p \), where

  • the values of variables depends on inputs and the database.

• \( a'_j|Lstore-\{v_1,...,v_m\} = a'_j|Lstore-\{v_1,...,v_m\} \) because:

• \( a'_j|Lstore-\{v_1,...,v_m\} = a'_j|Lstore-\{v_1,...,v_m\} \) and \( a'_j|Lstore-\{v_1,...,v_m\} = a'_j|Lstore-\{v_1,...,v_m\} \).

• \( a'_j|\{v_1,...,v_m\} = a'_j|\{v_1,...,v_m\} \): because: \( a'_0 = a'_0 \) and \( a'_0|DV = I_0 \).

**Lemma 9.** Let \( S \) a Colombo service, \( E^k(S) = (\mathbb{Q}^k, \mathbb{Q}_0^k, F^k, \Delta^k) \) its \( k \)-bounded extended state machine and \( E(M(S)) \) the extend state machine of DB-less \( M(S) \), then

- If \( (q_i, \alpha'_i) \in \mathbb{Q}_M(S) \) then \( \exists (q_i, I_i, \alpha_i) \in \mathbb{Q}^k \) s.t \( \alpha'_i|Lstore = \alpha_i \) and \( \alpha'_i|DV = I_i \) and
The base machines by induction: will prove by induction the correspondence between the two extended state

Proof. The proof of this lemma is in the same spirit of the previous one, we will prove by induction the correspondence between the two extended state machines by induction:

The base:

1. Suppose \((q_0, \alpha'_0) \in Q_{M(S)}\), from the construction of \(M(S)\) there exists a state \((q_0, I_0, \alpha_0) \in Q^k\) s.t (i) \(\alpha'_0|_{\text{Lstore}} = \alpha_0 = \emptyset\) and (ii) \(\alpha'_0|_{\text{DV}} = I_0\). (i) comes from the definition of an execution of a service, all variables of Lstore start with null values. (ii) comes from the construction of \(M(S)\), where there is a transition from \(q_{\text{init}}\) to \(q_0\) labelled with reception of messages over all variables of \(\text{DV}\), because the variables and the database are range over the same infinite domain, necessarily there exists a valuation of the variables of the message \(\alpha'_0(\text{DV}) = I_0\).

2. Let \((q_0, \alpha'_0) \xrightarrow{m} (q'_j, \alpha'_j)\) a transition in \(\Delta_{M(S)}\), then from the construction of \(M(S)\) and 1, we know there exists a transition \((q_0, I_0, \alpha_0) \xrightarrow{m} (q_j, I_j, \alpha_j)\) in \(\Delta^k\) where:

- if \(\mu'_0 = ?m(\alpha'_j(u_1), ..., \alpha'_j(u_n))\), then \(\mu_0 = ?m(\alpha_j(u_1), ..., \alpha_j(u_n))\) where:
  
  • \(I_j = \alpha'_j|_{\text{DV}}\) because \(\alpha'_j|_{\text{DV}} = \alpha'_0|_{\text{DV}}\) and \(\alpha'_0|_{\text{DV}} = I_0\) and \(I_0 = I_j\).
  
  - if \(\mu'_0 = !m(\alpha'_0(u_1), ..., \alpha'_0(u_n))\), then \(\mu_0 = !m(\alpha_0(u_1), ..., \alpha_0(u_n))\) where:
    
    • \(\alpha_j = \alpha'_j|_{\text{Lstore}}\) because \(\alpha'_0 = \alpha'_j\) and \(\alpha_0 = \alpha'_0|_{\text{Lstore}}\) and \(\alpha_0 = \alpha_j\).
    
    • \(I_j = \alpha'_j|_{\text{DV}}\) because \(\alpha'_j = \alpha_j\) and \(I_0 = I_j\) and \(I_0 = \alpha'_0|_{\text{DV}}\).

- if \(\mu'_0 = p(\alpha'_0(v_1), ..., \alpha'_0(v_m), \alpha'_0(\text{DV}); \alpha'_j(v_1), ..., \alpha'_j(v_m), \alpha'_j(\text{DV}))\), then \(\mu_0 = p(\alpha_0(v_1), ..., \alpha_0(v_m), \alpha_0(\text{DV}); \alpha_j(v_1), ..., \alpha_j(v_m), \alpha_j(\text{DV}))\) where:

  • \(I_j = \alpha'_j|_{\text{DV}}\) : from construction of \(M(S)\) and \(\alpha'_0|_{\text{Lstore}} = \alpha_0\) and \(\alpha'_0|_{\text{DV}} = I_0\), \(\text{DV}\) is modified regarding to updates made by \(p\), where the values of variables depends on inputs and the database.

  • \(\alpha_j|_{\text{Lstore} - \{v_1, ..., v_m\}} = \alpha'_j|_{\text{Lstore} - \{v_1, ..., v_m\}}\) because \(\alpha'_j|_{\text{Lstore} - \{v_1, ..., v_m\}} = \alpha_j|_{\text{Lstore} - \{v_1, ..., v_m\}}\) and \(\alpha'_j|_{\text{Lstore} - \{v_1, ..., v_m\}} = \alpha'_j|_{\text{Lstore} - \{v_1, ..., v_m\}}\).

  • \(\alpha_j|_{\{v_1, ..., v_m\}} = \alpha'_j|_{\{v_1, ..., v_m\}}\) : From the construction of \(p_v\) and \(\alpha'_0|_{\text{DV}} = I_0\) and \(\alpha_j = \alpha_j\).
1. Suppose \((q_i, \alpha'_i) \in \Delta_{\mathcal{M}(S)}\), and there exists \((q_i, I_i, \alpha_i) \in \mathbb{Q}^k\) where \(\alpha'_{i|DV}\) = \(I_i\) and \(\alpha'_{i|Lstore} = \alpha_i\), then for each transition \((q_0, \alpha_0) \xrightarrow{\mu_0} (q_j, \alpha'_j)\), we know there exists a transition \((q_i, I_i, \alpha_i) \xrightarrow{\mu_i} (q_j, I_j, \alpha_j)\) in \(\Delta^k\) from the construction of \(\mathcal{M}(S)\) and \(1\) where:

- if \(\mu_i = m(\alpha_j(u_1), ..., \alpha_j(u_n))\), then \(\mu_i = m(\alpha_i(u_1), ..., \alpha_i(u_n))\) where:
  - \(\alpha_j = \alpha'_{j|Lstore}\) because \(\alpha'_i = \alpha'_j\) and \(\alpha_i = \alpha'_{i|Lstore}\) and \(\alpha_i = \alpha_j\).
  - \(I_j = \alpha'_{j|DV}\) because \(\alpha'_i = \alpha'_j\) and \(I_i = I_j\) and \(I_i = I'_i\).

- if \(\mu'_i = p(\alpha'_j(u_1), ..., \alpha'_j(u_n), \alpha'_j(DV_1), ..., \alpha'_j(DV_n))\), then \(\mu'_i = p(\alpha_i(u_1), ..., \alpha_i(u_n), \alpha_i(DV_1), ..., \alpha_i(DV_n), \alpha_j(DV_1), ..., \alpha_j(DV_n))\) where:
  - \(\alpha_j = \alpha'_{j|DV}\) from construction of \(\mathcal{M}(S)\) and \(\alpha'_{i|Lstore} = \alpha_i\) and \(\alpha'_{i|DV} = I_i\) and \(\mathcal{M}(S)\) are modified regarding to updates made by \(p\), where the values of variables depends on inputs and the database.
  - \(\alpha'_{0|Lstore} = \alpha'_{0|Lstore}\) because:
    - \(\alpha'_0 = \alpha'_{0|Lstore}\) and \(\alpha'_0 = \alpha'_{0|Lstore}\).
    - \(\alpha_j = \alpha'_{j|Lstore}\) because \(\alpha'_0 = I_0\) and \(\alpha_0 = \alpha_0\).

The two previous lemma show that there exists an equivalence between \(k\)-bounded extended state machine of a service \(S\) and the extended state machine of its \(\mathcal{M}(S)\) preserving the values of the \(Lstore\) and bounded databases. Now we can give the following Theorem.

**Theorem 6.** Let \(S, S'\) two Colombo services, then \(S \preceq_k S'\) iff \(\mathcal{M}(S) \preceq \mathcal{M}(S')\).

**Proof.** We will prove the two direction.

If \(E^k(S) \preceq E^k(S')\) then \(E(\mathcal{M}(S)) \preceq E(\mathcal{M}(S'))\): from definition of simulation and Lemma 8.

If \(E(\mathcal{M}(S)) \preceq E(\mathcal{M}(S'))\) then \(E^k(S) \preceq E^k(S')\): from definition of simulation and Lemma 9.

48
5.4 Complexity of k-Bounded simulation \( \preceq_k \)

In this section we will prove the 2-EXPTIME completeness of checking k-bounded simulation. First we will show the membership to 2-EXPTIME, then the 2-EXPTIME hardness is proven by reduction from the problem of infinite execution of an exponentially space bounded alternating Turing machine \( M \) (for an input word \( w \) of size \( n \), \( M \) can explore \( 2^n \) cells). The proof is similar to the reduction used for complexity of DB-less Colombo.

2-EXPTIME membership

**Proposition 2.** Let \( S_1 \preceq_k S_2 \) be test of k-Bounded simulation between two Colombo services. The region automata associated to \( S_1 \) and \( S_2 \) noted \( \mathcal{M}(S_1) \) and \( \mathcal{M}(S_2) \) have a double exponential size.

**Proof.** Let \( S_1 \) and \( S_2 \) be two Colombo services with \( C \) the set of constants and \( X \) the set of variables in \( S_1 \) and \( S_2 \). Testing \( S_1 \preceq_k S_2 \) is achieved by testing \( \mathcal{M}(S_1) \preceq \mathcal{M}(S_2) \), where \( \mathcal{M}(S_1) \) and \( \mathcal{M}(S_2) \) are two DB-less services. We suppose that \( n \) is the number of constants and \( m \times k \times l \) the number of variables, with \( l \) the arity of \( W \) and \( m \) number of variables in \( S_1, S_2 \). A region is a set of intervals and a \( v \)-order on \( X \cup \{ \omega \} \). The number of intervals (resp. \( v \)-orders) is bounded by \( O(n) \) (resp. \( O((m \times k \times l)) \)). The number of regions is bounded by \( O(n^{m \times k \times l} \times !(m \times k \times l)) \) and therefore the number of states in the region automata is bounded by \( O(2^{\log |Q_1| + \log n + (\log m \times \log k \times \log l)}) \).

We conclude that the size of the region automata associated to \( \mathcal{M}(S_1) \) and \( \mathcal{M}(S_2) \) are respectively \( O(2^{\log |Q_1| + \log n + (\log m \times \log l \times 2^{\log k})}) \) and \( O(2^{\log |Q_1| + \log n + (\log m \times \log l \times 2^{\log k})}) \).

2-EXPTIME hardness We will prove that given a Turing machine \( M \) working on an exponential space bounded by the size \( n \) of an input word \( w \), the machine has an infinite execution iff \( S_{sp} \preceq_k S_{defender} \), where \( k = 2^n \).

As for the reduction of an alternating Turing machine work on space bounded polynomially by the size \( n \) of the input \( w \), \( S_{defender} \) will encode exactly the execution of the machine, and \( S_{sp} \) encode all transitions that the machine can do, infinitely often. We start by some needed notations and give the construction of \( S_{defender} \). Then we will prove the correspondence between the execution of the machine and the execution of the service, where the machine has an infinite execution on the input \( w \) iff the service \( S_{defender} \) has an infinite execution. Finally the reduction to test of simulation \( S_{sp} \preceq S_{defender} \).

**Service** \( S_{defender} \) The machine \( M \) works on a tape of size \( 2^n \), the service \( S_{defender} \) will use \( R(A_1, \ldots, A_n; W_1) \) to encode the \( 2^n \) cells. The key is on \( n \) attributes. Taking the domain of \( A_i \) \( \{0, 1\} \), the key is a binary number on \( n \) position, so we can reach \( 2^n \) tuples where their keys are from \( (0, \ldots, 0) \) to \( (1, \ldots, 1) \).
To simulate the head, we will use \( n \) variables where the value of each variable is 0/1, then, the actual values of \( x_1, \ldots, x_n \) correspond to the binary number \( x_1 \ldots x_n \), so it points on one tuple of \( R \). The move on the right of the machine is made by incrementing \( x_1 \ldots x_n \), then the new binary number points on the next tuple, similarly the move on the left is made by decrementing \( x_1 \ldots x_n \) and the binary number points on the previous tuple. The attribute \( W \) is used to store the letter of the cell. Let the machine be in a configuration \( C \) and the head points on a cell containing \( a \) then \( M \) can execute \( q \rightarrow q' \) by writing \( b \) in the current cell and move the head on the right. This step will be executed by 3 transition in \( S_{\text{defender}} \). First test if the actual tuple contains \( a \), then write \( b \) in the attribute \( W \) of this tuple, and move to the next tuple by executing a binary addition on \( x_1 \ldots x_n \).

**Example 25.** Figure 25(a) depicts a transition of the machine \( M \) of example 18. Where if the actual value of the cell pointed by the head is equal to \( a \) and the machine is in state \( q_0 \), the machine write \( a \), and move to the next cell and reaches the state \( q_1 \). Suppose the input word is \( ab \), so \( n=2 \). The part of \( S_{\text{defender}} \) representing this transition, start by store the value of attribute \( W \) corresponding to the tuple identified with the key \( x_1 x_2 \) in the variable letter: \( \text{letter} := f^R_n((x_1, x_2)) \), using the atomic process \( \text{get cell} \). Then before writing in the current tuple the new value of \( W \) with \( \text{set cell} \), the service tests if \( \text{letter} = a \). After that the service increments the binary number \( x_1 x_2 \) using the atomic
process NEXT, as consequence \(x_1, x_2\) points on the next tuple, the guard \(\neg(x_1 = 1 \land x_2 = 1)\), prevent to move on the right if the service points on the last cell.

Service \(S_{\text{defender}}\) and \(S_{\text{spoiler}}\), start with an initialization part in two step:

1. First because services starts with arbitrary database, we need to check if all tuples identified with key from \((0, ..., 0)\) to \((1, ..., 1)\) contains the symbol \(B\), which means the \(2^n\) cells are empty, then
2. initialize the \(n\) first tuple with the \(n\) characters of the input word \(w\).

Example 26. Continuing with our example, Figure 26 depicts the initialization part, where the services start by affect zero to \(x_1\) and \(x_2\), then check if the value of attribute \(W\) of the actual tuple identified with the key \(x_1, x_2\) is equal to \(B\) if it is the case and it is note the last tuple (key equal 11), increment the key and test the next tuple. If one of them does not contain \(B\), then the database is not standard and there is simulation. If all tuples range from 00 to 11 contains \(B\), the services reinitialize the variables to zero.

For all execution starting with a non standard database, \(S_{\text{spoiler}} \preceq S_{\text{defender}}\), because the two service have the same initialization part. Figure 27 depicts examples of standards and non-standard databases. As we can see the order of tuples is not important for standard database (database a and b). The database c fail in initialization because \(f^3_S(1, 1)\) and \(f^3_S(0, 1)\) is equal to \(\omega\), and the database d is not standard because there is two tuples which their values of \(W\) is different from \(B\).

Now we will give the formal definition of atomic processes and \(S_{\text{defender}}\).
Definition 16. \( P \) is the set of all atomic processes used to encode execution of \( M \):

- for each transition \( q \xrightarrow{a/bR} q' \) in \( M \):
  
  - \textit{get} \( \text{cell} \) \( _{qabq}^{R}(x_1,\ldots,x_n;\text{letter},CE) \) is an atomic process with one conditional effect:
    
    * \( \theta = \text{True} \).

    * \( \text{ev} : \)
    
    - \( \text{letter} := f_{n+1}^{R}(x_1,\ldots,x_n) \).
  
  - \textit{set} \( \text{cell} \) \( _{qabq}^{R}(x_1,\ldots,x_n,b) \) is an atomic process with one conditional effect:
    
    * \( \theta = \text{True} \).

    * \( \text{ev} : \)
    
    - \( \text{MODIFY} R(x_1,\ldots,x_n;b) \).
  
  - \( \text{NEXT}(x_1,\ldots,x_n;x_1,\ldots,x_n;\{CE\}) \) is an atomic process with \( n \) conditional effect

  where \( CE = \{(\theta_n,v_n)\} \cup \{(\theta_k,v_k) \mid k \in [n-1,1]\} \) and

  * \( \theta_n : x_n = 0 \)

  * \( v_n : x_n := 1 \)

  * \( \theta_k : x_n = 1 \land x_{n-1} = 1 \land \ldots \land x_{k+1} = 1 \land x_k = 0 \)

  * \( v_k : x_n := 0 \land x_{n-1} := 0 \land \ldots \land x_{k+1} := 0 \land x_k := 1 \)

- for each transition \( q \xrightarrow{a/bL} q' \) in \( M 

  - \textit{get} \( \text{cell} \) \( _{qabq}^{L}(x_1,\ldots,x_n;\text{letter},CE) \) is an atomic process with one conditional effect:
    
    * \( \theta = \text{True} \).

    * \( \text{ev} : \)
\[
\text{let } \gamma := f^R_{n+1}(x_1, \ldots, x_n).
\]
\[
\text{set cell}_{qabq}^{\delta L}(x_1, \ldots, x_n, b) \text{ is an atomic process with one conditional effect:}
\]
\[
\text{\qquad \ast } \gamma = \text{ True.}
\]
\[
\text{\qquad \ast } \text{ ev:}
\]
\[
\text{\quad \quad \quad } \text{MODIFY } R(x_1, \ldots, x_n; b).
\]
\[
\text{PREVIOUS}(x_1, \ldots, x_n; x_1, \ldots, x_n; \{CE\}) \text{ is an atomic process with n conditional effect}
\]
\[
\text{\qquad where } CE = \{(\gamma_n, v_n) \} \cup \{(\gamma_k, v_k) \mid k \in [n-1, 1]\} \text{ and}
\]
\[
\text{\qquad \ast } \gamma_n; x_n := 1
\]
\[
\text{\qquad \quad } v_n := x_n := 0
\]
\[
\text{\qquad \ast } \gamma_k; x_n = 0 \land x_{n-1} = 0 \land \ldots \land x_{k-1} = 0 \land x_k = 1
\]
\[
\text{\qquad \quad } v_k := x_{n-1} := 1 \land \ldots \land x_{k-1} := 1 \land x_k := 0
\]

Let \(G_A(S_{\text{defender}}) = \langle Q_{\text{defender}}, \delta_{\text{defender}}, q_{\text{start}}, L\text{store}(S_{\text{defender}}) \rangle\) where:

\(- \text{ the set of states } S_{\text{defender}} \text{ are following}
\]
\[
\text{\qquad \ast } \text{ For each state } q \text{ in } M, \text{ a state } l_q.
\]
\[
\text{\qquad \ast } \text{ a set of states } l_{\text{copy}}, l_{\text{zero}}, l_{\text{init}}, l_{\text{fail}}. \text{ Where } l_{\text{fail}} \text{ is final.}
\]
\[
\text{\qquad \ast } \{\text{choice}_{qabq} \mid q \xrightarrow{a/bd} q' \text{ in } M \text{ and } q \text{ an existential state and } d = R/L\}
\]
\[
\text{\qquad \ast } \{l'_{qbdq}, l''_{qbdq} \mid q \xrightarrow{a/bd} q' \text{ in } M \text{ and } d = R/L\}
\]
\[
\text{\ast } q_{\text{start}} \text{ is the initial state.}
\]
\[
\text{\ast } \text{ for each } w_i \text{ (i'th letter of the input word) a state } l_{w_i}.
\]
\[
\text{\ast } L\text{store}(S_{\text{defender}}) = \{x_1, \ldots, x_n\} \cup \{\text{letter}\}.
\]
\[
\text{\ast } \delta_{\text{defender}} \text{ is composed of the following sets of transitions:}
\]
\[
\text{\qquad \ast } (l_{\text{start}}, \text{ True, init(} \emptyset; \text{letter, } x_1, \ldots, x_n), l_{\text{zero}}).
\]
\[
\text{\qquad \ast } (l_{\text{zero}}, \text{ True, get cell}(x_1, \ldots, x_n; \text{letter}), l_{\text{init}}).
\]
\[
\text{\qquad \ast } (l_{\text{init}}, \text{letter} = B \land \neg(x_1 = 1 \land \ldots \land x_n = 1), \text{ NEXT}(x_1, \ldots, x_n; x_1, \ldots, x_n),
\]
\[
\text{\qquad \qquad } l_{\text{zero}}).
\]
\[
\text{\qquad \ast } (l_{\text{init}}, \text{letter} \neq B, \text{ no-op}, l_{\text{fail}}).
\]
\[
\text{\qquad \ast } (l_{\text{init}}, \text{letter} = B \land x_1 = 1 \land \ldots \land x_n = 1, \text{ init(} \emptyset; \text{letter, } x_1, \ldots, x_n), l_{w_i}).
\]
\[
\text{\qquad \ast } (l_{\text{copy}}, \text{ True, } P \cup \{l_{\text{copy}}\}, l_{\text{copy}}), \text{ where } P \text{ is the set of all atomic process in } S_{\text{defender}}.
\]
\[
\text{\ast } \text{ for each } w_i \text{ (i'th letter of the input word) two transition:}
\]
\[
\text{\qquad \ast } (l_{w_i-1}, \text{ True, Insert}(x_1, \ldots, x_n, w_i), l_{w_i})
\]
\[
\text{\qquad \ast } (l_{w_i}, \text{ True, NEXT}(x_1, \ldots, x_n, l_{w_{i+1}})
\]
\[
\text{\ast } \text{ For each transition } q \xrightarrow{a/bd} q' \text{ in } M, \text{ where } q \text{ is a universal state:}
\]
\[
\text{\qquad \ast } \text{ if } d = R \text{ then :}
\]
\[
\text{\qquad \quad \ast } \{(l_q, \text{ True, get cell}_{qabq}^R(x_1, \ldots, x_n; \text{letter}), l'_{qbRq}^R)\}.
\]
\[
\text{\qquad \quad \ast } \{(l'_{qbRq}^R, \text{ letter} = a, \text{ set cell}_{qabq}^R(x_1, \ldots, x_n; b; \emptyset), l''_{qbRq}^R)\})
\]
\[
\text{\qquad \quad \ast } \{(l''_{qbRq}^R, \neg(x_1 = 1 \land \ldots \land x_n = 1), \text{ NEXT}(x_1, \ldots, x_n; x_1, \ldots, x_n, l_{q'})\}
\]
\[
\text{\qquad \quad \ast } \{(l_q, \text{ True, } l_{\text{copy}}), l_{\text{copy}}\}
\]
\[
\text{\qquad \ast } \{(l_q, \text{ True, } P \setminus P_{\text{test cell}}, l_{\text{copy}})\}, \text{ } P_{\text{test cell}} \text{ is the set of atomic process test cell used to encode transition from state } q \text{ of } M.
\]
The service starts with the initialization part, then reaches $l_q$. For a given transition \( q \xrightarrow{a/bd} q' \) in \( M \), where \( q \) is a existential state:

- if \( d = L \) then :
  - \( \{ (l_q, \text{True}, \text{get}_\text{cell}(q_{abq'} L(x_1, ..., x_n; \text{letter})), l'_{q RH}) \} \).
  - \( \{ (l'_{q RH}, \text{letter} = a, \text{set}_\text{cell}(q_{abq'} L(x_1, ..., x_n, b; \emptyset)), l''_{q RH} ) \} \).
  - \( \{ (l''_{q RH}, \neg(x_1 = 0 \land ... \land x_n = 0), \text{Previous}(x_1, ..., x_n; l'_{q}) \} \).
  - \( \{ (l_q, \text{True}, \text{P}(l_{\text{copy}})) \} \).
  - \( \{ (l_q, \text{True}, \text{P} \setminus \text{P}_{\text{test}_\text{cell}, l_{\text{copy}}}) \} \).
- if \( d = R \) then :
  - \( \{ (l_q, \text{True}, \text{I}(\text{copy})) \} \).
  - \( \{ (l_q, \text{True}, \text{P} \setminus \{ l_{\text{copy}} \}, l'_{q RH}) \} \).
  - \( \{ (\text{choice}_{q RH}, \text{True}, \text{get}_\text{cell}(q_{abq'} R(x_1, ..., x_n; \text{letter})), l'_{q RH}) \} \).
  - \( \{ (l'_{q RH}, \text{letter} = a, \text{set}_\text{cell}(q_{abq'} R(x_1, ..., x_n, b)), l''_{q RH} ) \} \).
  - \( \{ (l''_{q RH}, \neg(x_1 = 1 \land ... \land x_n = 1), \text{Next}(x_1, ..., x_n; l'_{q}) \} \).
  - \( \{ (\text{choice}_{q RH}, \text{True}, \text{P} \setminus \{ \text{get}_\text{cell}(q_{abq'} R(x_1, ..., x_n; \text{letter})), l_{\text{copy}} \}) \} \).
- if \( d = L \) then :
  - \( \{ (l_q, \text{True}, \text{I}(\text{copy})) \} \).
  - \( \{ (l_q, \text{True}, \text{P} \setminus \{ l_{\text{copy}} \}, l'_{q LH}) \} \).
  - \( \{ (\text{choice}_{q LH}, \text{True}, \text{get}_\text{cell}(q_{abq'} L(x_1, ..., x_n; \text{letter})), l'_{q LH}) \} \).
  - \( \{ (l'_{q LH}, \text{letter} = a, \text{set}_\text{cell}(q_{abq'} L(x_1, ..., x_n, b)), l''_{q LH} ) \} \).
  - \( \{ (l''_{q LH}, \neg(x_1 = 0 \land ... \land x_n = 0), \text{Previous}(x_1, ..., x_n; l'_{q}) \} \).
  - \( \{ (\text{choice}_{q LH}, \text{True}, \text{P} \setminus \{ \text{get}_\text{cell}(q_{abq'} L(x_1, ..., x_n; \text{letter})), l_{\text{copy}} \}) \} \).

The service starts with the initialization part, then reaches $l_q$, where $q$ is the initial state of $M$. For a given transition \( q \xrightarrow{a/bd} q' \) in $M$, then the service $S_{\text{defender}}$ has three transitions representing the transition of $M$, if $q$ is existential, the service send the message $m()$, then execute the transitions.

**Service $S_{\text{spoiler}}$ :** Let $GA(S_{\text{spoiler}}) = \langle Q_{\text{spoiler}}, \delta_{\text{spoiler}}, q_{\text{start}}, LStore(S_{\text{spoiler}}) \rangle$ where:

- the set of states of $S_{\text{defender}}$ are following
  - a set of states $l_{\text{start}}, l_{\text{zero}}, l_{\text{init}}, l_{\text{fail}}, l_{\text{fail}}$, $l_{\text{finally}}$. Where $l_{\text{final}}$ is final.
  - $\{ l_{\text{get}}' \}$ \( q \xrightarrow{a/bd} q' \) in $M$
  - $q_{\text{start}}$ is the initial state.
- for each $w_i$ (i'th letter of the input word) a state $l_{w_i}$.
- $Lstore(S_{\text{defender}}) = \{ x_1, ..., x_n \} \cup \{ \text{letter} \}$
- $\delta_{\text{defender}}$ is composed of the following sets of transitions:

54
The part of initialization of the input word and checking the database is the same as in $S_{defender}$, the difference is after storing the last letter of input word $w$, $S_{spoiler}$ go to state $l_y$.

$(l_y, True, !m(), l_3)$.

For each transition $q' \xrightarrow{a/bd} q'$ in $M$, where $q$ is a universal state:

* if $d = R$ then :
  
  $\cdot \{ (l_y, True, get\_cell_{qabq'} R(x_1, ..., x_n; letter), l_{qbdq'} ) \}.$
  
  $\cdot \{ (l_{qbdq'}, True, set\_cell_{qabq'} R(x_1, ..., x_n, b; \emptyset), l_{qbdq'} ) \}.$
  
  $\{ (l'_{qbdq'}, \neg(x_1 = 1 \land ... \land x_n = 1), \text{NEXT}(x_1, ..., x_n; x_1, ..., x_n), l_y ) \}.$

* if $d = L$ then :
  
  $\cdot \{ (l_y, True, get\_cell_{qabq'} L(x_1, ..., x_n; letter), l_{qbdq'} ) \}.$
  
  $\cdot \{ (l_{qbdq'}, True, set\_cell_{qabq'} L(x_1, ..., x_n, b; \emptyset), l_{qbdq'} ) \}.$
  
  $\{ (l'_{qbdq'}, \neg(x_1 = 0 \land ... \land x_n = 0), \text{Previous}(x_1, ..., x_n; x_1, ..., x_n), l_y ) \}.$

For each transition $q' \xrightarrow{a/bd} q'$ in $M$, where $q$ is an existential state:

* if $d = R$ then :
  
  $\cdot \{ (l_3, True, get\_cell_{qabq'} R(x_1, ..., x_n; letter), l_{qbdq'} ) \}.$
  
  $\cdot \{ (l_{qbdq'}, True, set\_cell_{qabq'} R(x_1, ..., x_n, b; \emptyset), l_{qbdq'} ) \}.$
  
  $\{ (l'_{qbdq'}, \neg(x_1 = 1 \land ... \land x_n = 1), \text{NEXT}(x_1, ..., x_n; x_1, ..., x_n), l_y ) \}.$

* if $d = L$ then :
  
  $\cdot \{ (l_3, True, get\_cell_{qabq'} L(x_1, ..., x_n; letter), l_{qbdq'} ) \}.$
  
  $\cdot \{ (l_{qbdq'}, True, set\_cell_{qabq'} L(x_1, ..., x_n, b; \emptyset), l_{qbdq'} ) \}.$
  
  $\{ (l'_{qbdq'}, \neg(x_1 = 1 \land ... \land x_n = 1), \text{PREVIOUS}(x_1, ..., x_n; x_1, ..., x_n), l_y ) \}.$

Lemma 10. Given an alternating Turing machine $M$ work on space exponentially bounded by the input $w$, $M$ has an infinite execution on $w$ iff $S_{defender}$ has an infinite execution without passing through the state $l_{copy}$.

Proof. This lemma show the connection between each configuration of $M$ and id of $E(S_{defender})$. Here a configuration $C$ of $M$ is $y_1, ..., qy_j, ..., y_{2^n}$, where the head points on the j'th cell.

Suppose the initial configuration of $M$ is $C_0=qy_1, ..., y_{2^n}$ and $q$ universal, from construction of $S_{defender}$, after checking the database and initializing $n$ first tuples with the word $w$, $E(S_{defender})$ is in id=$(l_q, \alpha(Lstore), I)$, where the binary number $\alpha(x_1)...\alpha(x_n)$ points on the first tuple and $f_n^{R+1}(\alpha(x_1), ..., \alpha(x_n))=y_1$. Suppose $C_0 \xrightarrow{qabq'} (C')$ exists, that means $y_1=a$ in $C_0$, so $f_n^{R+1}(\alpha(x_1), ..., \alpha(x_n))=a$. From construction of $S_{defender}$ there is a transition $l_q \xrightarrow{\text{get\_cell}_{qabq'} (x_1, ..., x_n; letter)} l'_{qbdq'}$, so id $\xrightarrow{\text{get\_cell}_{qabq'} (x_1, ..., x_n; letter)}$.
id' exist where letter in id' is equal to a. There is also a transition 

\[ l'_{qbdq'} \overset{\text{letter}=a}{\longrightarrow} \text{set cell}_{qabq'} R(x_1,\ldots,x_n;b;\emptyset) \]

id'' because letter=a. \( y_1 \) in \( C' \) is equal to \( b \) and the head point in the second cell. \( f_{n+1}^R(x_1,\ldots,x_n)=b \) in id''. In \( S_{\text{defender}} \) there is a transition 

\[ l'_{qbdq'} \overset{x_1=1\wedge\ldots\wedge x_n=1}{\longrightarrow} \text{NEXT}(x_1,\ldots,x_n;\ldots,\ldots) \]

\( l'_{qbdq} \). We can conclude that \( C' \) is encoded in id'''.

If \( C_0 \xrightarrow{q/a} C' \) does not exist, then letter is different from a in id' and the condition of \( l'_{qbdq'} \overset{\text{letter}=a}{\longrightarrow} \text{set cell}_{qbdq'} R(x_1,\ldots,x_n;\emptyset) \) is not verified so the execution of \( S_{\text{defender}} \) blocks.

Now suppose the execution of \( M \) is in configuration \( C = y_1,\ldots,qy_j,\ldots,y_{2^n} \) where \( q \) is universal and there exists an id in \( E(S_{\text{defender}}) \) where the binary number \( x_1,\ldots,x_n \) is equal to \( j \) and the control state is \( l_q \). Suppose \( C \xrightarrow{qabRq} C' \) exists, that means \( y_j=a \) in \( C \), so \( f_{n+1}^R(\alpha(x_1),\ldots,\alpha(x_n)) = a \). From construction of \( S_{\text{defender}} \) there is a transition 

\[ l_q \overset{\text{True \ get cell}_{qbdq'} R(x_1,\ldots,x_n;\text{letter})}{\longrightarrow} l'_{qbdq} \]

so id 

\[ \text{set cell}_{qbdq'} R(x_1,\ldots,x_n;\emptyset) \]

id'' because letter=a. \( y_j \) in \( C' \) is equal to \( b \) and the head point in the \( j+1 \)th cell. \( f_{n+1}^R(x_1,\ldots,x_n)=b \) in id'' in \( S_{\text{defender}} \) there is a transition 

\[ l'_{qbdq'} \overset{x_1=1\wedge\ldots\wedge x_n=1}{\longrightarrow} \text{NEXT}(x_1,\ldots,x_n;\ldots,\ldots) \]

\( l_q \) so there is a transition id'' 

\[ \text{NEXT}(x_1,\ldots,x_n;\ldots,\ldots) \]

id''' where the binary number \( x_1,\ldots,x_n \) in id''' is equal to \( j+1 \). We can conclude that \( C' \) is encoded in id'''.

From construction of \( S_{\text{defender}} \), after the part if initialisation, the execution of the service is in an id, with the control state \( l_q \) where \( q \) is the initial state of \( M \), suppose it is universal, all \( x_i \) are equal to zero, and the \( n \) first tuples contains the letters of the input word \( w \), then id correspond to \( C_0 \) in the execution of \( M \), and \( y_1=f_{n+1}^R(\alpha(x_1),\ldots,\alpha(x_n)) \). If there exists in \( E(S_{\text{defender}}) \) transitions id 

\[ \text{get cell}_{qbdq'} R(x_1,\ldots,x_n;\text{letter}) \]

id'' 

\[ \text{NEXT}(x_1,\ldots,x_n;\ldots,\ldots) \]

id''' then from construction of \( S_{\text{defender}} \), there exists a transition \( q \xrightarrow{a/bR} q' \) in M. We know also \( f_{n+1}^R(\alpha(x_1),\ldots,\alpha(x_n))=a \) in id and letter=a in d so \( y_1=a \) in C. Then \( C_0 \xrightarrow{qabRq} C' \). \( f_{n+1}^R(\alpha''(x_1),\ldots,\alpha''(x_n))=b \) in id''. Then \( y_1 \) in \( C' \) = b, and the head point on 2, because after executing next \( x_1,\ldots,x_n=1 \).

Suppose \( E(S_{\text{defender}}) \) contains an id where the control state correspond to an universal state of the machine and id correspond to a configuration.
C of the machine. Let a set of transition id \( \text{get}_{cell} \rightarrow id' \)
\( \text{set}_{cell} \rightarrow \text{id}'' \).
\( \text{NEXT} \rightarrow \text{id}''' \).
From construction of \( S_{\text{defender}} \), there is a transition \( q \rightarrow q' \) in \( M \).
\( y_j = f_{n+1}(a(x_1, ..., a(x_n)) \) in id, letter=a in d so \( y_j = a \), then
\( C_{abolish} \rightarrow C', f_{n+1}(a'' (x_1, ..., a''(x_n)) = b \) in \( id''' \).
Then \( y_j \) in \( C'=b \), and the head point on j+1, because after executing \( \text{NEXT} \),
the binary number \( x_1 ... x_n = j \).

The same reasoning is used where \( q \) is existential with the difference that
\( S_{\text{defender}} \) has an additional transition before executing \( \text{get}_{cell} \), it send the
message \( m(\) \), which does not change the values of the variables nor the database.
Also for the action labelled with \( L \), we just replace \( \text{NEXT} \) by \( \text{PREVIOUS} \).

**Lemma 11.** Given an alternating Turing machine \( M \) working in space bounded
by the size of the input \( w \), \( M \) has an infinite computation on \( w \) iff \( S_{\text{spoiler}} \prec \)
\( S_{\text{defender}} \).

**Proof.** \( S_{\text{spoiler}} \), \( S_{\text{defender}} \) start by checking the database and
initializing the \( n \) first tuple with the input word \( w \). If \( M \) has a
transition \( q \rightarrow \) \( q' \) and \( q \) universal, then \( S_{\text{spoiler}} \) has a loop :
\( l_q \rightarrow \) \( \text{get}_{cell} \rightarrow \) \( \text{set}_{cell} \rightarrow \) \( \text{letter} \rightarrow \) \( l_{qbdq} \). Then \( l_{qbdq}' \) \( l_{qbdq}'' \)
and the service \( S_{\text{defender}} \) contains transitions
\( \text{Next} \rightarrow \) \( l_{qbdq}'' \) if \( x_1 ... x_n \) points on a tuple
with value of \( W = a \). Suppose the condition is verified, then if \( S_{\text{spoiler}} \) chose to
execute any transition with label different from \( \text{get}_{cell} \), \( \text{set}_{cell} \), \( \text{letter} \),
\( S_{\text{defender}} \) wins the game by choosing the transition which reaches the state
\( l_{copy} \), if \( S_{\text{spoiler}} \) chooses to execute \( \text{get}_{cell} \), \( \text{set}_{cell} \), \( \text{letter} \),
then \( S_{\text{defender}} \) execute \( \text{get}_{cell} \), \( \text{set}_{cell} \), \( \text{letter} \) and it can execute
\( \text{set}_{cell} \), \( \text{letter} \) and the game continue. Now, suppose
the condition is not verified, then if \( S_{\text{spoiler}} \) chose to execute any transition
with label different from \( \text{get}_{cell} \), \( \text{set}_{cell} \), \( \text{letter} \), \( S_{\text{defender}} \) wins the game by choosing the transition which reaches the state
\( l_{copy} \), if \( S_{\text{spoiler}} \) chooses to execute \( \text{get}_{cell} \), \( \text{set}_{cell} \), \( \text{letter} \),
then \( S_{\text{defender}} \) execute \( \text{get}_{cell} \), \( \text{set}_{cell} \), \( \text{letter} \) but it can not execute
\( \text{set}_{cell} \), \( \text{letter} \), because the condition is not verified, so there is
not simulation.

If \( M \) has a transition \( q \rightarrow \) \( q' \) and \( q \) existental,
\( S_{\text{spoiler}} \) has transitions
\( l_q \rightarrow \) \( \text{True} \rightarrow \) \( m(\) \).

57
6.1 Automata over infinite domain

<table>
<thead>
<tr>
<th>Model</th>
<th>Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VA</td>
<td>EXPTIME</td>
</tr>
<tr>
<td>FA</td>
<td>EXPTIME</td>
</tr>
<tr>
<td>GVA</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>NFMA</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>Unbouded Colombo</td>
<td>Undecidable</td>
</tr>
<tr>
<td>DB - less Colombo</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>DB - bounded Colombo</td>
<td>2-EXPTIME-complete</td>
</tr>
</tbody>
</table>

Table 1. Result of simulation for automata over infinite domain
As shown in Table 6.1 there exists some results on simulation for data-aware services. We will detail each model and close some opening questions.

Colombo\(^{k,b}\) \cite{14} investigates the service composition problem using a very constrained class of Colombo, called Colombo\(^{k,b}\), which poses several restrictions (i) the number of access to database (ii) the number of new values incoming. As a consequence, Colombo\(^{k,b}\) is included in k-bounded Colombo service, where there is no cycle through a reception of message or modification database. The main result of \cite{14} is to show that service composition is 2-EXPTIME in Colombo\(^{k,b}\), which coincide with our upped bound for k-bounded Colombo. But the completeness still an open problem.

Variable automata \cite{30} introduce a new formalism called variable automata which we denote VA, a VA is a finite state machine where transition are labelled either with a constant, or variables. The set of variable in a VA is constituted with one free variable and bounded variables. During an execution, the values of the bounded variables is fixed (the value does not change during an execution). The value of free variable change each time the automata execute a transition labelled with it. Another constraint is that the value of free variable must be different from value of bounded variables and constants appearing in the automata. Also values of bounded variables must be different from constants appearing in automata.

FVA In \cite{10}, the authors define fresh-variable automata noted FVA, which is an automata where transition is labelled with constant or variables. Here during an execution the value of a variable change in specific states, called refresh states. Let \((q, x, q')\) be a transition and \(q'\) a refreshing state of the variable \(x\), then during an execution if actual value of the variable \(x\) is \(a\) and the automata is at state \(q\), it read \(a\) move to \(q'\) where the value of \(x\) changes and it is taken from an infinite domain. FVA are not comparable with VA owning to restrictions on values of free and bounded variables.

GVA In \cite{11}, they extend fresh-variable automata with guards on transitions, the model is called guarded variable automata (GVA), where a transition is of the form \((q, g, x, q')\) where \(g\) is a guard (conjunction of equality and inequality over variables and constants) and \(x\) a constant or a variable. They study the simulation for the two models. Note that, as mentioned by the authors VA are included in GVA. They also prove that the problem of simulation is EXPTIME for GVA. A GVA can be encoded in a DB-less Colombo service. But there subsist some differences with DB-less Colombo service: (i) in the domain which is not ordered and (ii) there is no atomic process, but the two facts does not affect the result on decidability and complexity of simulation, because an atomic process in DB-less service can be replaced by a reception of a message and guards, and the framework can be easily extended with an order on the infinite domain. They prove the decidability of simulation for guarded variables automata by proving the equivalence between test of simulation for infinite machines and the
test of simulation between two finite state machines. The simulation for GVA is EXPTIME-Complete, the membership was proved by the authors, which confirms our results on complexity of simulation for DB-LESS Colombo service, and we can use the same reduction from existing infinite execution for an Alternating Turing machine work on a space polynomially bounded by the size of its input to test of simulation between two GVA. They also prove the equivalence between GVA and NFMA [37] (Non-deterministic Finite Memory Automata). NFMA it is an automata with finite set of register or variables, the two notion are equivalent in the sense they contain one value at a given time of execution. NFMA contains two type of transition, either reading the content of a register, either refresh the value of a register with the constraint that value does not appear in other registers.

In [48], the authors study the composition problem for data-centric services using an approach based on the simulation relation. [48] considers also a very restrictive model where there is no notion of communication, neither variables, also the size of databases is bounded, and during an execution a service can only receive one parameter from the outside. The expressivity of the update language is also restricted, for example the following update can not be expressed in their model: \( \text{InsertR}(4, y) : \neg R(3, x) \land R(x, y) \), which made this model less expressive than k-bounded Colombo model. The authors show that, in this restricted context, service composition can be reduced into a simulation test between finite state machines. An interesting question is does the simulation remains decidable with remove the restriction on the size of database regarding to the expressivity of the updates.

7 Discussion

Several research works in the areas of component-based models and web services highlighted the benefits of the specification of the external visible behavior of the target software artefact [20,24,19,12,36]. In the web service area, a great deal of recent research efforts have been devoted to provide models and algorithms that can be exploited to facilitate service analysis and management. In the general area of formalizing Web service description, several models, targeting different objectives, have been proposed in the literature to describe services at different levels of abstraction [19,14,3,31,40]. Of particular interest, the need of models to describe external behavior of services has been highlighted in several papers (e.g., [19,15,13]).

Regarding the web service analysis dimension, recent approaches address the problem of verifying similarity and compatibility at different levels of abstractions of a service description (e.g., [9,17,24,51,53]) while verification and synthesis issues related to service composition attracted also a lot attention from the research community [16,19,25,29,26,14,52,7,15]. In most of the aforementioned research works, simulation preorder plays a fundamental role to solve the considered problems. For example, business protocol compatibility and
substitution are reducible to simulation between finite state machines. In it is shown that the unbounded variant of the protocol synthesis problem, i.e., when the number of instances of an available service that can be involved in a composition is not bounded a priori, can be recasted as a problem of deciding simulation between a finite state machine and an infinite state machine representing a shuffle closure of existing services.

In the formal verification area, simulation between finite state machines is well understood and several efficient algorithms have been proposed in the literature to cope with the underlying decision problem. The simulation between a finite state machine and infinite state machine has also been addressed in the literature. Depending on the considered infinite state machine (e.g. Petri nets), the simulation test in this context usually ranges from EXPTIME-complete to undecidable. The problem of checking simulation between two infinite state machine is in general undecidable and there are only very few classes, e.g., one-counter nets, where this problem is known to be decidable.

Verification of data-centric services and artifact-centric business processes attracted a lot of attention from the research community these recent years. In this context also the verification problem is undecidable in the general case. Existing works focus on identification of specific models and restrictions in which the verification problem can be solved.

Our future work will be devoted to the investigation of decidability and complexity of verification and composition of data-centric service in order to provide tight results regarding different restrictions on such models.

References


63