Verification of Data-Aware Processes

Data Centric Dynamic Systems

Diego Calvanese, Marco Montali

Research Centre for Knowledge and Data (KRDB)
Free University of Bozen-Bolzano, Italy

29th European Summer School in Logic, Language, and Information
(ESSLLI 2017)
Toulouse, France – 17–28 July 2017

Outline

Formalization of Data Centric Dynamic Systems

A **DCDS** $\mathcal{X}$ is a triple $\langle \mathcal{D}, \mathcal{P}, \mathcal{I}_0 \rangle$, where:

- $\mathcal{D}$ is the data layer, capturing the static aspects of the DCDS;
- $\mathcal{P}$ is the process layer, capturing the dynamic aspects of the DCDS;
- $\mathcal{I}_0$ is the initial database.

The **data layer** $\mathcal{D}$ is a pair $\langle \mathcal{R}, \mathcal{C} \rangle$, where:

- $\mathcal{R}$ is a relational schema, i.e., a set of relation schemas (relation names with their arity).
- $\mathcal{C}$ is a set of integrity constraints.
  We assume the constraints to be domain-independent FOL sentences.

The **initial database** $\mathcal{I}_0$ is a relational DB conforming to the data layer, i.e.:

- it is a database for the relational schema $\mathcal{R}$, and
- it satisfies the constraints $\mathcal{C}$.

Note: having an initial DB is an assumption that we make to ensure good computational properties (notably, decidability) of verification in our framework.

The process layer of DCDSs

The **process layer** $\mathcal{P}$ is a triple $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \varrho \rangle$, where:

- $\mathcal{F}$ is a finite set of functions, each representing the interface to an external service.
  - The services can be called, and as a result the function is activated and the answer is produced.
  - How the result is actually computed is unknown to the DCDS since the services are external.
- $\mathcal{A}$ is a finite set of actions, whose execution updates the data layer, and may involve external service calls.
- ϱ is a finite set of condition-action rules that form the specification of the overall process, which tells at any moment which actions can be executed.

Actions and process

An **action** α is a triple constituted by:

- A action name $\alpha{\cdot}\mathrm{NAME}$;
- A list $\alpha{\cdot}\mathrm{PARS}$ of individual input parameters;
  - Parameters are substituted by constants for the execution of the action.
- An effect specification $\alpha{\cdot}\mathrm{EFF}$ consisting of a set of effects.
  - Effects are assumed to take place simultaneously.

The **process** ϱ is a finite set of condition-action rules.

Each condition-action rule has the form $Q \mapsto \alpha$, where:

- α is an action in $\mathcal{A}$;
- Q is a FO query over $\mathcal{R}$:
  - the free variables are exactly the parameters of α;
  - the other terms in the query can be either quantified variables or constants in $\mathrm{ADOM}(\mathcal{I}_0)$.

Action effects

Each **effect** $e \in \alpha\cdot\textsc{eff}$ has the form $Q_e \rightsquigarrow$ **add** A **del** D where:

- Q_e is a query over $\mathcal{R}$ that may include as terms of the query:
  - constants of $\textsc{adom}(\mathcal{I}_0)$,
  - some of the input parameters.
- A and D are two sets of facts over $\mathcal{R}$, respectively to be added to and deleted from the current state to produce a new state.

  They may include as terms:
  - constants in $\textsc{adom}(\mathcal{I}_0)$, parameters, free variables of Q_e, and
  - **functions calls**, which formalize call to (atomic) **external services**.

Note:

- All queries (in the action effects, and in the condition of the process rules) are evaluated over the active domain $\textsc{adom}(\mathcal{I})$ of the current database.
  $\rightsquigarrow$ **Active domain semantics**.
- Hence the queries instantiate the facts to add and delete in the action effects with known values only.

However, new values are **introduced by the calls to external services**!

Deterministic vs. non-deterministic services

DCDSs admit two different semantics for service-execution:

Deterministic services semantics

Along a run, when the same service is called again with the same arguments, it returns the same result as in the previous call.

Are used to model an environment whose behavior is completely determined by the parameters.

Example: temperature, given the location and the date and time

Non-deterministic services semantics

Along a run, when the same service is called again with the same arguments, it may return a different value than in the previous call.

Are used to model:

- an environment whose behavior is determined by parameters that are outside the control of the system;
- input of external users, whose choices depend on external factors.

Example: current temperature, given the location

An example: Hotels and price conversion

Data Layer: Info about room prices for hotels and their currency

$$\mathsf{Cur} = \langle \mathit{UserCurrency} \rangle \qquad \mathsf{CH} = \langle \underline{\mathit{Hotel}}, \mathit{Currency} \rangle$$

$$\mathsf{PEntry} = \langle \underline{\mathit{Hotel}}, \mathit{Price}, \underline{\mathit{Date}} \rangle$$

Process Layer/1: User selection of a currency

- Process: $\mathit{true} \longmapsto \mathsf{ChooseCur}()$
- Service call for currency selection: $\textsc{uInputCurr}()$
  - Models user input with **non-deterministic** behavior.
- $\mathsf{ChooseCur}() : \left\{ \begin{array}{r} \mathsf{Cur}(c) \rightsquigarrow \textbf{del}\{\mathsf{Cur}(c)\} \\ \mathit{true} \rightsquigarrow \textbf{add}\{\mathsf{Cur}(\textsc{uInputCurr}())\} \end{array} \right\}$

An example: Hotels and price conversion

Data Layer: Info about room prices for hotels and their currency

$$\mathsf{Cur} = \langle UserCurrency \rangle \qquad \mathsf{CH} = \langle \underline{Hotel}, Currency \rangle$$
$$\mathsf{PEntry} = \langle \underline{Hotel}, Price, \underline{Date} \rangle$$

Process Layer/2: Price conversion for a hotel

- Process: $\mathsf{Cur}(c) \wedge \mathsf{CH}(h, c_h) \wedge c_h \neq c \longmapsto \mathsf{ApplyConv}(h, c)$
- Service call for currency selection: $\textsc{conv}(price, from, to, date)$
  - Models historical conversion with deterministic behavior.
- $\mathsf{ApplyConv}(h, c)$:

$$\left\{\begin{array}{c} \mathsf{PEntry}(h,p,d) \rightsquigarrow \textbf{del}\{\mathsf{PEntry}(h,p,d)\} \\ \mathsf{PEntry}(h,p,d) \wedge \\ \mathsf{CH}(h, c_{old}) \rightsquigarrow \textbf{add}\{\mathsf{PEntry}(h, \textsc{conv}(p, c_{old}, c, d), d)\} \\ \mathsf{CH}(h, c_{old}) \rightsquigarrow \textbf{del}\{\mathsf{CH}(h, c_{old})\},\ \textbf{add}\{\mathsf{CH}(h, c)\} \end{array}\right\}$$

Outline

Execution semantics of dynamic systems

Typically given in the form of a **transition system**.

(Propositional) transition system

Given a set Σ of state propositions, a (propositional) transition system is a tuple $\langle S, s_0, prop, \Rightarrow \rangle$, where:

- S is a **finite** set of states;
- $s_0 \in S$ is the initial state;
- $prop : S \to 2^\Sigma$ is an assignment, mapping each state in S to the set of propositions from Σ holding in that state;
- $\Rightarrow \subseteq S \times S$ is the transition relation between states.

Usually, the transitions are labeled with corresponding **actions**.

Impact of data on verification

The presence of data complicates verification significantly:

- **States** must be **modeled relationally** rather than propositionally.
- The resulting transition system is typically **infinite state**.
- Query languages for analysis need to combine two dimensions:
  - a temporal dimension to query the process execution flow, and
  - a first-order dimension to query the data present in the relational structures.

$\rightsquigarrow$ We need **first-order variants of temporal logics**.

What if the system evolves a database?

We get a transition system in which each state is a relational database.

In the following, we fix an infinite domain Δ of data items (also called values).

Relational transition system (RTS)

Is a tuple $\langle \Delta, \mathcal{R}, S, s_0, db, \Rightarrow \rangle$, where:

- $\mathcal{R}$ is a database schema;
- S is a **possibly infinite** set of states;
- $s_0 \in S$ is the initial state;
- db is a function that, given a state $s \in S$, returns the database instance $db(s)$ over $\mathcal{R}$ and Δ;
- $\Rightarrow \subseteq S \times S$ is the transition relation between states.

Execution semantics of a DCDS

Is determined by the **relational transition system** that accounts for all possible runs of the DCDS:

- States **are** database instances (i.e., db is the identity function).
- Transitions: correspond to *legal* applications of an action with parameter instantiation + service call evaluations.
  - Action with param. instantiation: executable according to the process rules.
  - Satisfaction of constraints ensured by each DB instance.

We obtain a possibly **infinite-state** (relational) transition system, intuitively constructed as follows:

- start from the initial DB;
- apply transitions in all possible ways;
- continue (ad infinitum) on the newly obtained states.

Outline

Action execution – Intuition

To **execute an action** α in state s according to $Q \mapsto \alpha$:

1. Evaluate Q over $db(s)$, and choose a parameters assignment σ for α.
2. If such an assignment exists, then α is executable and instantiated with σ.
3. $\alpha\sigma$ is executed: for each effect $Q_i \rightsquigarrow$ **add** A_i **del** D_i of α:
    1. $Q_i\sigma$ is evaluated over $db(s)$, getting variable assignments $\rho_1, \ldots, \rho_n$;
    2. for each ρ_j, the grounded facts $A_i\sigma\rho_j$ and $D_i\sigma\rho_j$ are obtained;
    3. all service calls contained in all $A_i\sigma\rho_j$ and $D_i\sigma\rho_j$ are issued;
    4. the next state is obtained by removing from the current state all facts $D_i\sigma\rho_j$ and adding all facts $A_i\sigma\rho_j$, after the inclusion of all service call results.

We now define the semantics formally.

Semantics of an action execution (1/3)

Consider:

- a DCDS $\mathcal{X} = \langle \mathcal{D}, \mathcal{P}, \mathcal{I}_0 \rangle$, where $\mathcal{D} = \langle \mathcal{R}, \mathcal{C} \rangle$ and $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \varrho \rangle$;
- the current instance $\mathcal{I}$ of $\mathcal{X}$, which is a DB state for $\mathcal{R}$ satisfying $\mathcal{C}$;
- an action $\alpha \in \mathcal{A}$;
- a parameter substitution σ assigning to $\alpha \cdot \text{PARS}$ actual parameters from Δ.

We want to compute the possible new instances $\mathcal{I}'$ obtained from $\mathcal{I}$ by executing action α under substitution σ.

We need to deal with **incomplete instances** $\overline{\mathcal{I}}$, which may contain service calls.

We define basic functions capturing the execution in $\mathcal{I}$ of α under σ:

- $\text{DEL}(\mathcal{I}, \alpha, \sigma)$ computes the (incomplete) facts to delete from $\mathcal{I}$;
- $\text{ADD}(\mathcal{I}, \alpha, \sigma)$ computes the (incomplete) facts to add to $\mathcal{I}$;
- $\text{CALLS}(\overline{\mathcal{I}})$ denotes the service calls contained in an incomplete instance $\overline{\mathcal{I}}$;
- $\text{EVALS}_{\Delta}(\overline{\mathcal{I}})$ denotes the set of substitutions that replace all service calls in $\overline{\mathcal{I}}$ with values from Δ.

Semantics of an action execution (2/3)

To compute a possible new instance $\mathcal{I}'$ from the current instance $\mathcal{I}$:

1. Compute the incomplete facts $\overline{\mathcal{I}}_{del} = \textsc{del}(\mathcal{I}, \alpha, \sigma)$ (for deletion), where:

$$\textsc{del}(\mathcal{I}, \alpha, \sigma) = \bigcup_{(Q_i \rightsquigarrow \textbf{add } A_i \ \textbf{del } D_i) \in \alpha\cdot\textsc{eff}} \ \bigcup_{\rho \in ans(Q_i\sigma, \mathcal{I})} D_i \sigma \rho$$

2. Compute the incomplete facts $\overline{\mathcal{I}}_{add} = \textsc{add}(\mathcal{I}, \alpha, \sigma)$ (for addition), where:

$$\textsc{add}(\mathcal{I}, \alpha, \sigma) = \bigcup_{(Q_i \rightsquigarrow \textbf{add } A_i \ \textbf{del } D_i) \in \alpha\cdot\textsc{eff}} \ \bigcup_{\rho \in ans(Q_i\sigma, \mathcal{I})} A_i \sigma \rho$$

3. Compute the set of substitutions for the service calls in $\overline{\mathcal{I}} = \overline{\mathcal{I}}_{del} \cup \overline{\mathcal{I}}_{add}$:

$$\textsc{evals}_{\Delta}(\overline{\mathcal{I}}) = \{\theta \mid \theta \text{ is a total function, } \theta : \textsc{calls}(\overline{\mathcal{I}}) \to \Delta\}$$

4. A possible new instance is $\mathcal{I}' = \textsc{do}(\mathcal{I}, \alpha, \sigma, \theta)$ for $\theta \in \textsc{evals}_{\Delta}(\overline{\mathcal{I}})$, where:

$$\textsc{do}(\mathcal{I}, \alpha, \sigma, \theta) = (\mathcal{I} \setminus \overline{\mathcal{I}}_{del}\theta) \cup \overline{\mathcal{I}}_{add}\theta$$

Note: additions globally take preference over deletions.

Semantics of an action execution (3/3)

- Each $\mathcal{I}' = \text{DO}(\mathcal{I}, \alpha, \sigma, \theta)$, for each $\theta \in \text{EVALS}_\Delta(\overline{\mathcal{I}})$, represents a potential new state resulting from the execution of α under parameter instantiation σ.
- However, $\mathcal{I}'$ must also satisfy the constraint of the data layer.

Legal new state

We say that $\mathcal{I}'$ is **legal** w.r.t. state $\mathcal{I}$, action α, parameter substitution σ, and service evaluation θ, if the following holds:

1. There exists a condition-action rule $Q \mapsto \alpha$ in ϱ such that $\sigma \in ans(Q, \mathcal{I})$.
2. The new state $\mathcal{I}'$ satisfies all constraints in $\mathcal{C}$.

To understand the possible executions of DCDSs, we have to clarify how external services behave across runs.

An example: Hotels and price conversion

Data Layer: Info about room prices for hotels and their currency

$$\mathsf{Cur} = \langle \mathit{UserCurrency} \rangle \qquad \mathsf{CH} = \langle \underline{\mathit{Hotel}}, \mathit{Currency} \rangle$$
$$\mathsf{PEntry} = \langle \underline{\mathit{Hotel}}, \mathit{Price}, \underline{\mathit{Date}} \rangle$$

Process Layer/1: User selection of a currency

- Process: $\mathit{true} \longmapsto \mathsf{ChooseCur}()$
- Service call for currency selection: $\textsc{uInputCurr}()$
  - Models user input with **non-deterministic** behavior.
- $\mathsf{ChooseCur}()$: $\left\{\begin{array}{r} \mathsf{Cur}(c) \rightsquigarrow \textbf{del}\{\mathsf{Cur}(c)\} \\ \mathit{true} \rightsquigarrow \textbf{add}\{\mathsf{Cur}(\textsc{uInputCurr}())\} \end{array}\right\}$

An example: Hotels and price conversion

Data Layer: Info about room prices for hotels and their currency

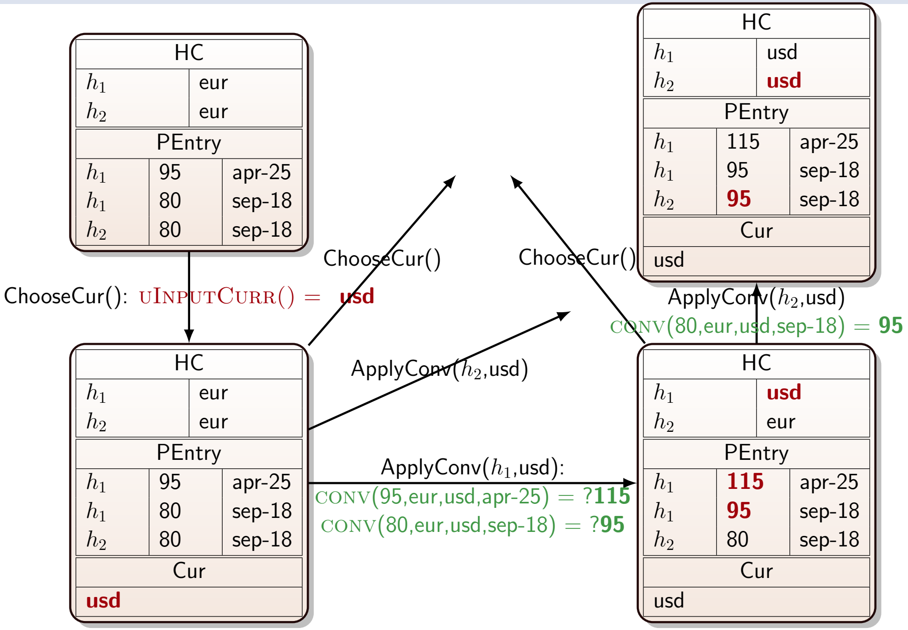
$$\mathsf{Cur} = \langle \mathit{UserCurrency} \rangle \qquad \mathsf{CH} = \langle \underline{\mathit{Hotel}}, \mathit{Currency} \rangle$$
$$\mathsf{PEntry} = \langle \underline{\mathit{Hotel}}, \mathit{Price}, \underline{\mathit{Date}} \rangle$$

Process Layer/2: Price conversion for a hotel

- Process: $\mathsf{Cur}(c) \land \mathsf{CH}(h, c_h) \land c_h \neq c \longmapsto \mathsf{ApplyConv}(h, c)$
- Service call for currency selection: $\textsc{conv}(\mathit{price}, \mathit{from}, \mathit{to}, \mathit{date})$
  - Models historical conversion with deterministic behavior.
- $\mathsf{ApplyConv}(h, c)$:

$$\left\{ \begin{array}{r} \mathsf{PEntry}(h,p,d) \rightsquigarrow \textbf{del}\{\mathsf{PEntry}(h,p,d)\} \\ \mathsf{PEntry}(h,p,d) \land \qquad\qquad\qquad\qquad\qquad\qquad \\ \mathsf{CH}(h,c_{old}) \rightsquigarrow \textbf{add}\{\mathsf{PEntry}(h, \textsc{conv}(p, c_{old}, c, d), d)\} \\ \mathsf{CH}(h,c_{old}) \rightsquigarrow \textbf{del}\{\mathsf{CH}(h,c_{old})\},\ \textbf{add}\{\mathsf{CH}(h,c)\} \end{array} \right\}$$

Run of the system



HC	
h_1	eur
h_2	eur

PEntry		
h_1	95	apr-25
h_1	80	sep-18
h_2	80	sep-18

ChooseCur(): uInputCurr() = **usd**

HC	
h_1	eur
h_2	eur

PEntry		
h_1	95	apr-25
h_1	80	sep-18
h_2	80	sep-18

Cur
usd

ChooseCur()
ApplyConv(h_2,usd)
ApplyConv(h_1,usd):
CONV(95,eur,usd,apr-25) = ?**115**
CONV(80,eur,usd,sep-18) = ?**95**
ChooseCur()

HC	
h_1	**usd**
h_2	eur

PEntry		
h_1	**115**	apr-25
h_1	**95**	sep-18
h_2	80	sep-18

Cur
usd

ApplyConv(h_2,usd)
CONV(80,eur,usd,sep-18) = **95**

HC	
h_1	usd
h_2	**usd**

PEntry		
h_1	115	apr-25
h_1	95	sep-18
h_2	**95**	sep-18

Cur
usd

Outline

Execution semantics for non-deterministic services

The above semantics for action execution provides directly the execution semantics for a DCDS $\mathcal{X}$ with non-deterministic services.

Transition relation $\textsc{n-exec}_{\mathcal{X}}$ over states

Captures how states evolve:

$$\langle \mathcal{I}, \alpha\sigma, \mathcal{I}' \rangle \in \textsc{n-exec}_{\mathcal{X}}$$

if and only if

there exists $\theta \in \textsc{evals}_{\Delta}(\textsc{del}(\mathcal{I}, \alpha, \sigma) \cup \textsc{add}(\mathcal{I}, \alpha, \sigma))$ such that $\mathcal{I}' = \textsc{do}(\mathcal{I}, \alpha, \sigma, \theta)$ is legal w.r.t. $\mathcal{I}$, α, σ, θ.

Note:

- The condition-action rules of ϱ are taken into account in $\textsc{n-exec}_{\mathcal{X}}$.
- States that do not satisfy the constraints in the data layer of the DCDS are not considered.
- There is no condition on how service calls evaluate across different states.
- However, within one execution step, all occurrences of the same service call evaluate to the same result.

Concrete transition system under non-deterministic services

Making use of the transition relation $\text{N-EXEC}_{\mathcal{X}}$, we can now define the transition systems generated by a DCDS under non-deterministic services semantics.

Concrete RTS for a DCDS $\mathcal{X} = \langle \Delta, \mathcal{D}, \mathcal{P}, \mathcal{I}_0 \rangle$ under non-deterministic services

Is an RTS $\Upsilon^n_{\mathcal{X}} = \langle \Delta, \mathcal{R}, S, s_0, db, \Rightarrow_n \rangle$, where:

- $s_0 = \mathcal{I}_0$;
- db is the identity function, i.e., $db(\mathcal{I}) = \mathcal{I}$;
- $\mathcal{R}$ is the database schema of $\mathcal{X}$;
- the states S and the transition relation $\Rightarrow_n$ are defined by simultaneous induction as the smallest set satisfying the following properties:
  1. $s_0 \in S$;
  2. if $\mathcal{I} \in S$, then for all actions α, parameter substitutions σ, and states $\mathcal{I}'$ such that $\langle \mathcal{I}, \alpha\sigma, \mathcal{I}' \rangle \in \text{N-EXEC}_{\mathcal{X}}$, we have that $\mathcal{I}' \in S$ and $\mathcal{I} \Rightarrow_n \mathcal{I}'$.

Execution semantics under deterministic services

To ensure that service calls behave deterministically, the states of the transition system keep track of all results of the service calls made so far.

- Let $\mathbb{SC} = \{f(v_1, \ldots, v_n) \mid f \in \mathcal{F} \text{ and } \{v_1, \ldots, v_n\} \subseteq \Delta\}$ be the set of (Skolem terms representing) service calls.
- A **service call map** is a partial function $\mathcal{M} : \mathbb{SC} \to \Delta$.
- A **state** of the transition system **is a pair** $\langle \mathcal{I}, \mathcal{M} \rangle$, where:
  - $\mathcal{I}$ is a relational instance for $\mathcal{D}$, and
  - $\mathcal{M}$ is a service call map.

Transition relation $\text{D-EXEC}_{\mathcal{X}}$ over states

$$\langle \langle \mathcal{I}, \mathcal{M} \rangle, \alpha\sigma, \langle \mathcal{I}', \mathcal{M}' \rangle \rangle \in \text{D-EXEC}_{\mathcal{X}}$$

if and only if

there exists $\theta \in \text{EVALS}_{\Delta}(\text{DEL}(\mathcal{I}, \alpha, \sigma) \cup \text{ADD}(\mathcal{I}, \alpha, \sigma))$ such that:

- $\mathcal{I}' = \text{DO}(\mathcal{I}, \alpha, \sigma, \theta)$ is legal w.r.t. $\mathcal{I}$, α, σ, θ,
- θ and $\mathcal{M}$ agree on the common values in their domain, and
- $\mathcal{M}' = \mathcal{M} \cup \theta$.

Concrete transition system under deterministic services

Making use of the transition relation $\text{D-EXEC}_{\mathcal{X}}$, we can now define the RTS generated by a DCDS under deterministic services semantics.

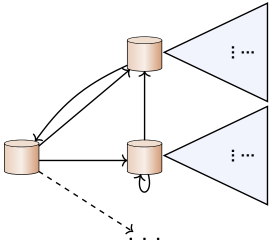
Concrete RTS for a DCDS $\mathcal{X} = \langle \Delta, \mathcal{D}, \mathcal{P}, \mathcal{I}_0 \rangle$ under deterministic services

Is an RTS $\Upsilon_{\mathcal{X}}^{d} = \langle \mathcal{R}, S, s_0, db, \Rightarrow_d \rangle$, where:

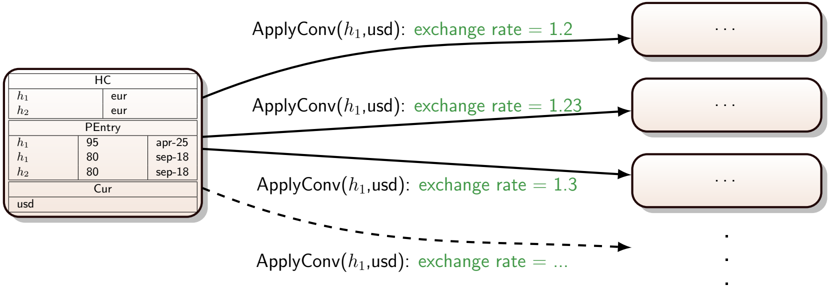
- $s_0 = \langle \mathcal{I}_0, \emptyset \rangle$;
- db is such that $db(\langle \mathcal{I}, \mathcal{M} \rangle) = \mathcal{I}$;
- $\mathcal{R}$ is the database schema of $\mathcal{X}$;
- the states S and the transition relation $\Rightarrow_d$ are defined by simultaneous induction as the smallest set satisfying the following properties:
  1. $s_0 \in S$;
  2. if $\langle \mathcal{I}, \mathcal{M} \rangle \in S$, then for all actions α, parameter substitutions σ, and states $\langle \mathcal{I}', \mathcal{M}' \rangle$ such that $\langle \langle \mathcal{I}, \mathcal{M} \rangle, \alpha\sigma, \langle \mathcal{I}', \mathcal{M}' \rangle \rangle \in \text{D-EXEC}_{\mathcal{X}}$,
     we have that $\langle \mathcal{I}', \mathcal{M}' \rangle \in S$ and $\langle \mathcal{I}, \mathcal{M} \rangle \Rightarrow_d \langle \mathcal{I}', \mathcal{M}' \rangle$.

Sources of unboundedness/infinity

In general: service calls cause . . .



- infinite branching (due to all possible results of service calls);
- infinite runs (usage of values obtained from unboundedly many service calls);
- unbounded DBs (accumulation of such values).



HC	
h_1	eur
h_2	eur

PEntry		
h_1	95	apr-25
h_1	80	sep-18
h_2	80	sep-18

Cur
usd

ApplyConv(h_1,usd): exchange rate = 1.2
ApplyConv(h_1,usd): exchange rate = 1.23
ApplyConv(h_1,usd): exchange rate = 1.3
ApplyConv(h_1,usd): exchange rate = ...
. . .