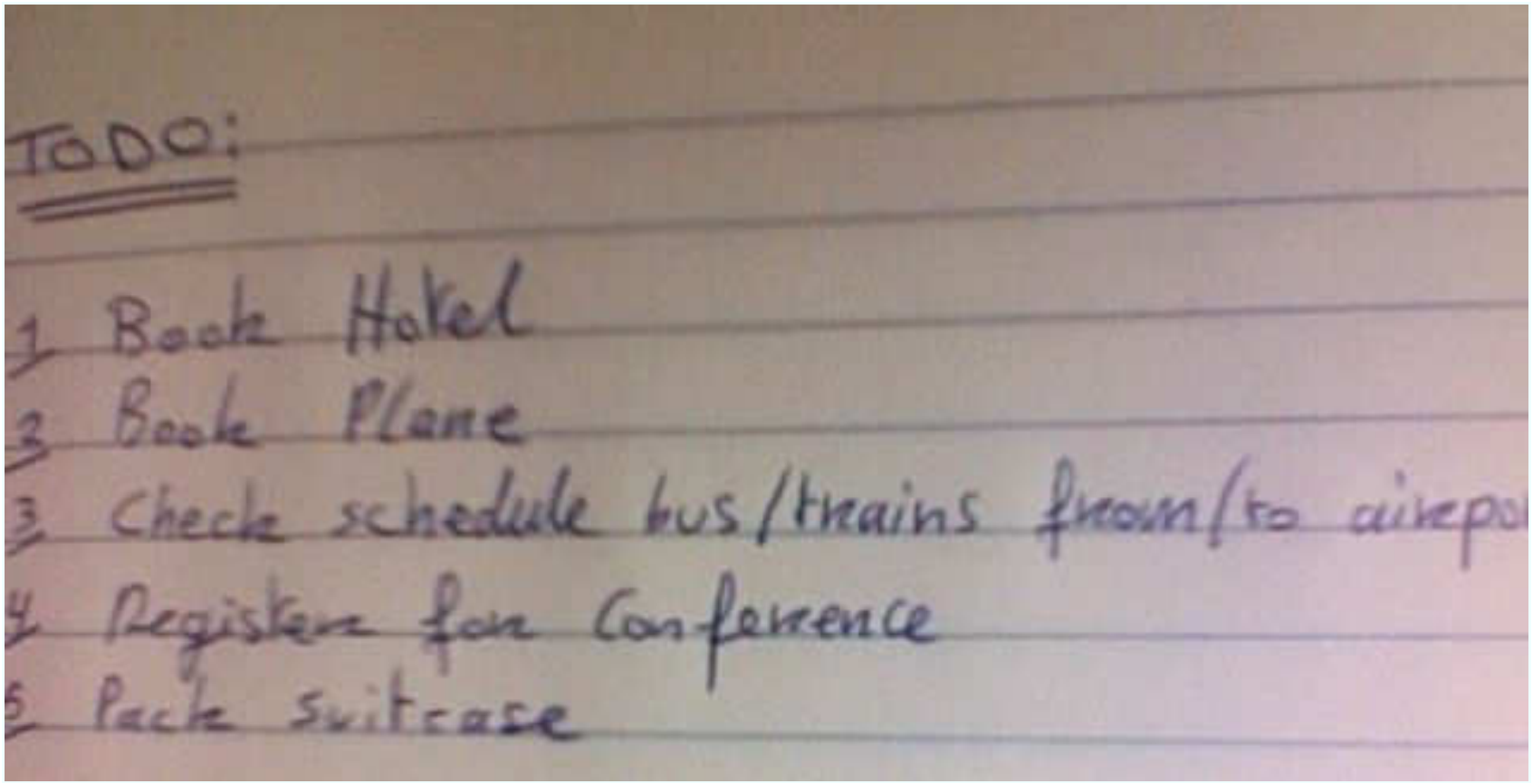
Complexity and Composition of Synthesized Web Services

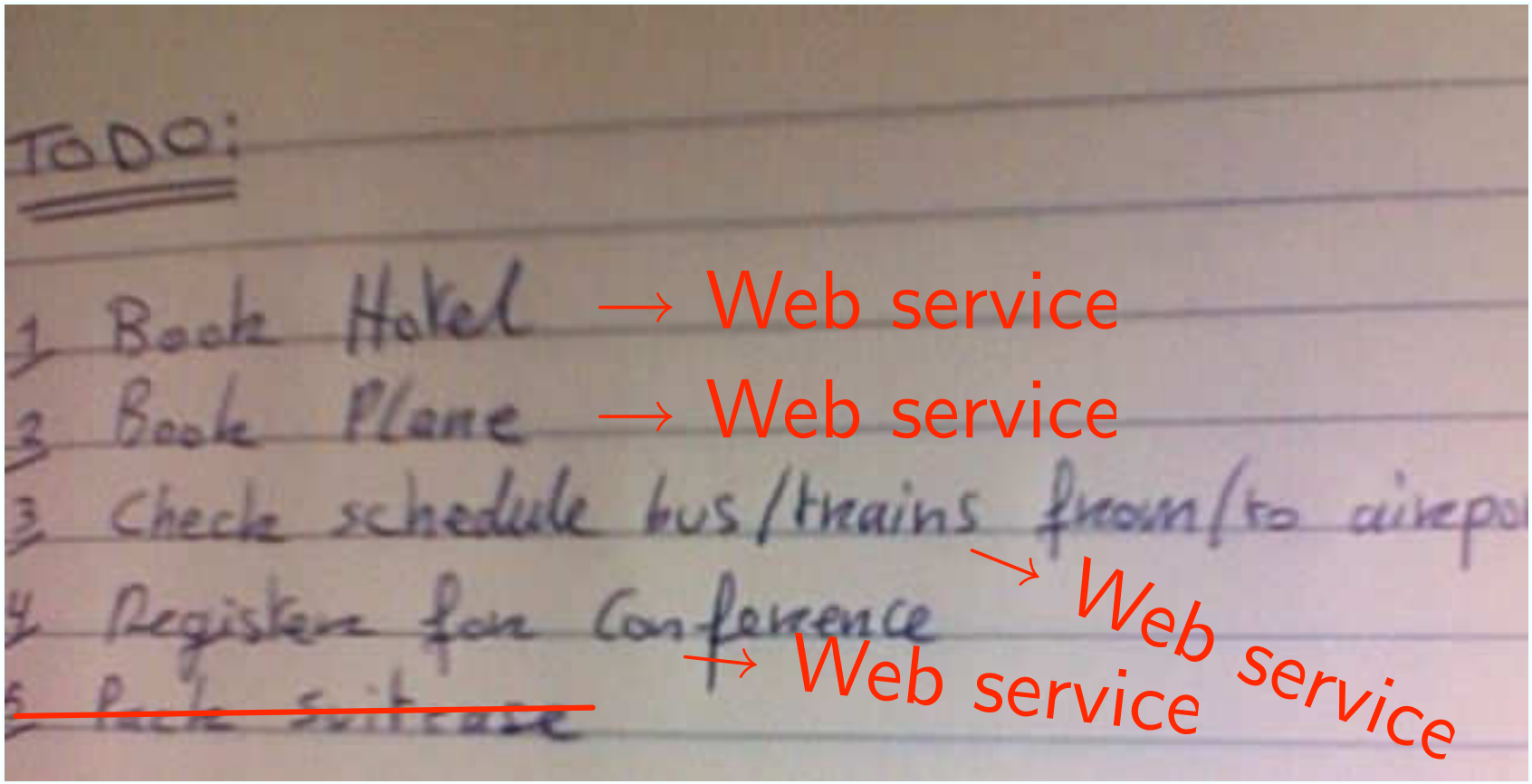
Wenfei Fan[1,2] & Floris Geerts[1] & Wouter Gelade[3] & Frank Neven[3] & Antonella Poggi[4]

[1] University of Edinburgh, UK
[2] Bell Labs, US
[3] Hasselt University & transnational University Limburg, Belgium
[4] SAPIENZA Università di Roma, Italy

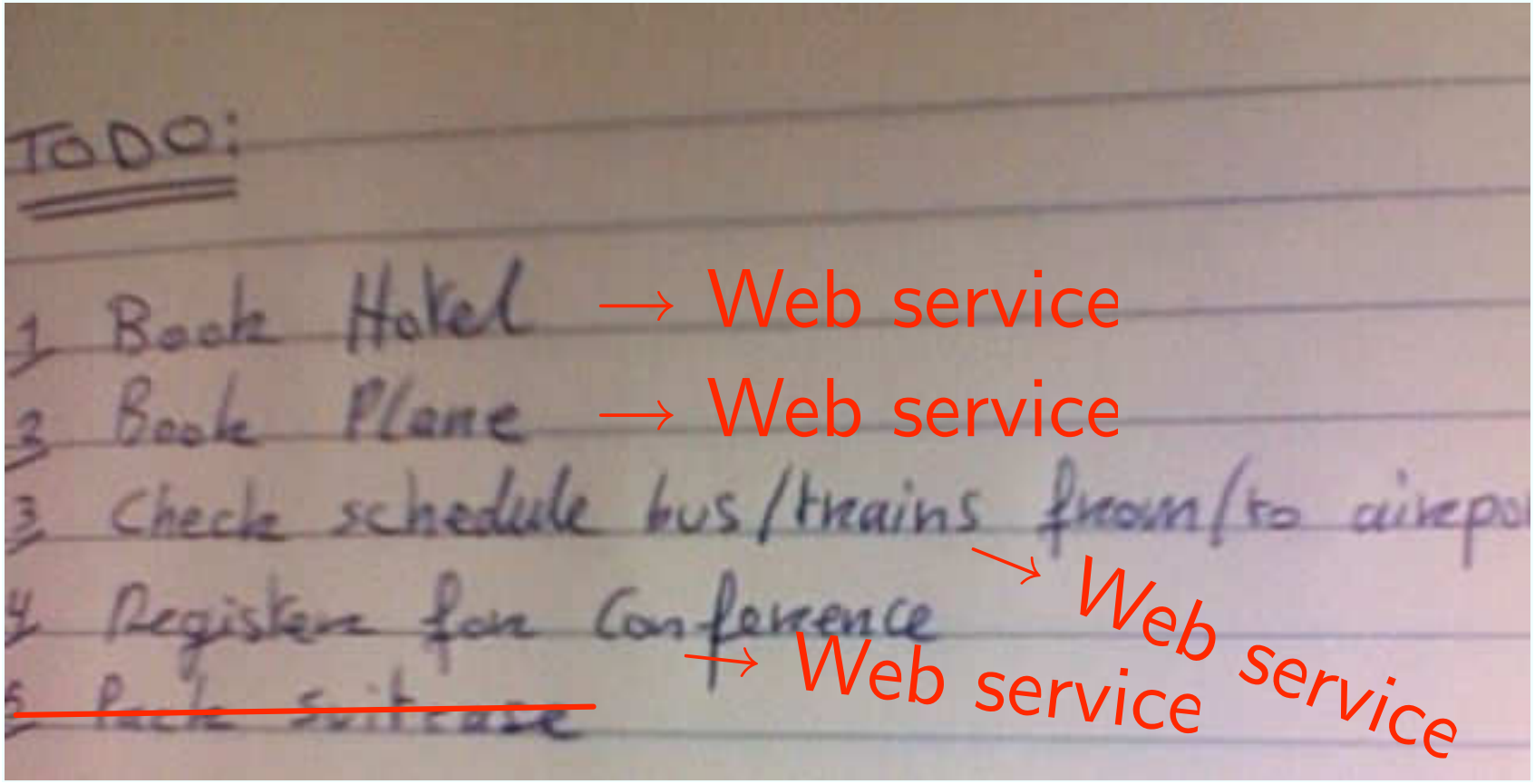
Before coming to Vancouver...



Before coming to Vancouver...

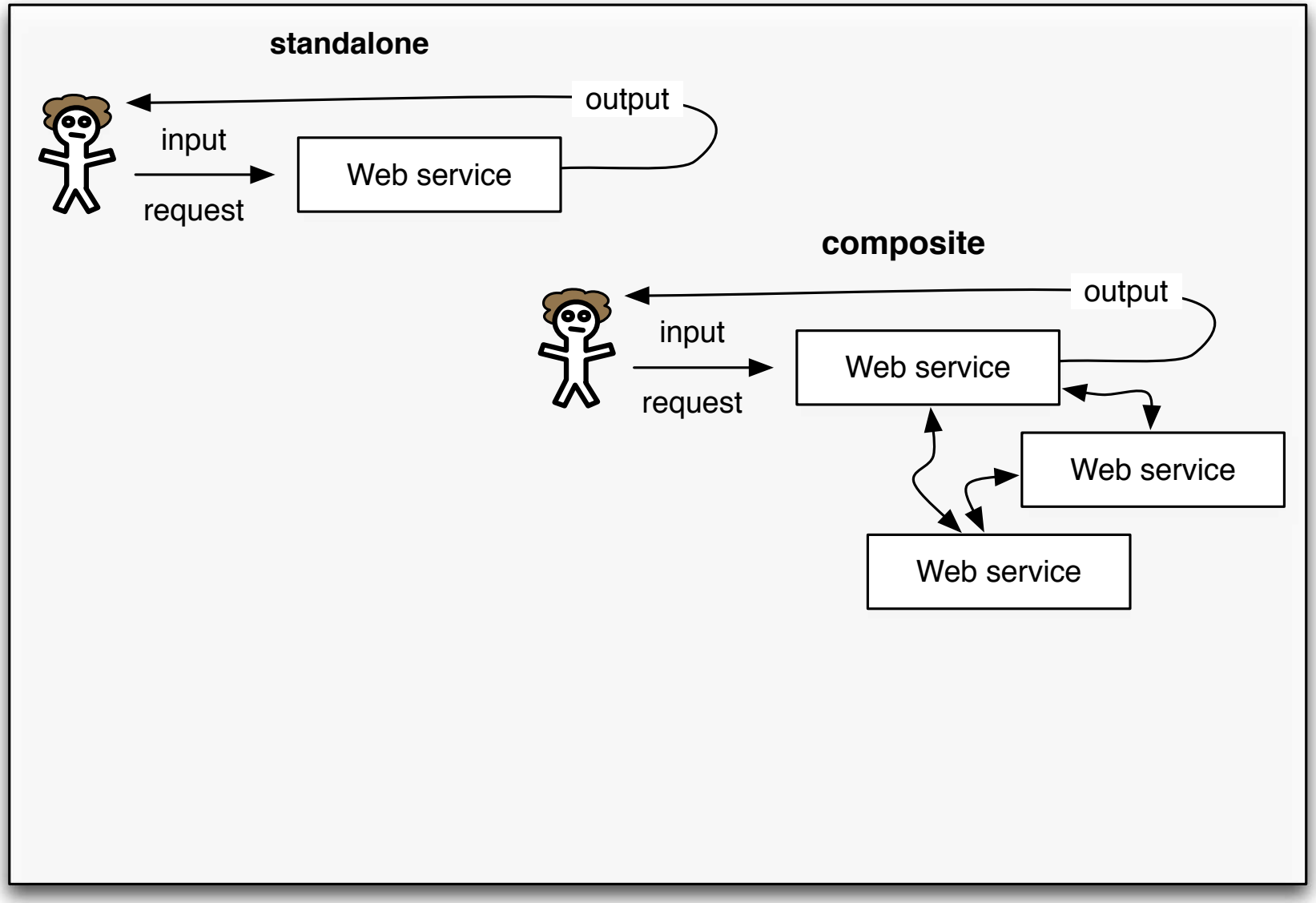


Before coming to Vancouver...

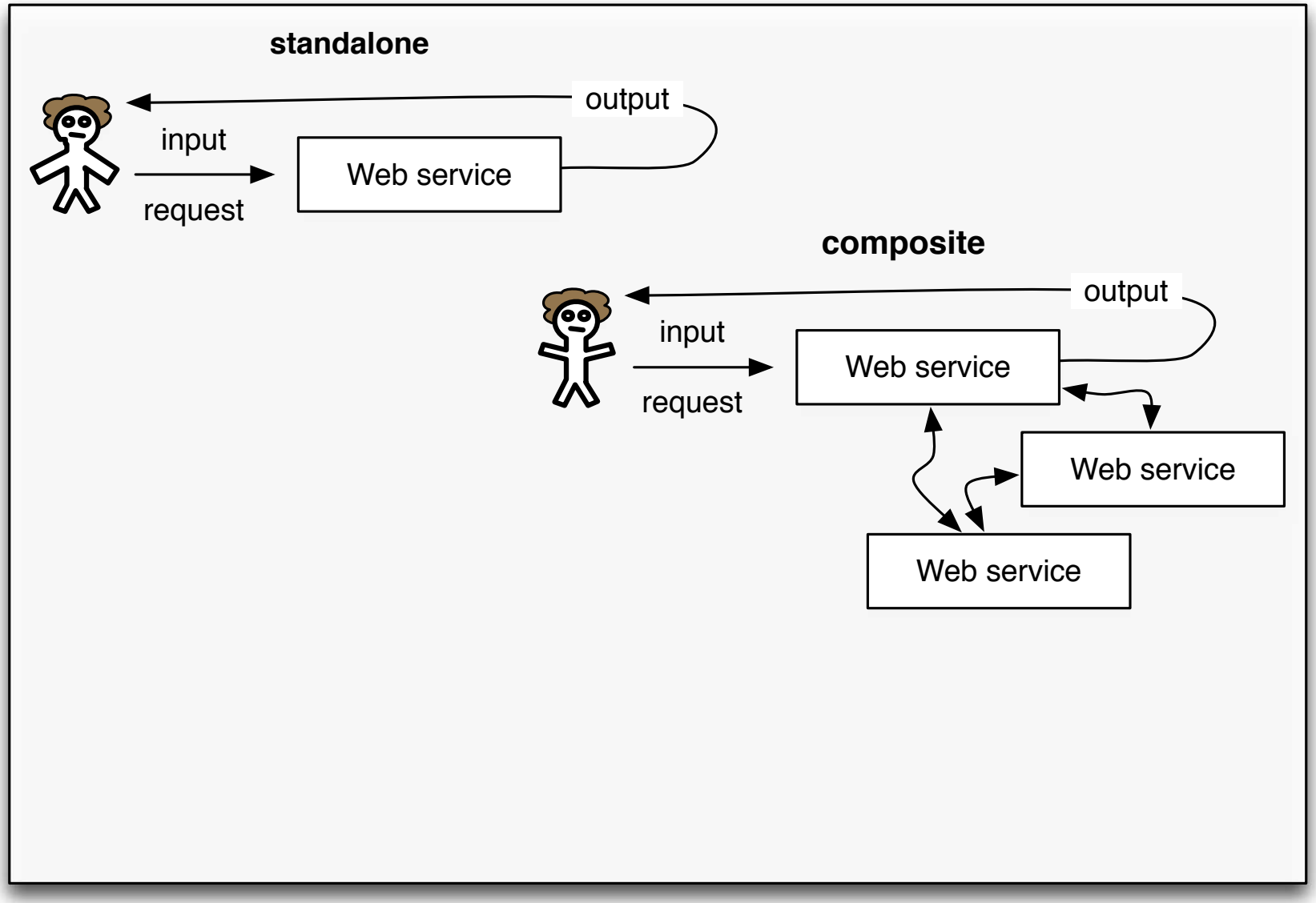


Prevalent use of Web services highlights the need for studying their properties ...

Web services



Data-driven Web services



Models for Web services

- Standards: WDSL, WSCL, OWL-S, SWFL, BEPL, ...
- Automata-based models:
  - Roman model [Berardi, Calvanese, De Giacomo, Lenzerini, Mecella, 2005]
  - Guarded automata [Fu, Bultan, Su, 2004]
  - Colombo model [BCGLM, Hull, 2005]
  - In/Ouput NFA's [Pistore, Traverso, Bertoli, Marconi, 2005]
  - ...
- Transducer-based models:
  - Relational transducers [Abiteboul, Vianu, Fordham, Yesha, 2000]
  - Abstract state machines [Spielmann, 2000]
  - Peer model [Deutsch, Sui, Vianu, Zhou, 2006]
  - ...

Goals of this work

1. Introduce alternation (such as in alternating finite state automata) in Web services in the form action synthesis.

   $\Rightarrow$ Synthesized Web services (SWS)

2. Study classical decision problems for SWS's (non-emptiness, validity, equivalence)

3. Study the composition problem for SWS's (taken the synthesis into account)

4. Finite state automata are a special case of SWS's $\Rightarrow$ revisit composition problem

Modelling data-driven Web services ...

.... is not an easy job!

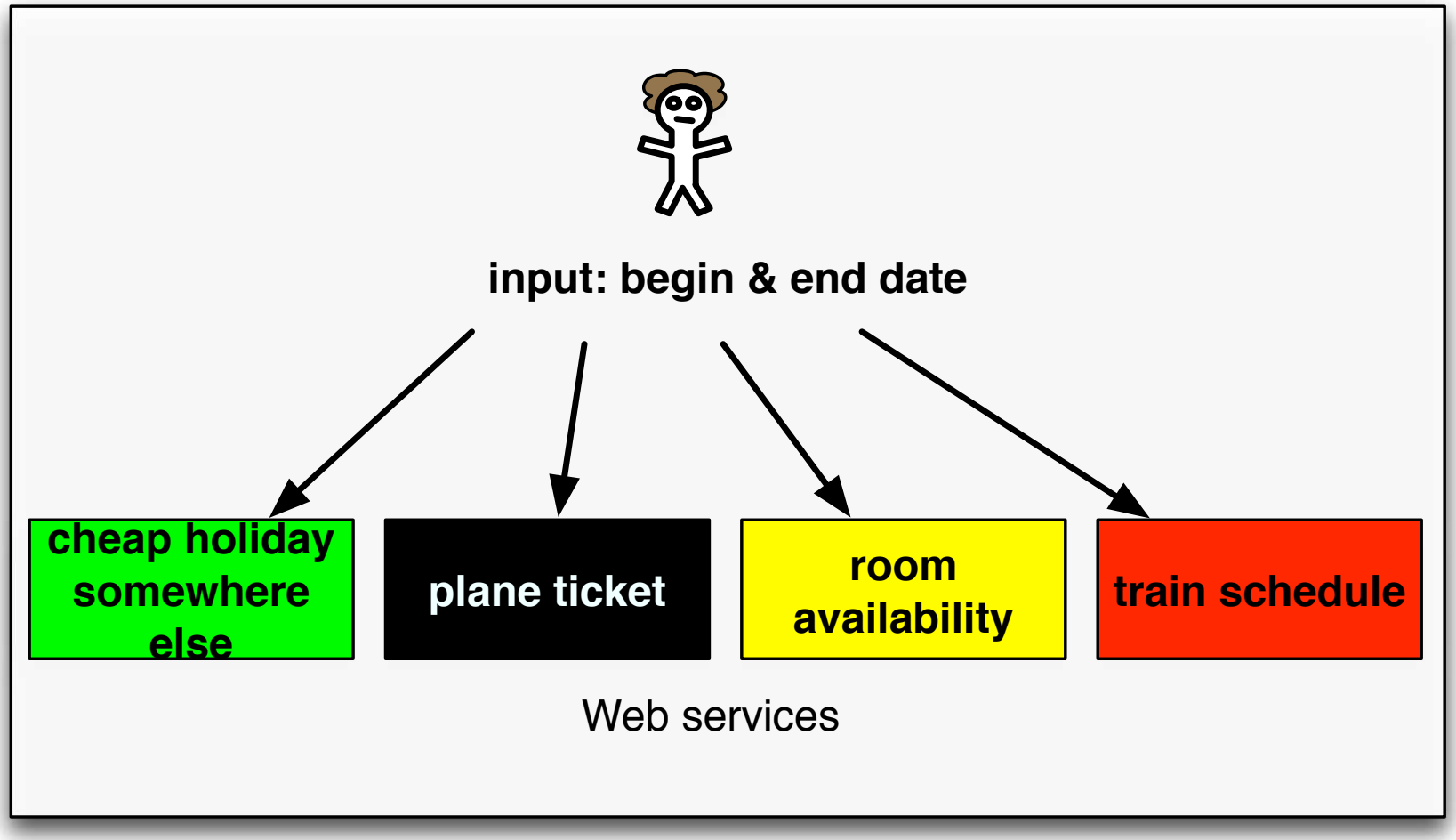
- Many different aspects are involved
  - conversation protocols/message passing;
  - interaction with databases;
  - communication channels (ideal vs. noisy);
  - queueing behavior;
  - ....
- We have to make choices what to model and what not.
- Our model of synthesized Web services focusses on message passing, interaction with data and action synthesis.

The Peer model [Deutsch, Sui, Vianu, Zhou, 2006]

- Individual Web service (peer) is a tuple $\mathcal{W} = (\mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathbf{Q}_{\text{in}}, \mathbf{Q}_{\text{out}}, \mathcal{R})$, where
  - $\mathbf{D}$ is a database schema;
  - $\mathbf{S}$ is a schema for states;
  - $\mathbf{I}$ is a schema for input;
  - $\mathbf{A}$ is a schema for actions;
  - $\mathbf{Q}_{\text{in}}$ and $\mathbf{Q}_{\text{out}}$ are schemas for in-queues and out-queues, respectively; and
  - $\mathcal{R}$ is a set of rules.

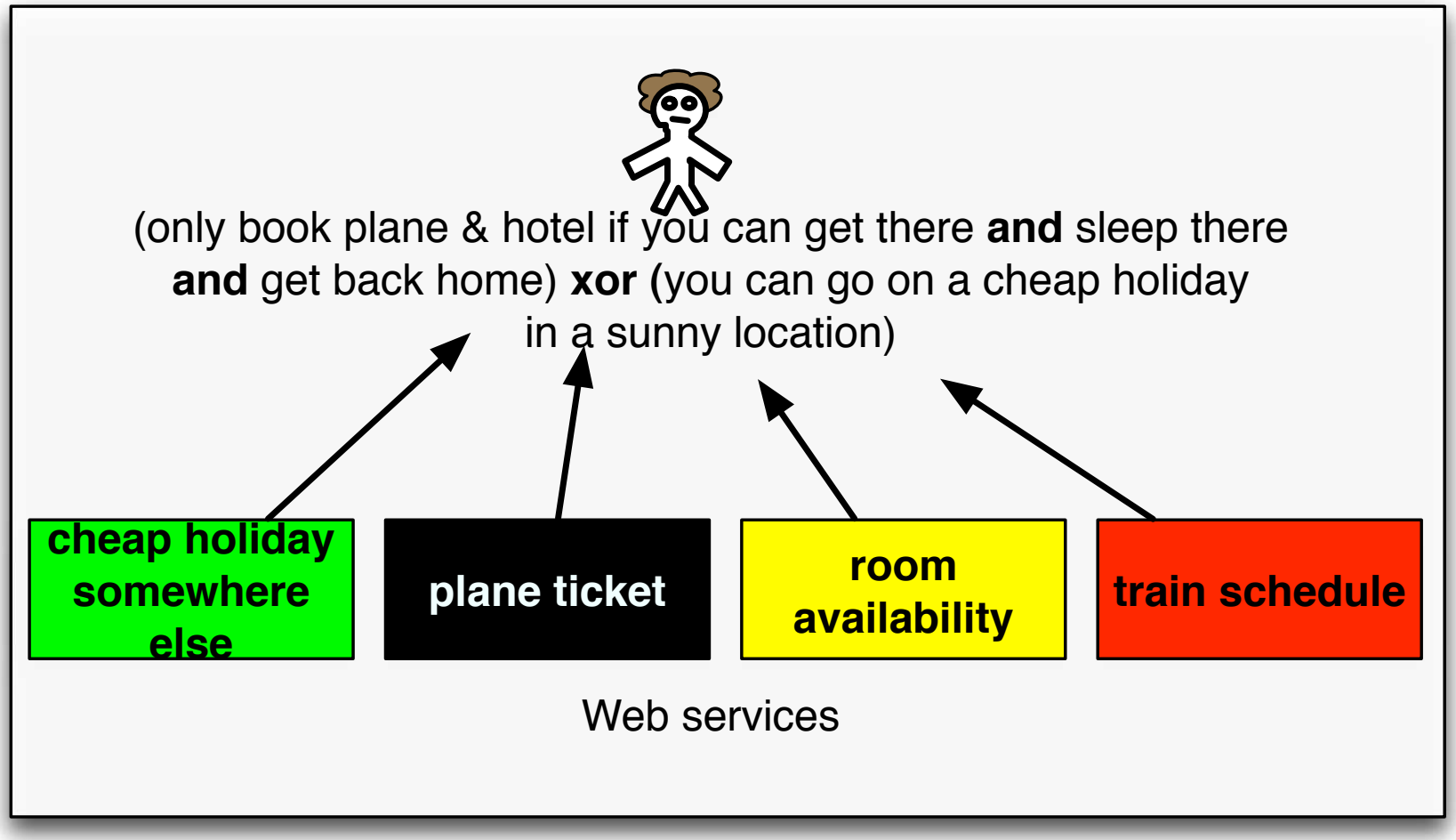
Action synthesis

As far as we know this is a feature not present in the current models ...



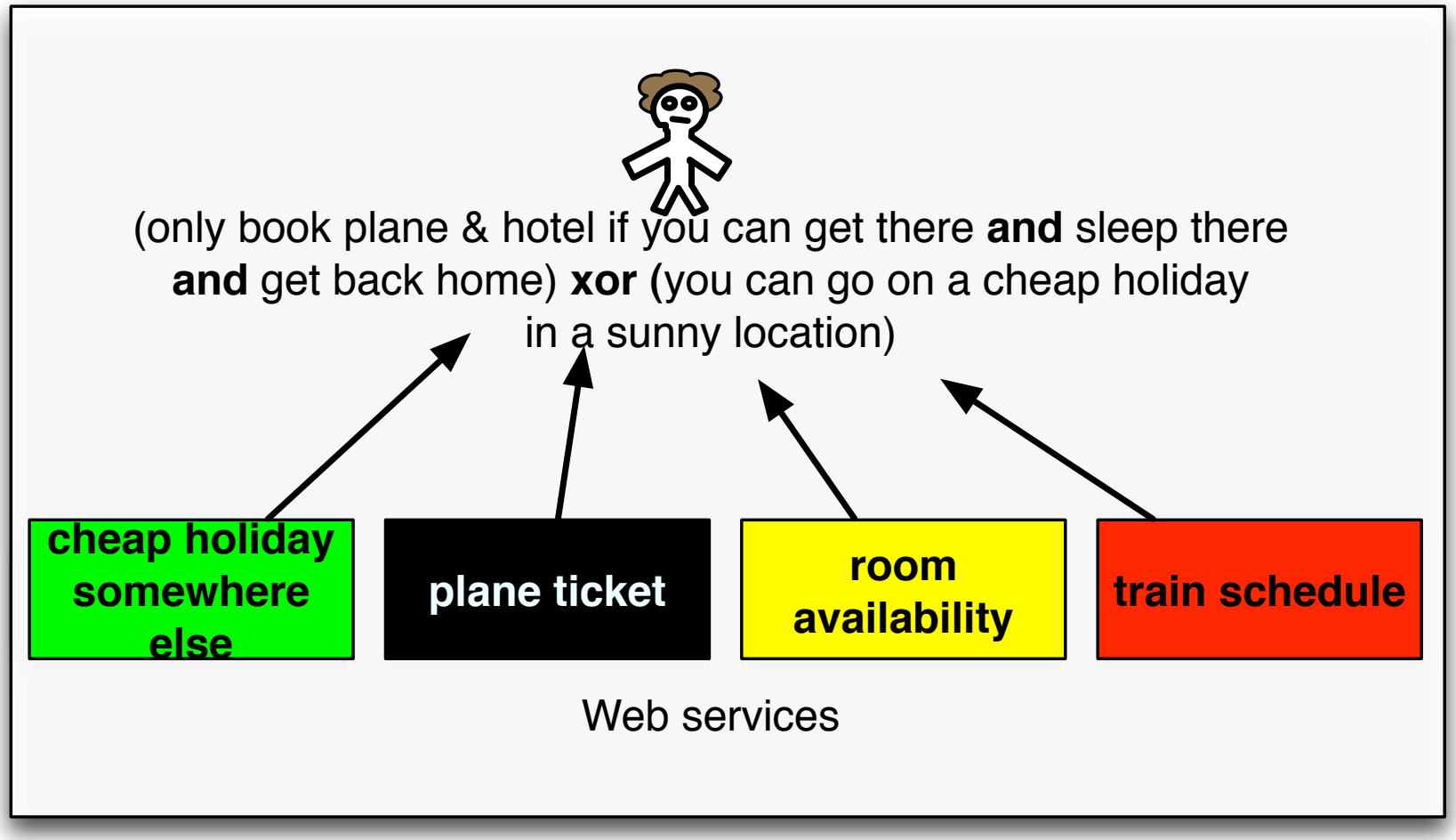
Action synthesis

As far as we know this is a feature not present in the current models ...



Action synthesis

As far as we know this is a feature not present in the current models ... Synthesis is required in order to generate output action.



Before presenting our model...

- We assume a single input schema $\mathcal{R}$ for the local database to which a Web service has access to;
- An instance D of $\mathcal{R}$ is to remain unchanged during the execution of a Web service;
- The user input is modelled by a time-stamped instance $\mathcal{I}$ over a schema R_{in} such that

$$I_j = \{\bar{t} \mid \bar{t} \in \mathcal{I} \wedge t[ts] = j\}$$

  is the j-th input message; So $\mathcal{I}$ corresponds to a single communication session.
- The output a Web service is an instance over schema R_{out}.

Up/Downward Message/Action passing

We assume the presence of two query languages:

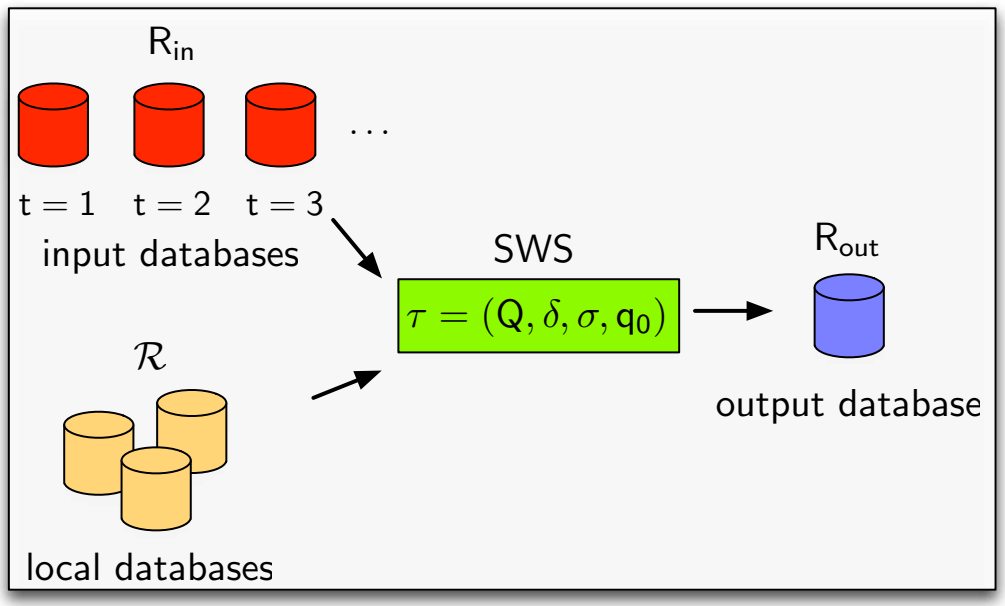
- $\mathcal{L}_{\text{Msg}}$ for downward message passing and interaction with user input $\mathcal{I}$ and local database D.
- We denote the results of these queries (i.e., the downward messages) by relations Msg over the input schema R_{in}.
- $\mathcal{L}_{\text{Act}}$ for upward action passing and the generation of the output.
- We denote the results of these queries (i.e., upward actions) by relations Act over the output schema R_{out}.

Synthesized Web services in SWS($\mathcal{L}_{\text{Msg}}, \mathcal{L}_{\text{Act}}$)

Definition (SWS)

A synthesized Web services (or SWS) τ over $\mathcal{R}$, R_{in}, and R_{out} is a 4-tuple $\tau = (Q, \delta, \sigma, q_0)$ where

- Q is a finite set of states; q_0 being the start state;
- δ consists of the set of transition rules; and
- σ consists of the set of synthesis rules.



Synthesized Web services in SWS($\mathcal{L}_{\text{Msg}}, \mathcal{L}_{\text{Act}}$)

Definition (SWS (cnt'd))

Associated with each state $q \in Q$ there are two special relations:

- an internal message register $\text{Msg}(q)$; and
- an action register $\text{Act}(q)$.

For each $q \in Q$, we have a unique transition rule $\delta(q)$ and synthesis rule $\sigma(q)$:

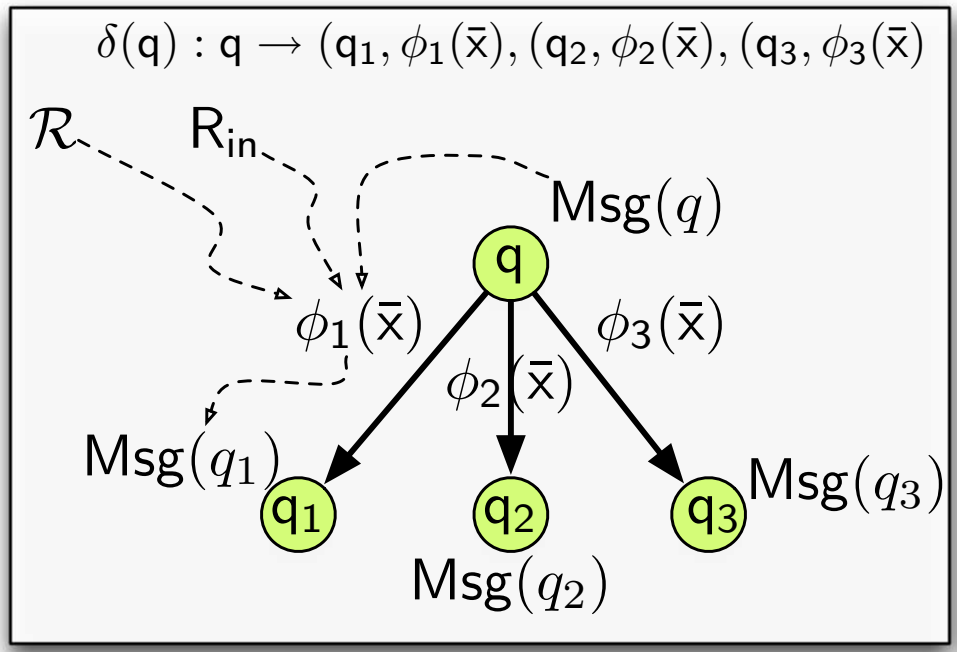
$$\begin{aligned} \delta(q): &\quad q \rightarrow (q_1, \phi_1(\bar{x}_1)), \ldots, (q_k, \phi_k(\bar{x}_k)). \\ \sigma(q): &\quad \text{Act}(q) \leftarrow \psi(\bar{y}), \end{aligned}$$

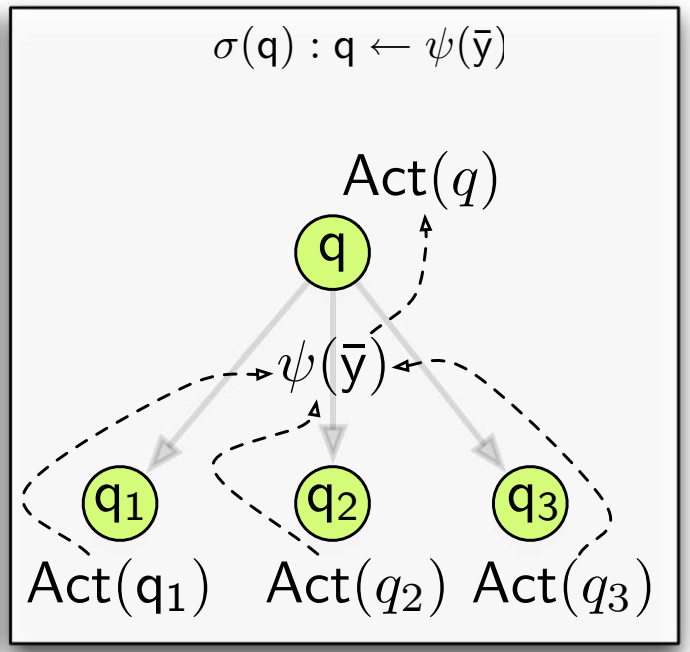
where the ϕ_i's are queries in $\mathcal{L}_{\text{Msg}}$ from $\mathcal{R}, R_{in}, R_{out}, \text{Msg}(q)$ to $Mes(q_i)$ and ψ is a query in $\mathcal{L}_{\text{Act}}$ from $\text{Act}(q_1), \ldots, \text{Act}(q_k)$ to $\text{Act}(q)$ if $k > 0$ and from $\mathcal{R}, R_{in}, \text{Msg}(q)$ to $\text{Act}(q)$ if $k = 0$.

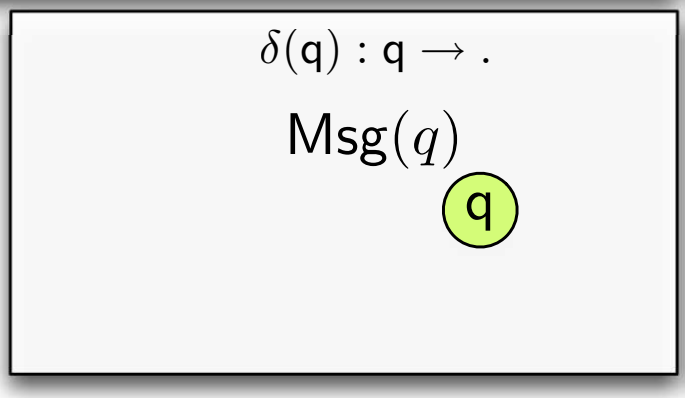
Synthesised Web services: δ- and σ-rules

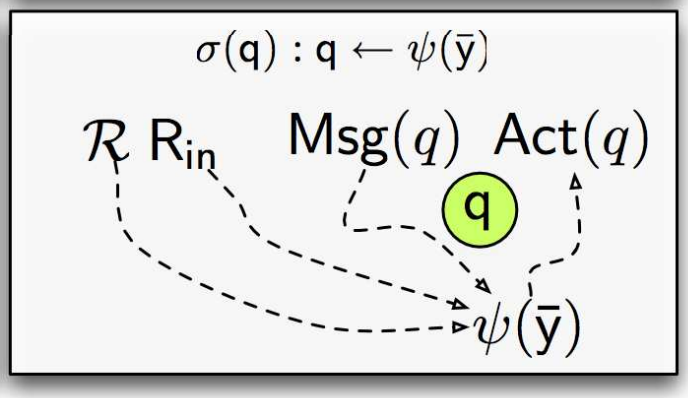
transition rules

synthesis rules







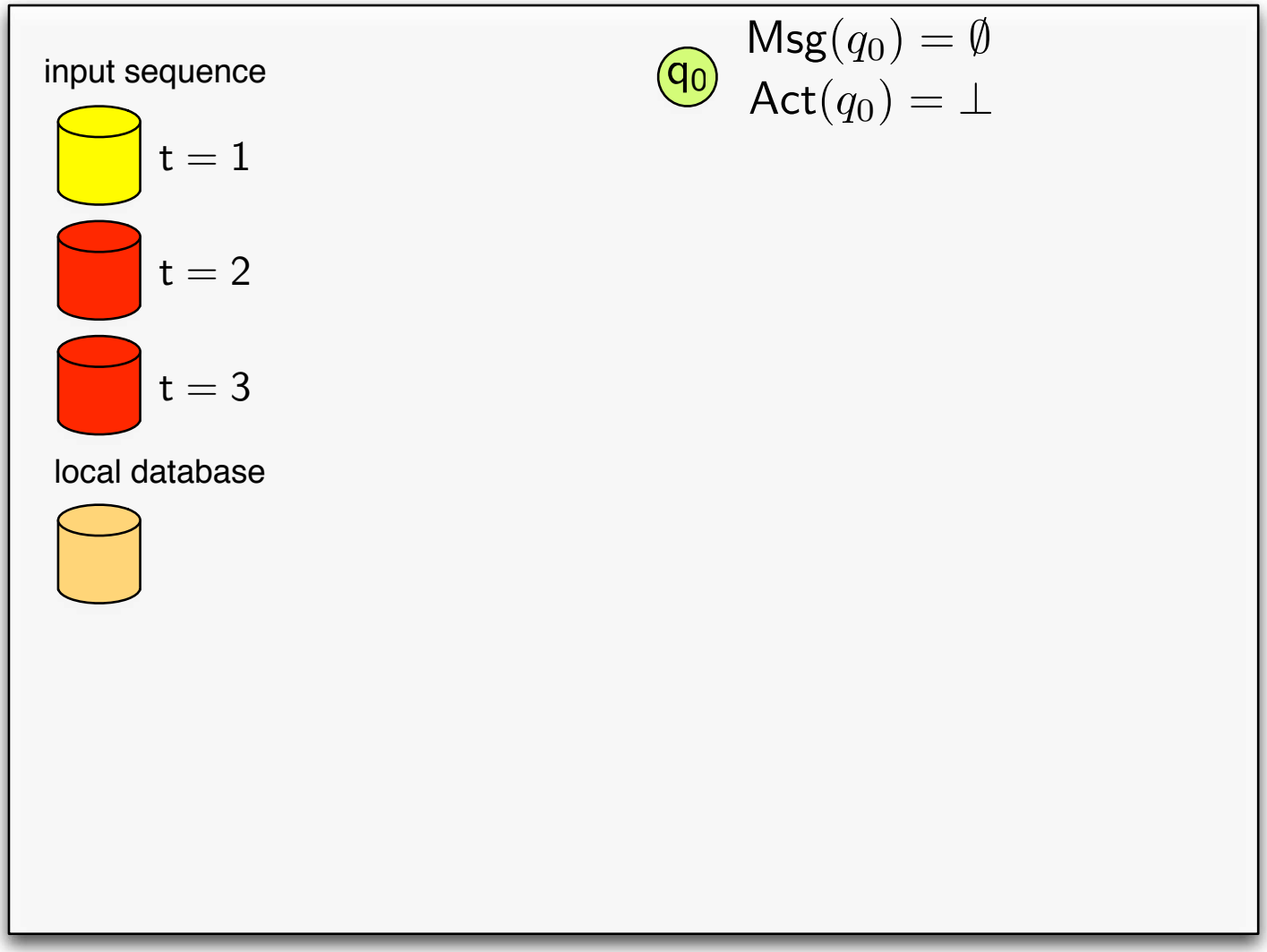


Key properties of a run of an SWS

1. The action registers Act are initially undefined.
2. In the downward phase, one input message is consumed at each step.
3. The generating phase stops in a state q if it has (i) an empty transition rule; or (ii) empty message $\mathrm{Msg}(q)$; or (iii) the input sequence is completely consumed.
4. The synthesis phase defines Act in a bottom-up way. This process starts in a state from the moment all the children's action registers are defined.
5. $\mathrm{Act}(q_0)$ is the result of the execution of τ on D and $\mathcal{I}$, denoted by $\tau(D, \mathcal{I})$.

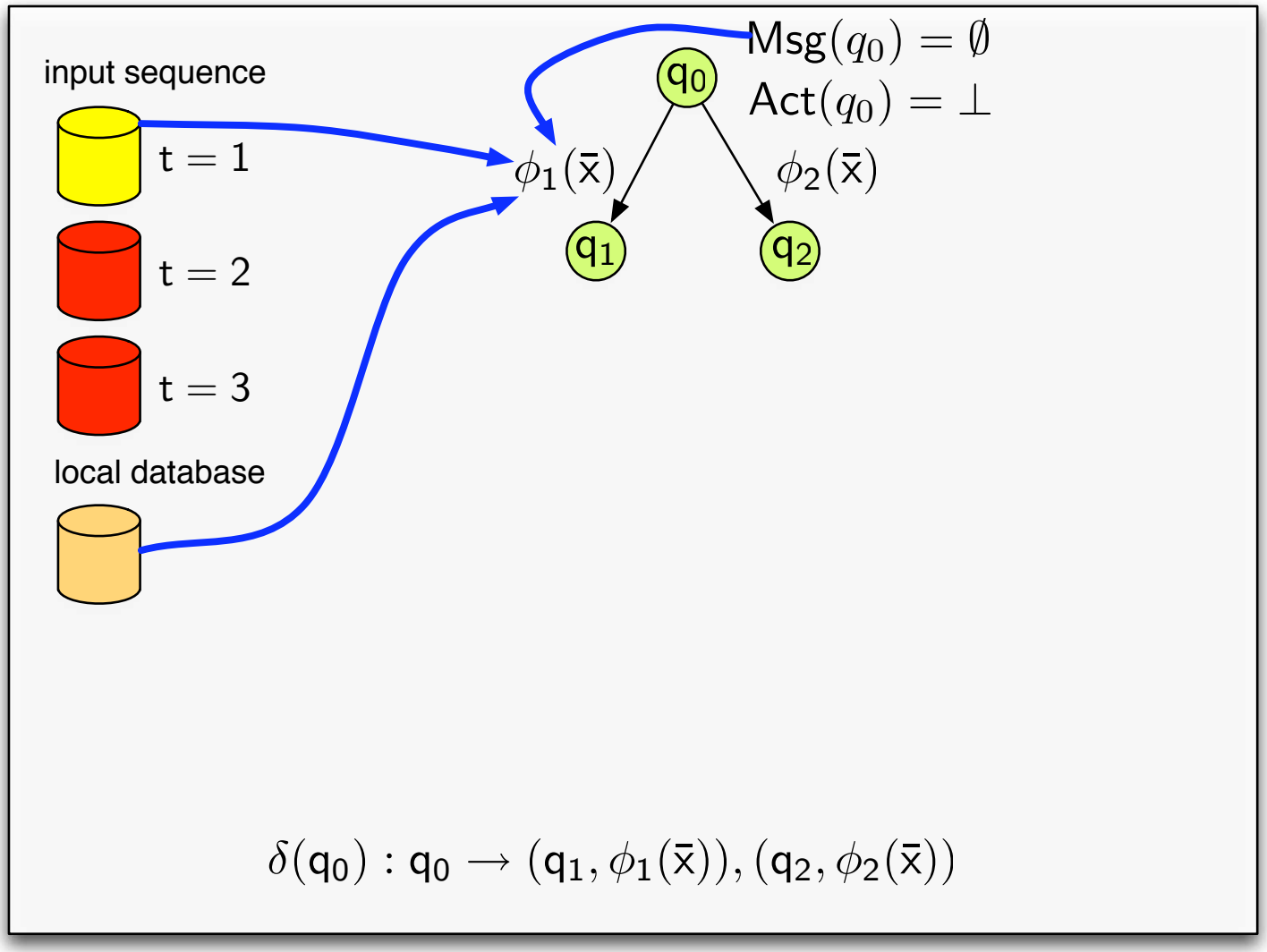
Run of a synthesized Web Service

1. Downward phase (downward using δ)



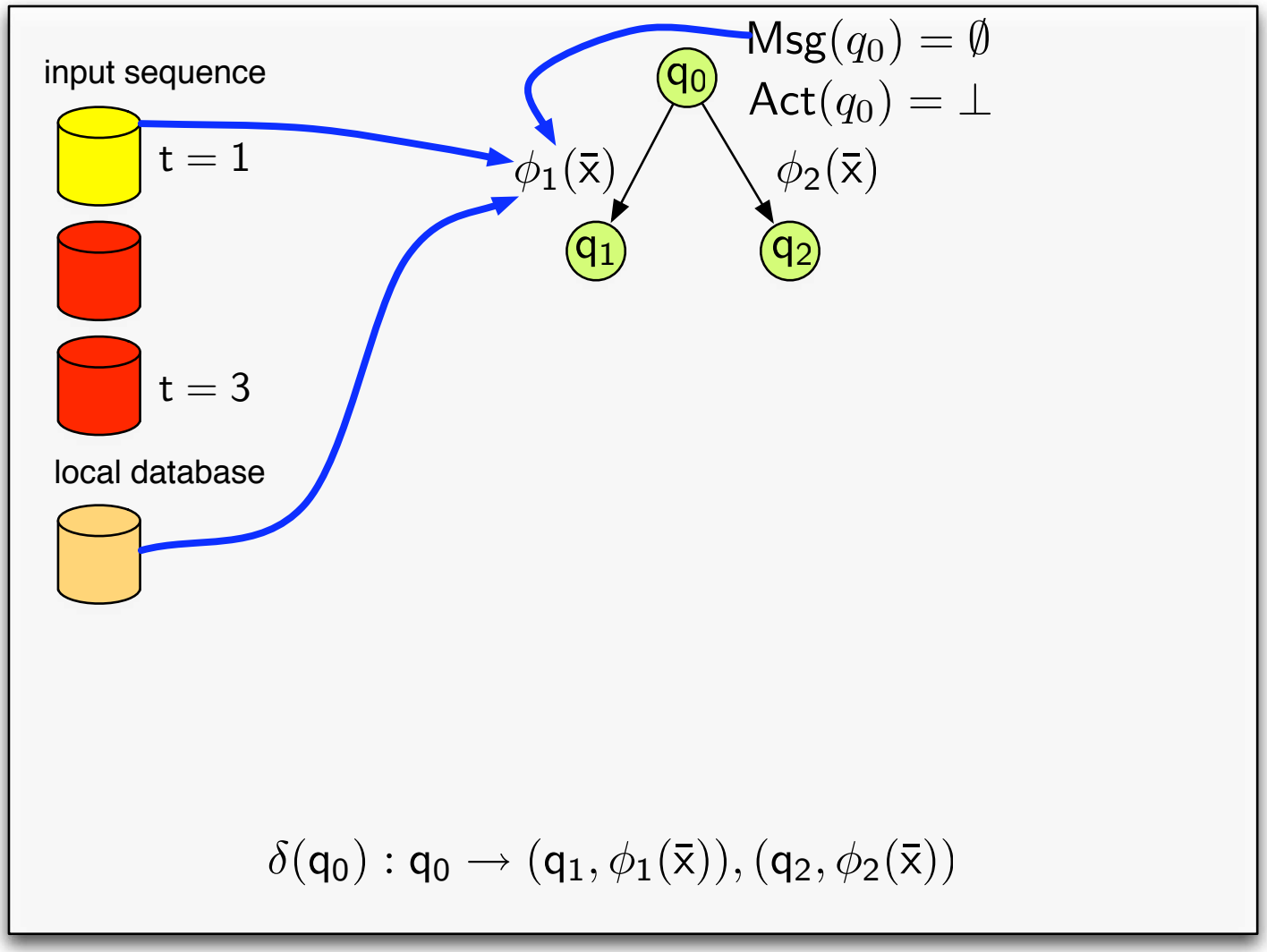
Run of a synthesized Web Service

1. Downward phase (downward using δ)



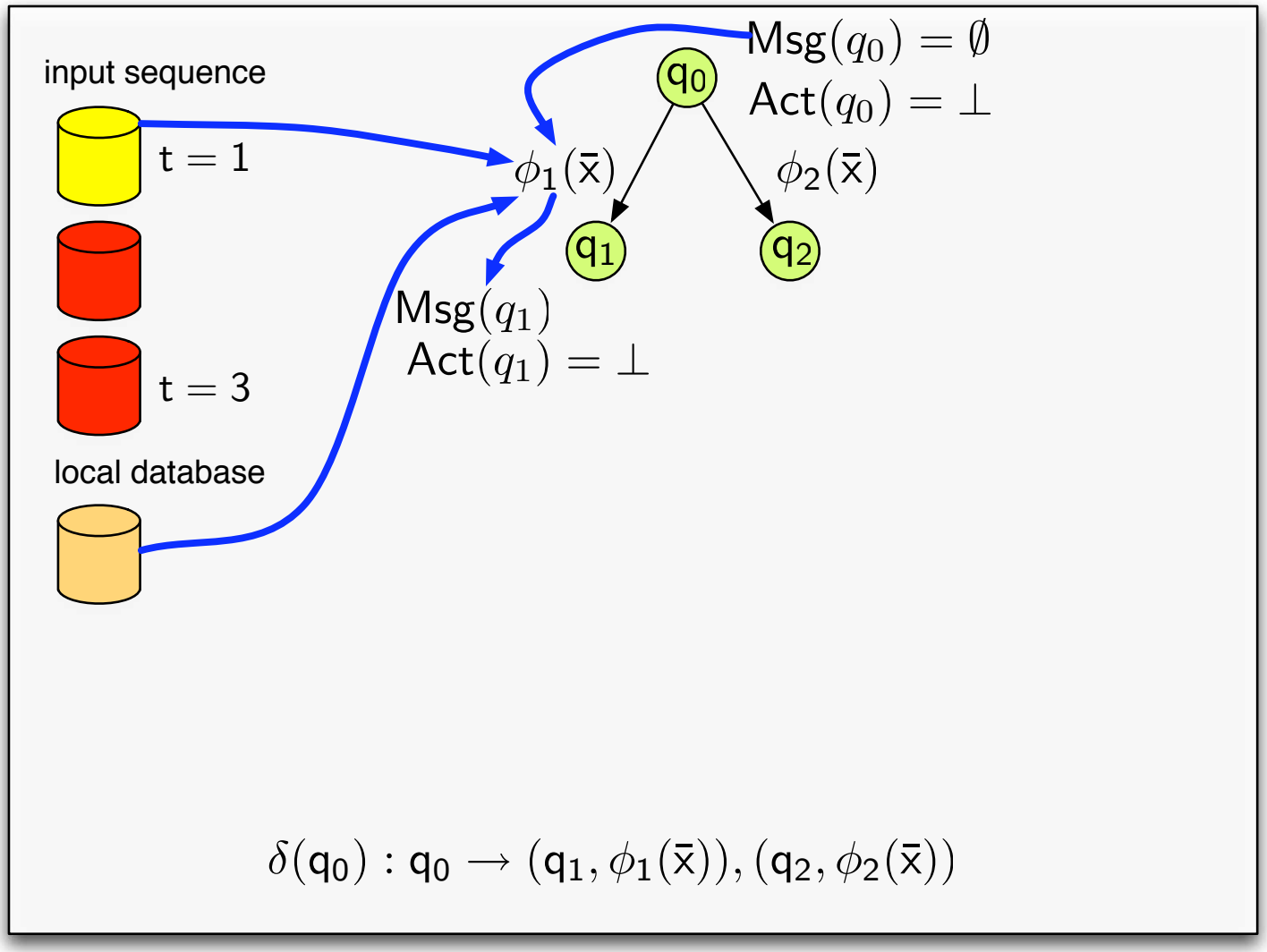
Run of a synthesized Web Service

1. Downward phase (downward using δ)



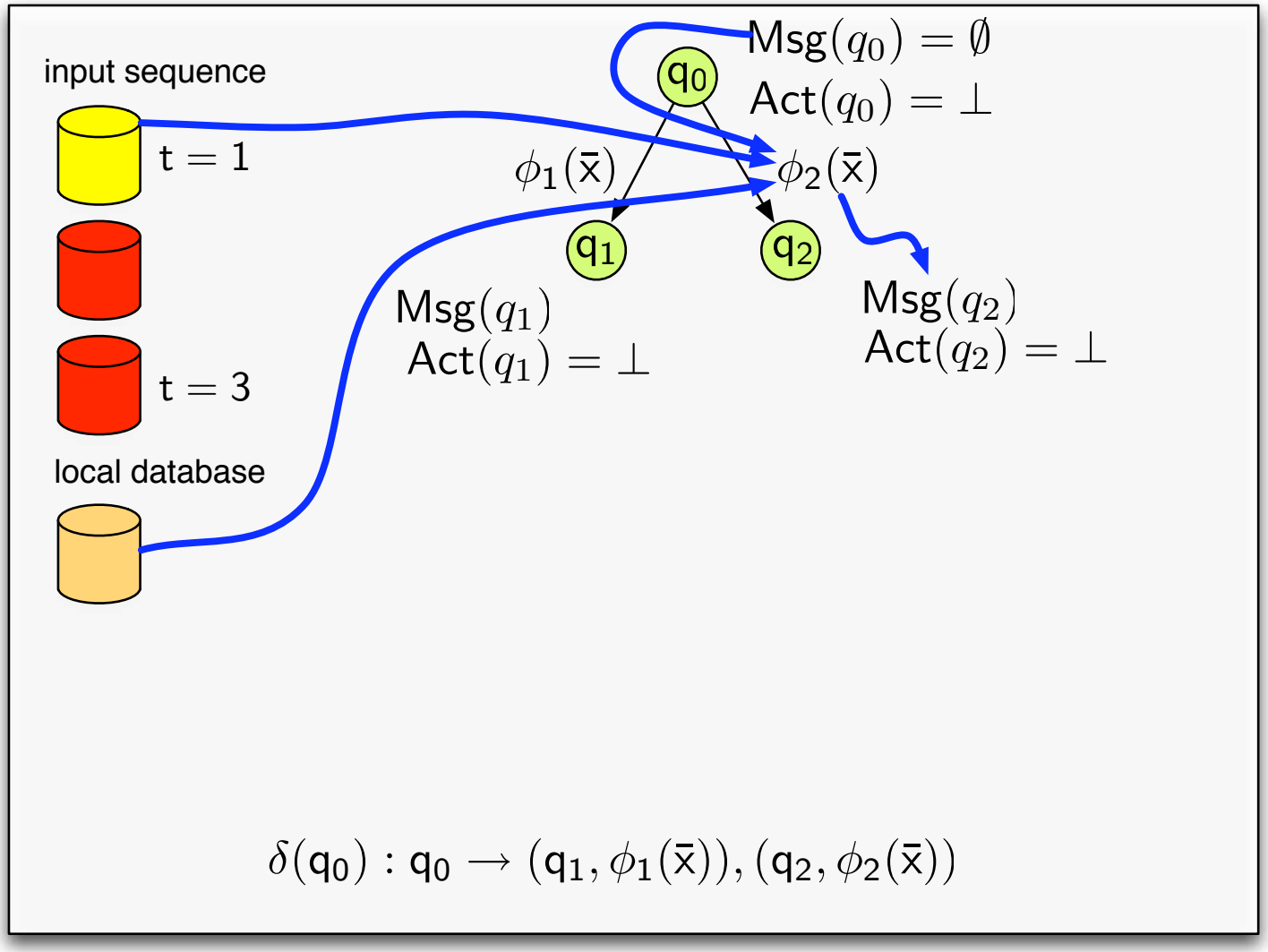
Run of a synthesized Web Service

1. Downward phase (downward using δ)



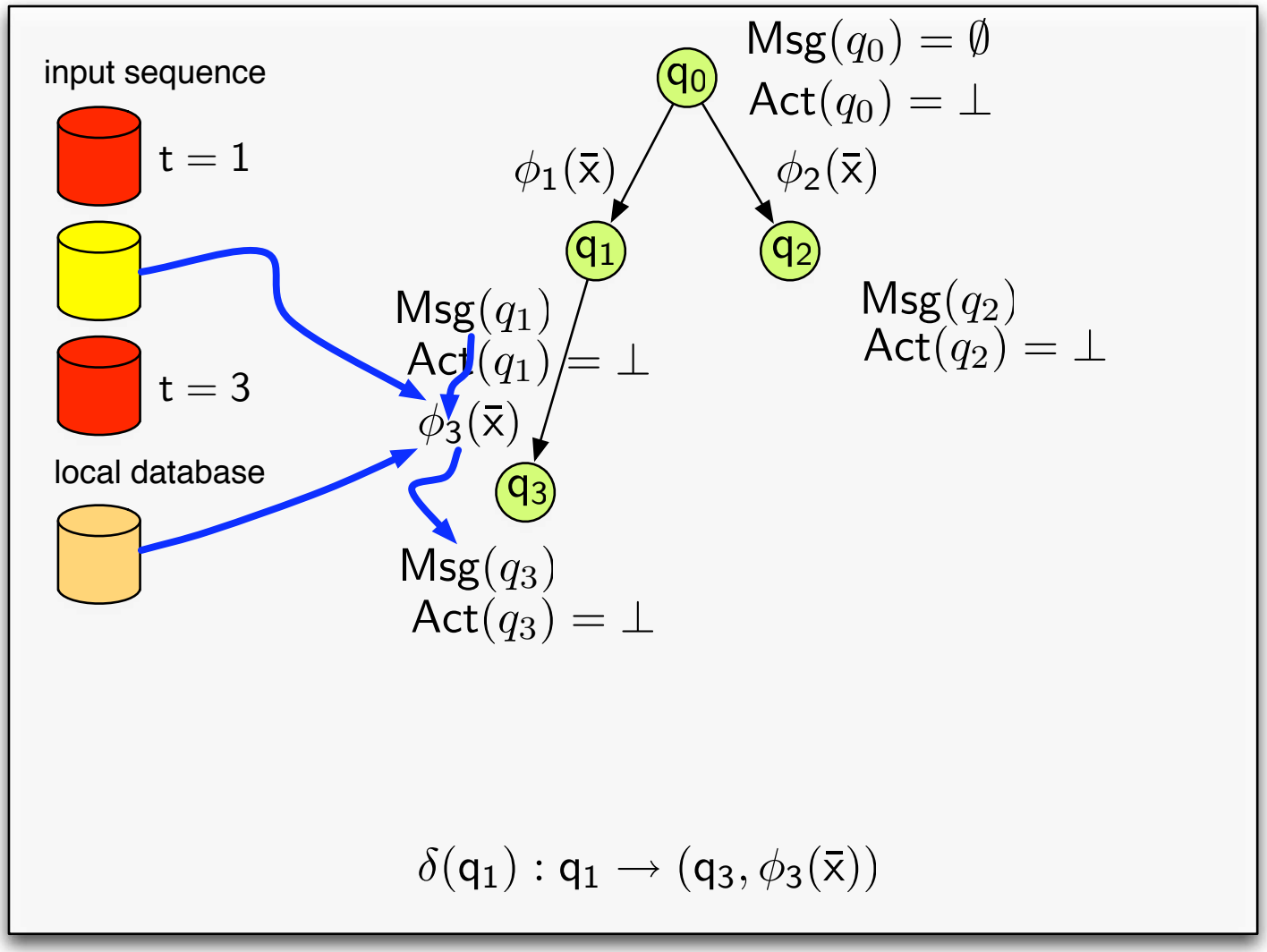
Run of a synthesized Web Service

1. Downward phase (downward using δ)



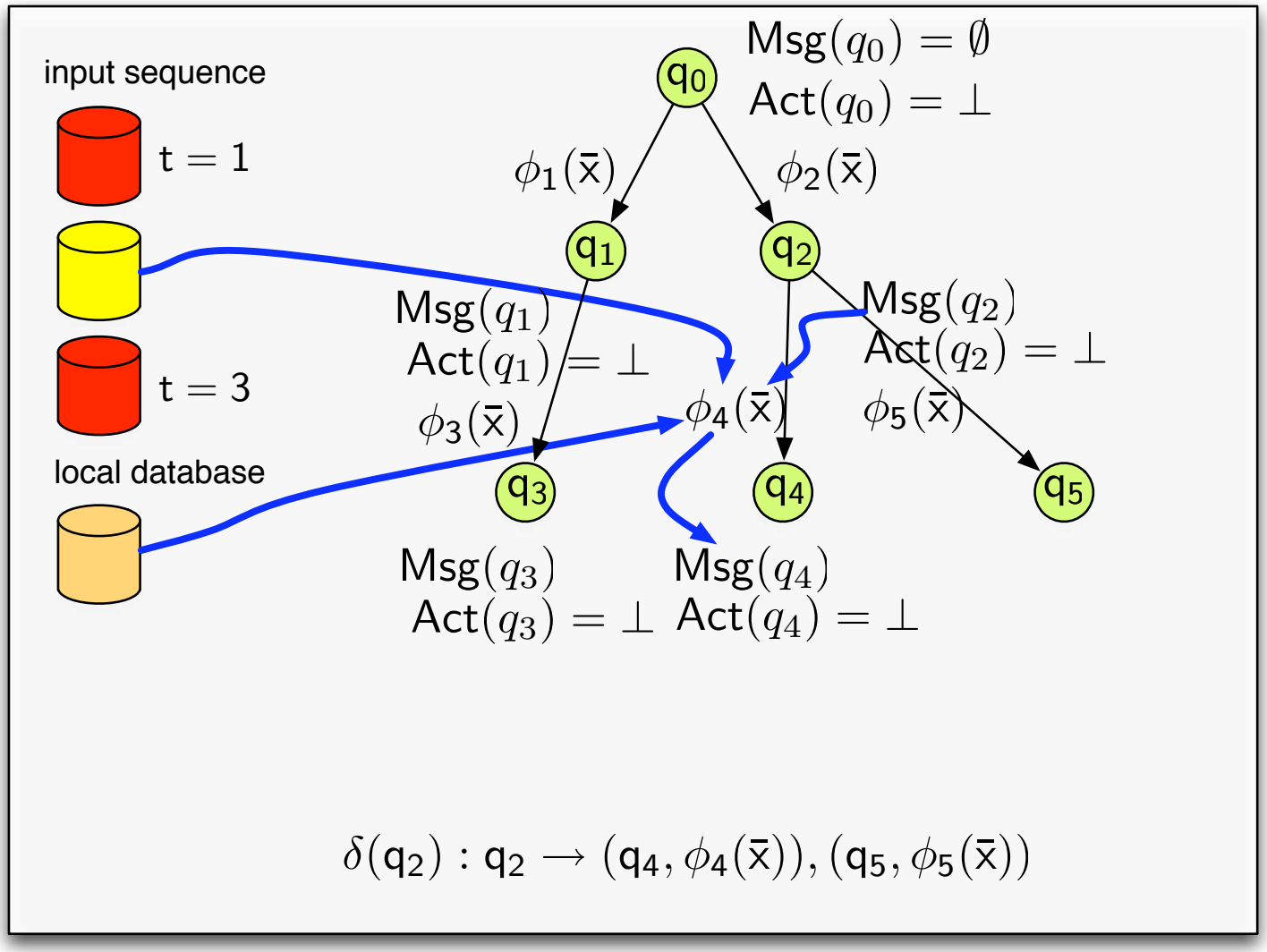
Run of a synthesized Web Service

1. Downward phase (downward using δ)



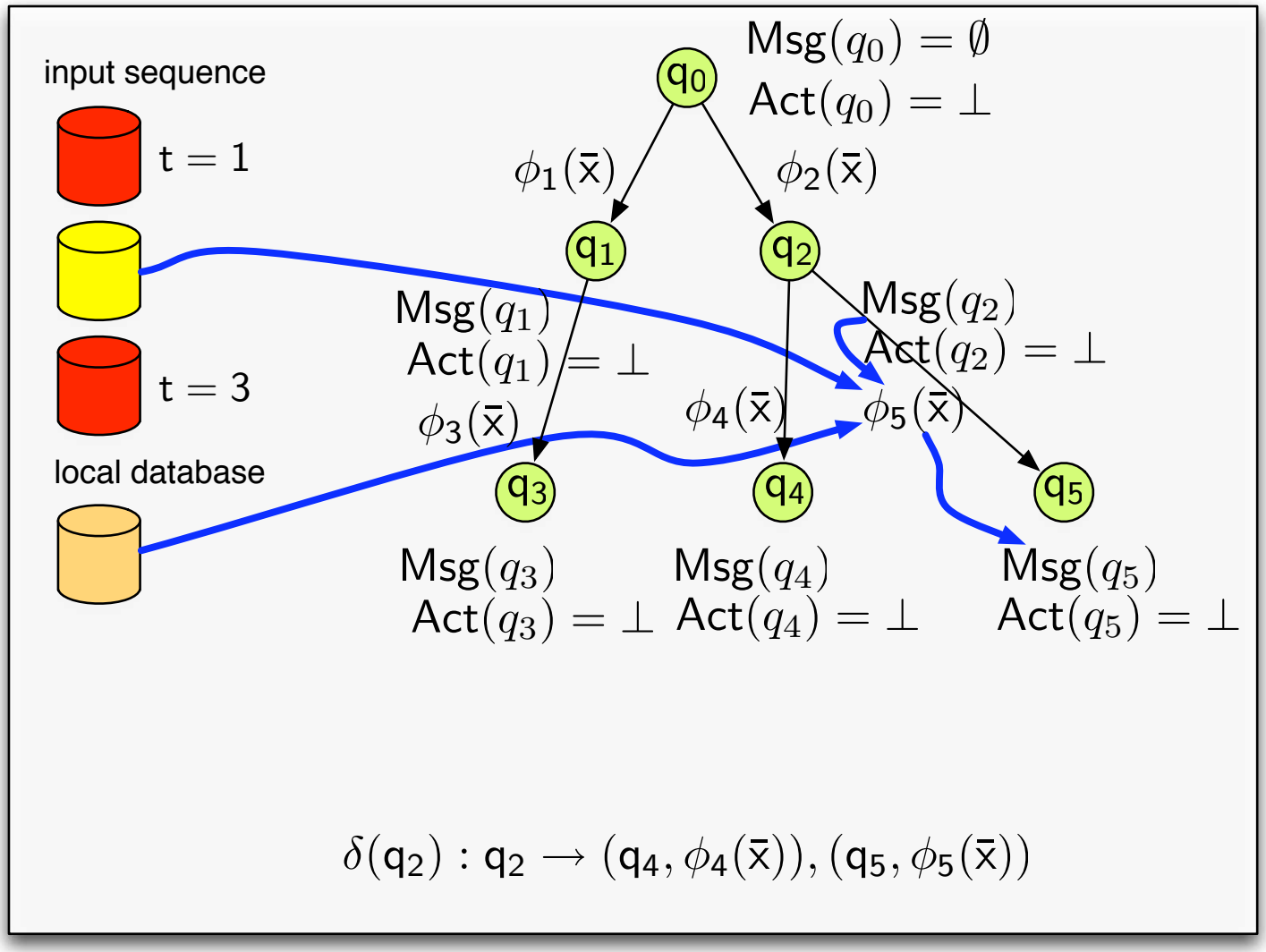
Run of a synthesized Web Service

1. Downward phase (downward using δ)



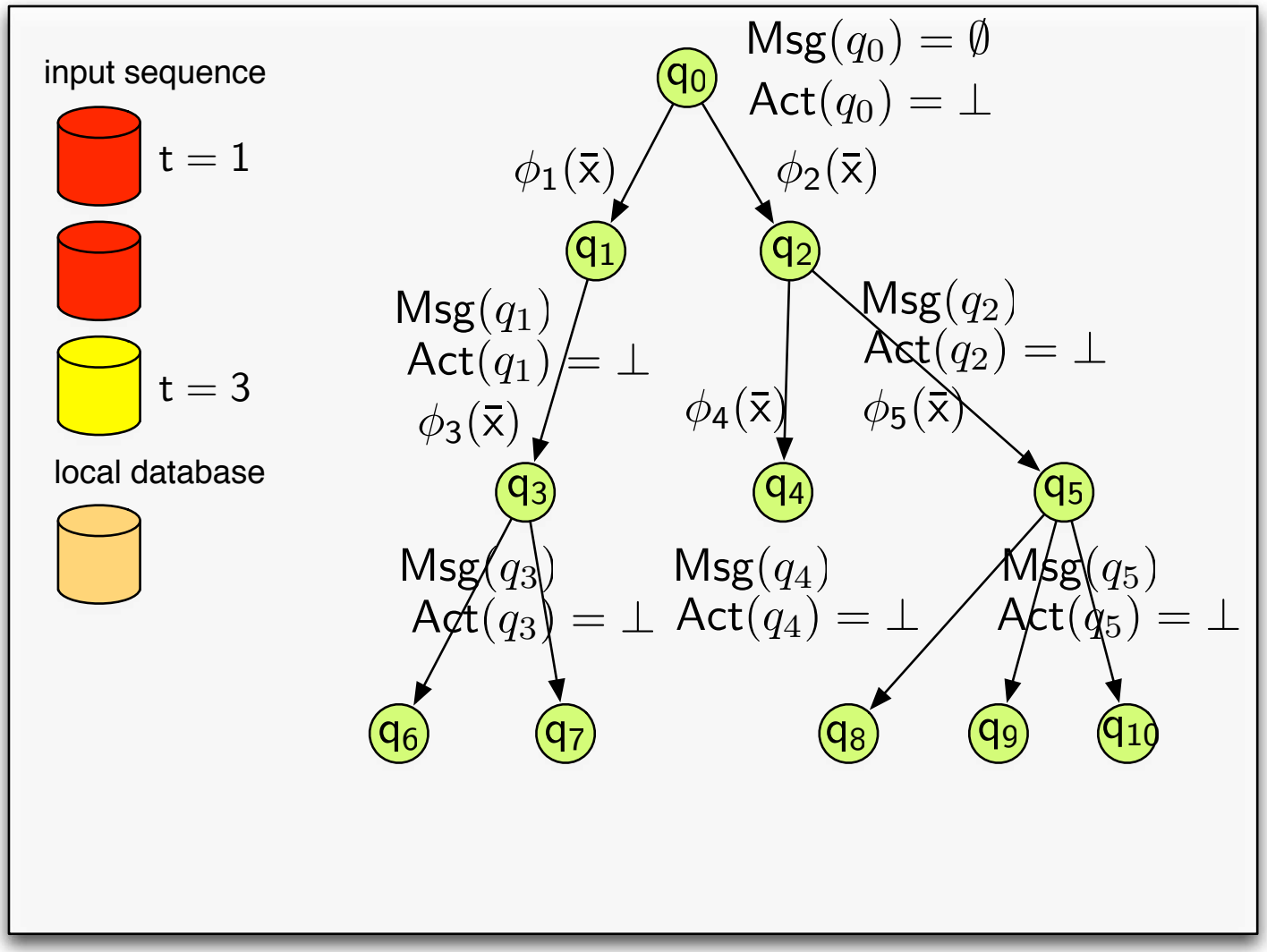
Run of a synthesized Web Service

1. Downward phase (downward using δ)



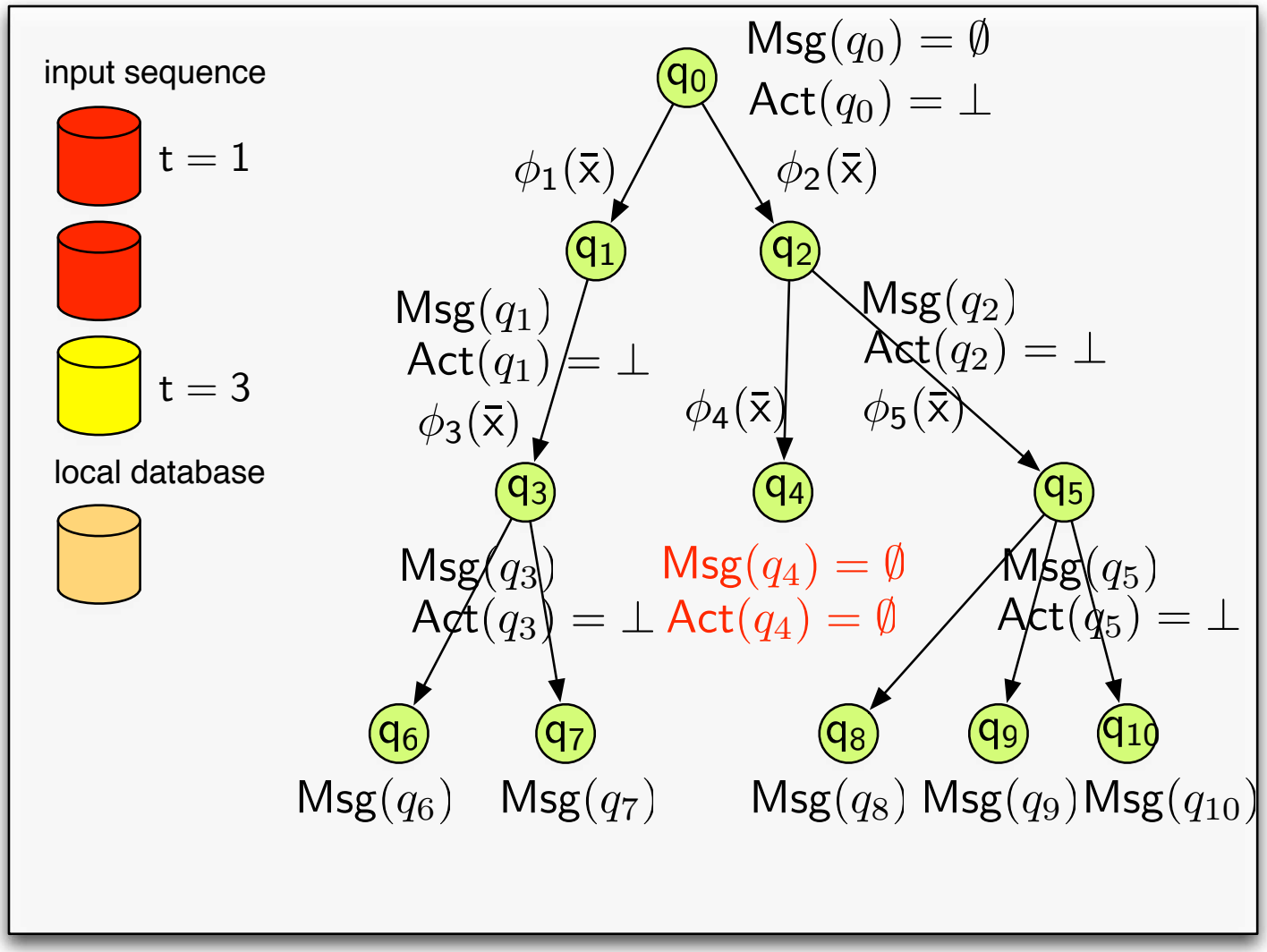
Run of a synthesized Web Service

1. Downward phase (downward using δ)



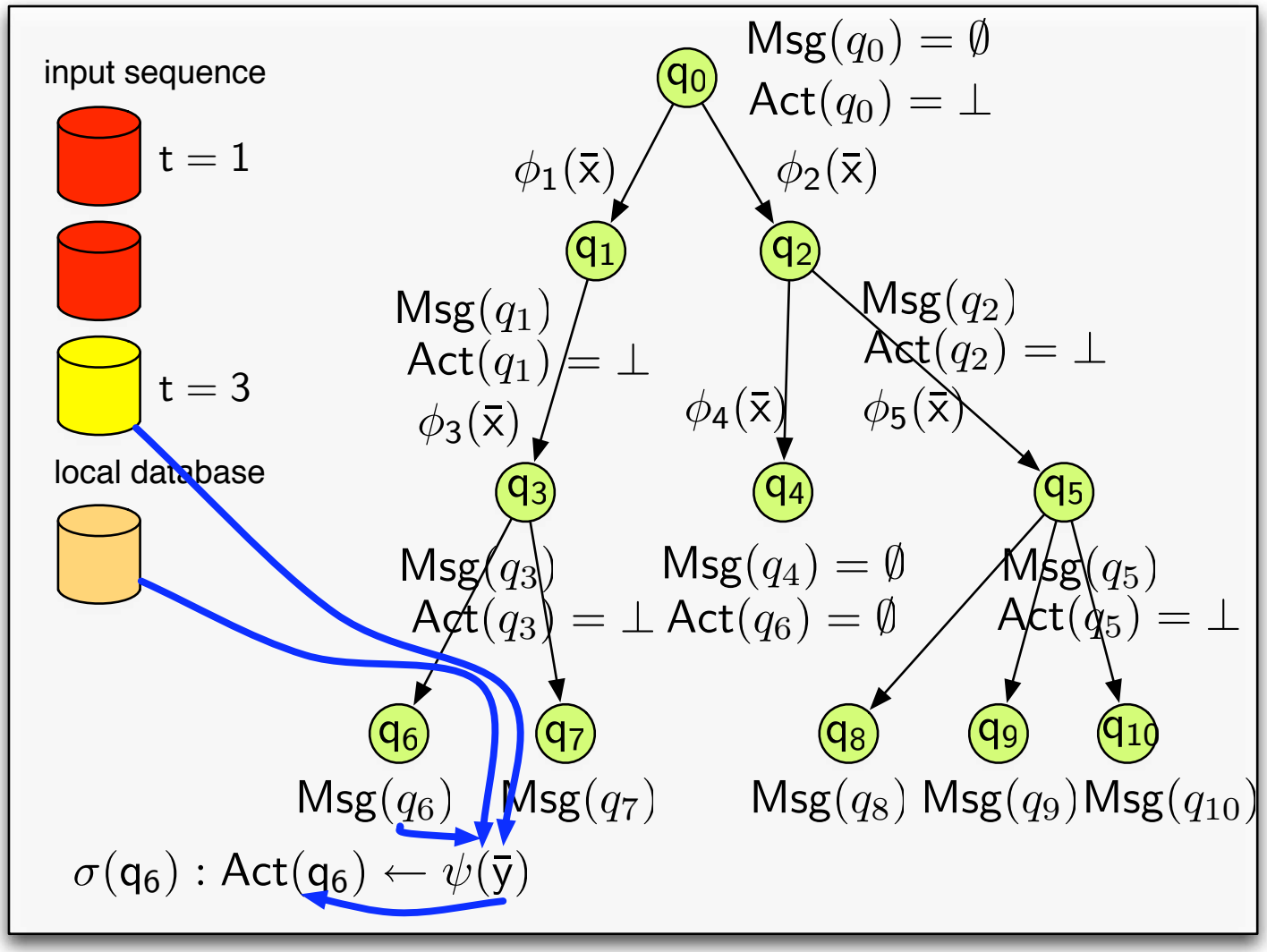
Run of a synthesized Web Service

1. Downward phase (downward using δ)



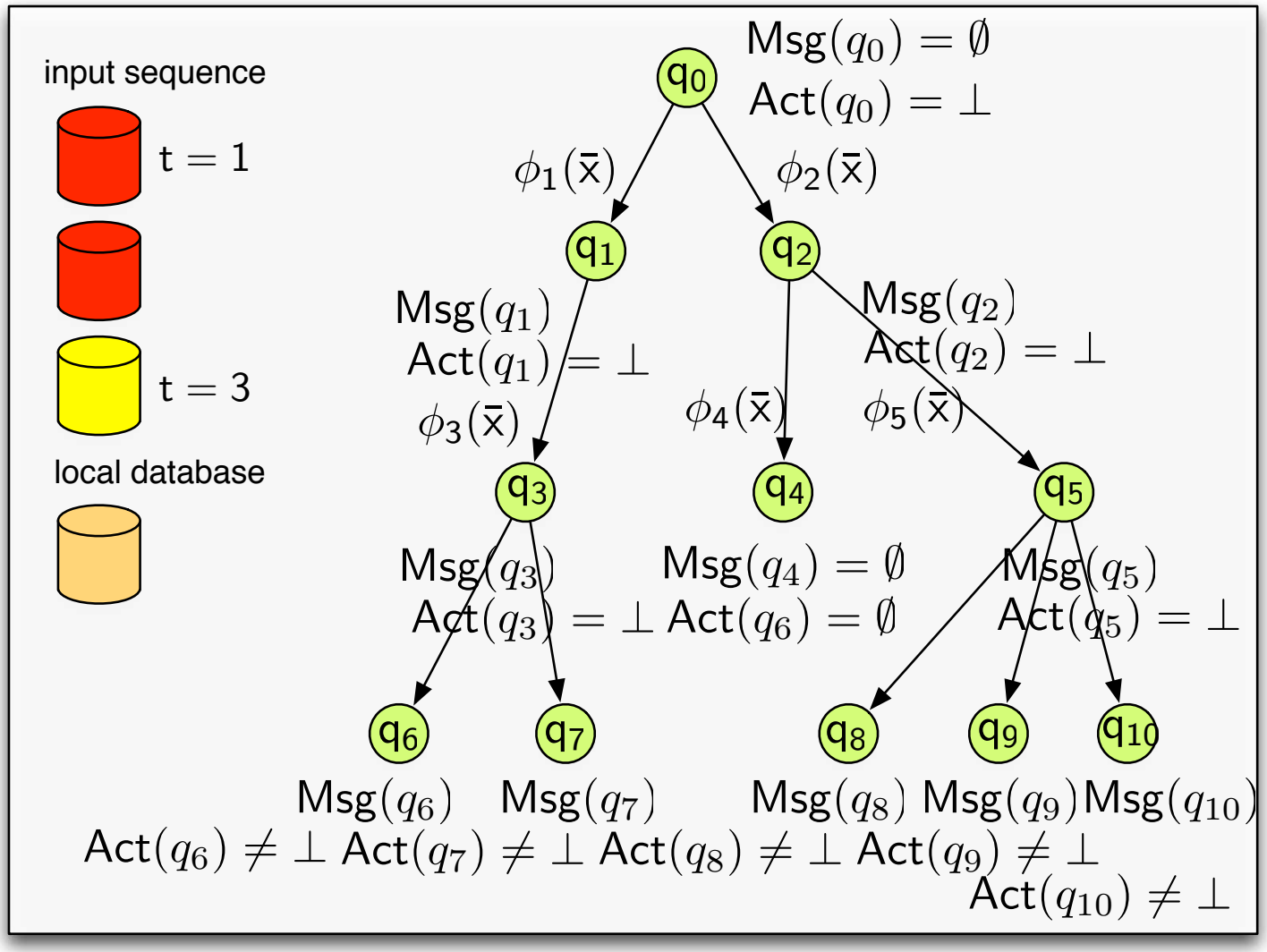
Run of a synthesized Web Service

2. Synthesis phase (upward using σ)



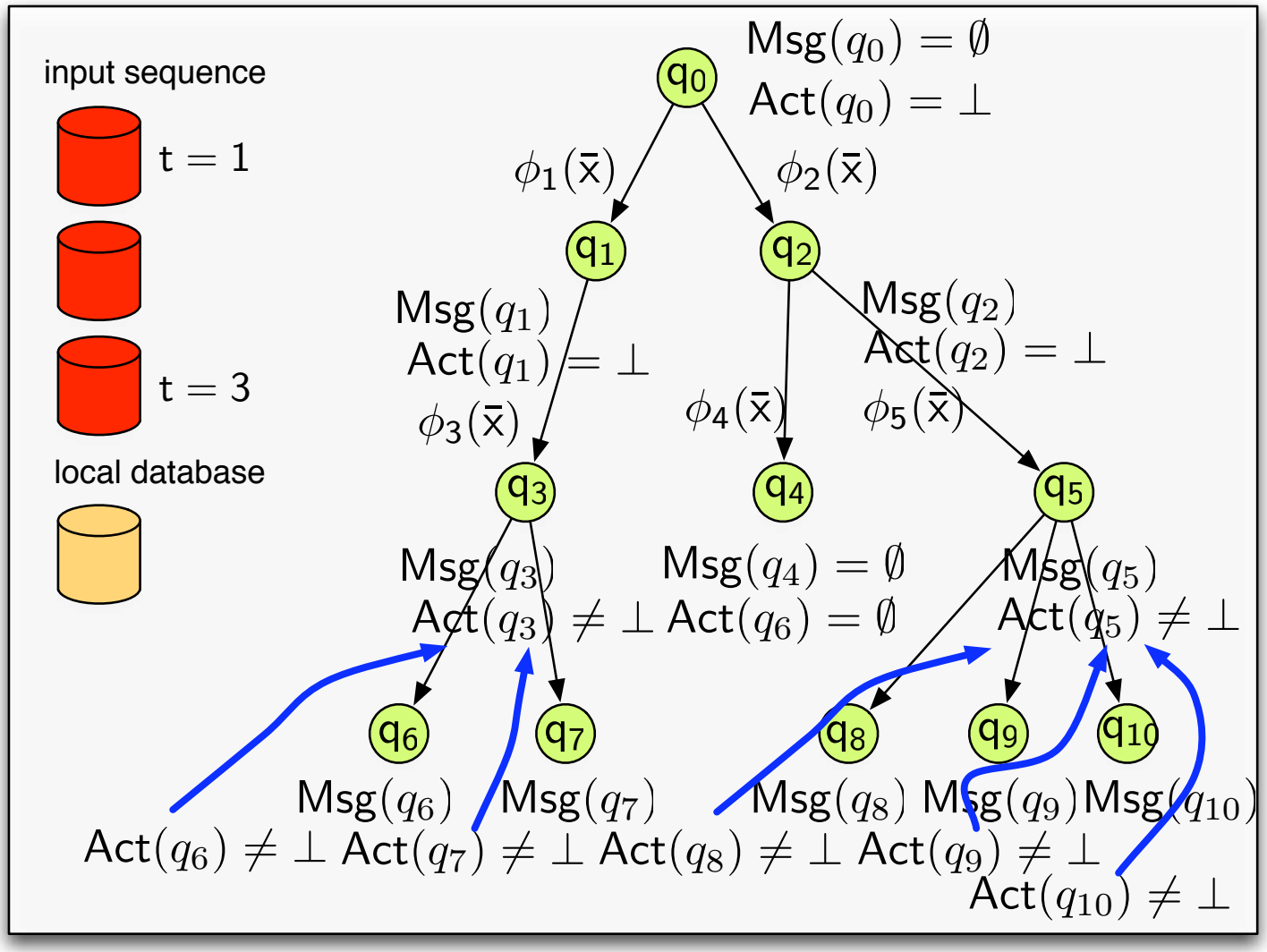
Run of a synthesized Web Service

2. Synthesis phase (upward using σ)



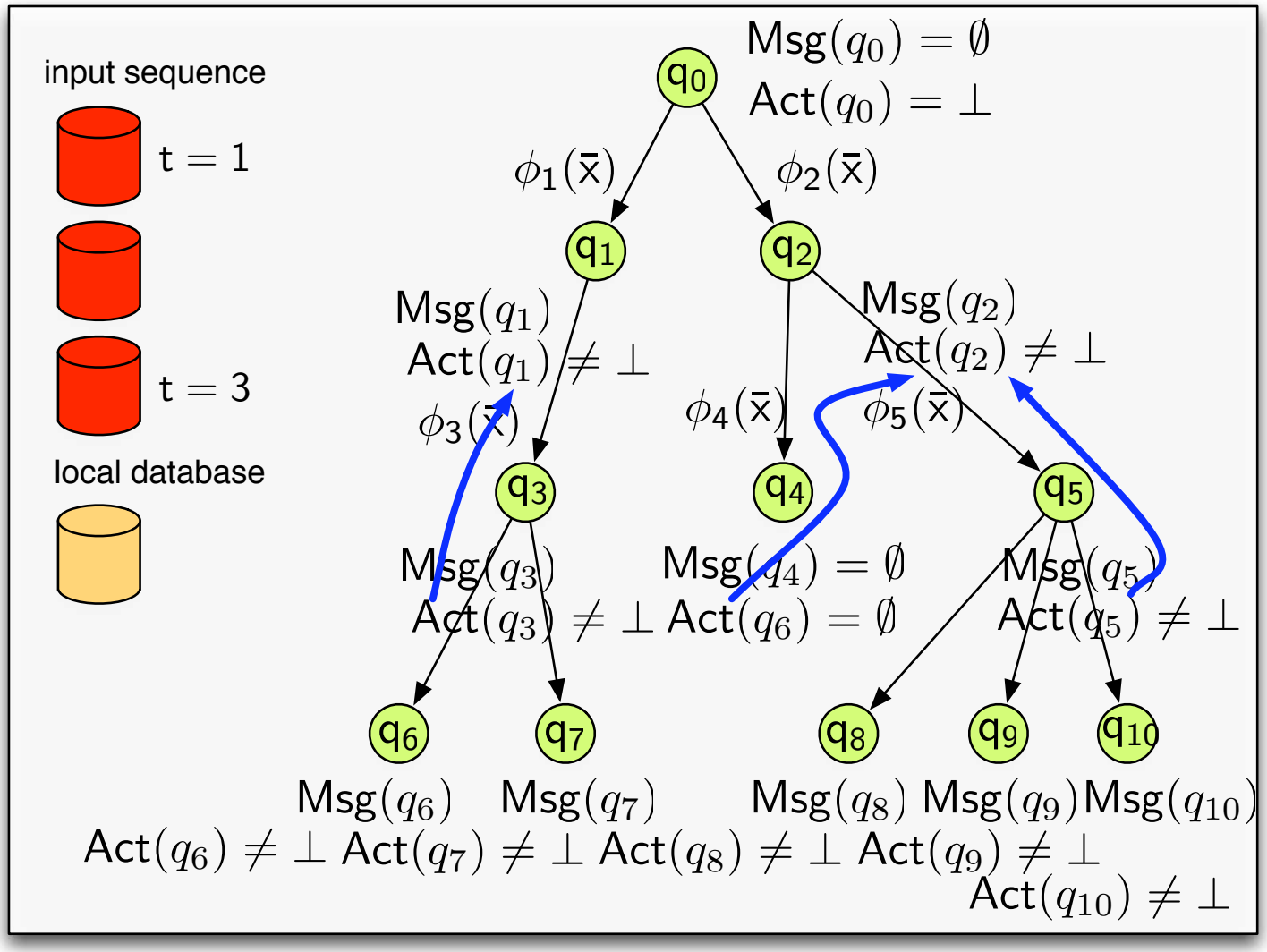
Run of a synthesized Web Service

2. Synthesis phase (upward using σ)



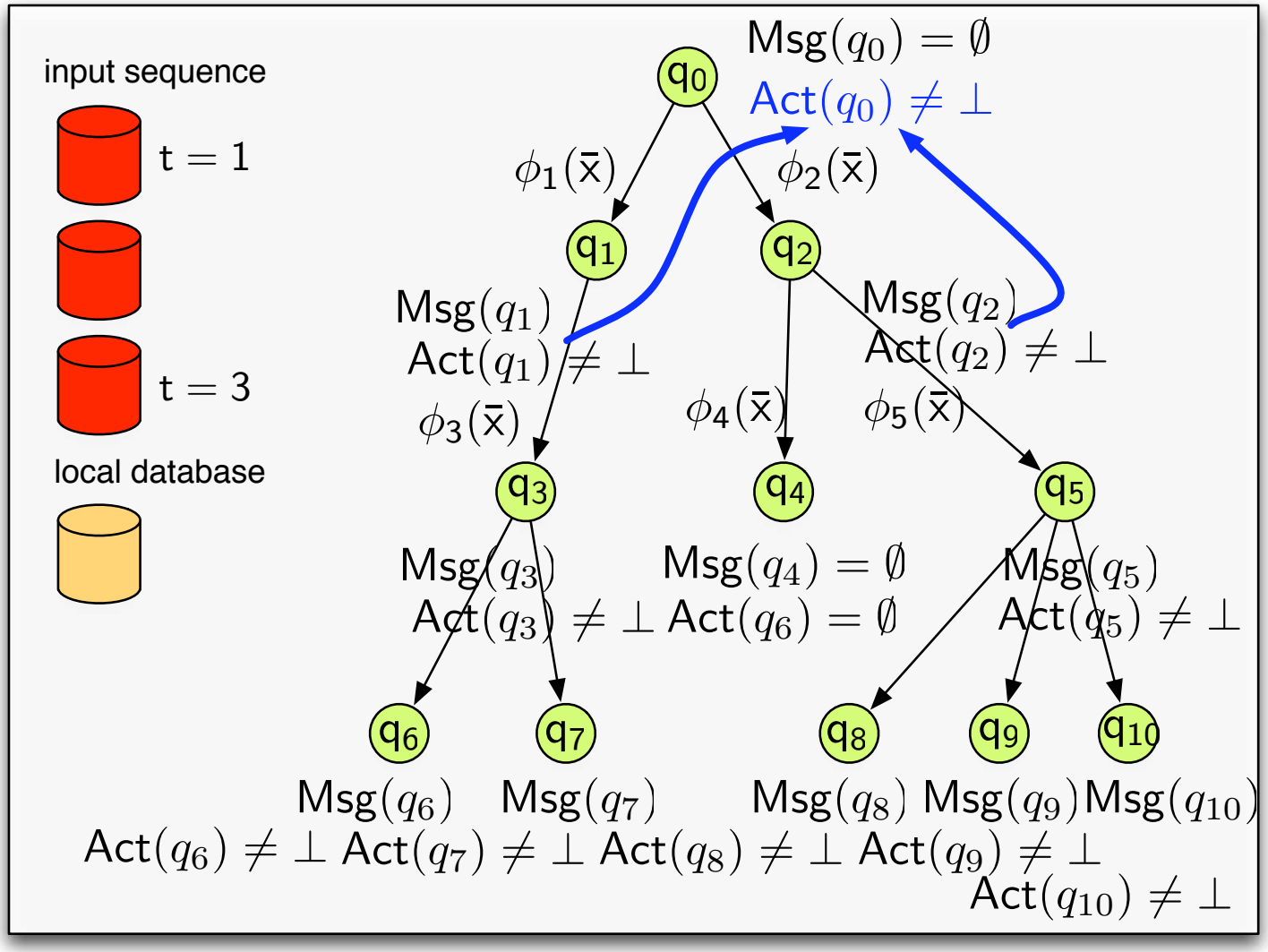
Run of a synthesized Web Service

2. Synthesis phase (upward using σ)



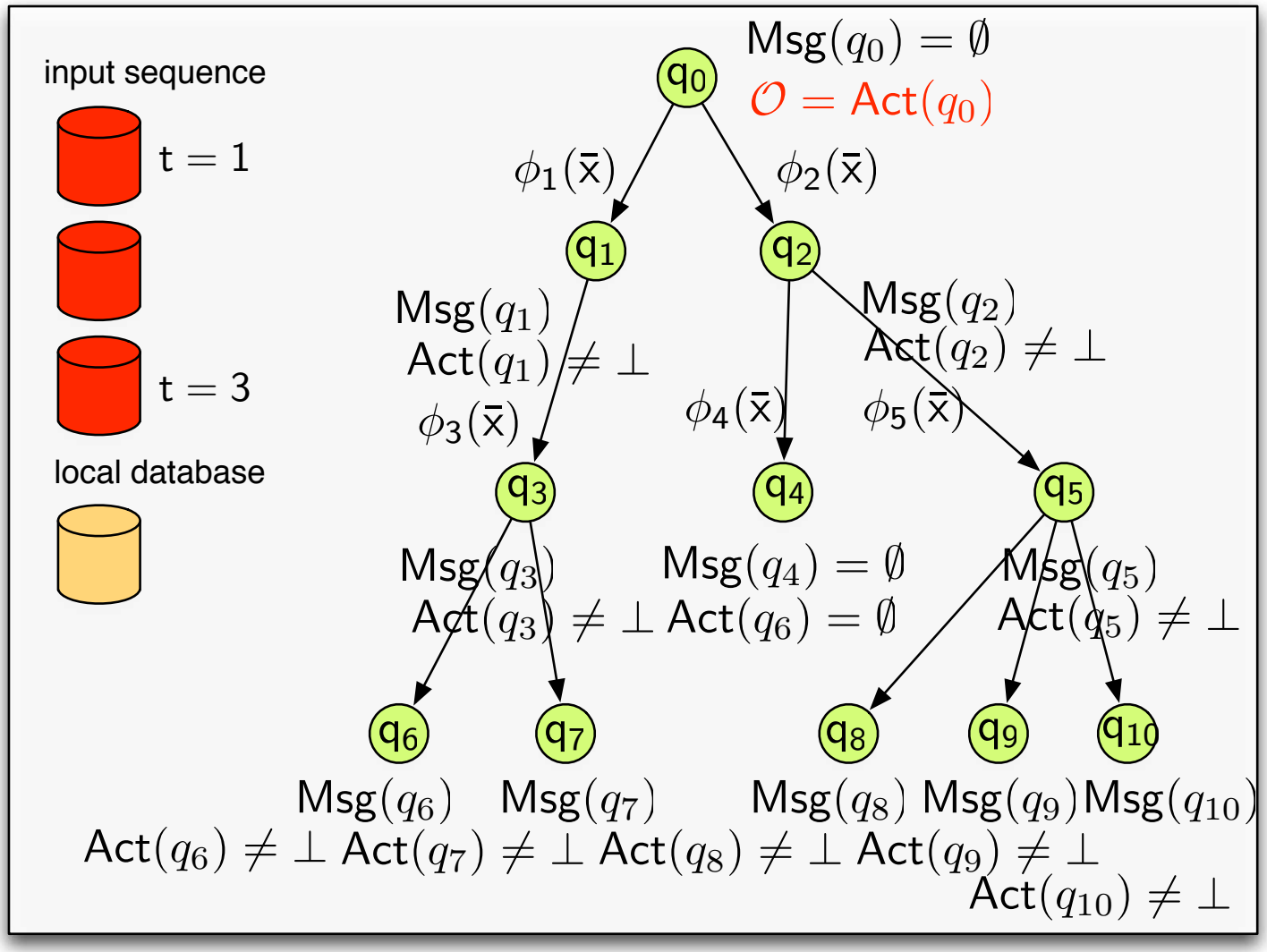
Run of a synthesized Web Service

2. Synthesis phase (upward using σ)



Run of a synthesized Web Service

3. Output



Different classes of SWS

- SWS ($\mathcal{L}_{\text{Msg}}$,$\mathcal{L}_{\text{Act}}$): parametrized by $\mathcal{L}_{\text{Msg}}$ and $\mathcal{L}_{\text{Act}}$.
  - SWS (FO,FO);
  - SWS (CQ,UCQ) (we allow $\neq$ in both CQ and UCQ)
- We also consider nonrecursive classes:
  - SWS_{nr}(FO,FO);
  - SWS_{nr}(CQ,UCQ).

Classical decision problems for SWS's.

Definition (non-emptiness problem)

Given an SWS over a schema $\mathcal{R}$ and R_{in}, does there exist an instance D of $\mathcal{R}$ and a sequence $\mathcal{I}$ of R_{in} such that $\tau(D, \mathcal{I}) \neq \emptyset$?

Definition (validation problem)

Given an SWS over a schema $\mathcal{R}$ and R_{in}, and a instance $\mathcal{O}$ of R_{out}, does there exist an instance D of $\mathcal{R}$ and a sequence $\mathcal{I}$ of R_{in} such that $\tau(D, \mathcal{I}) = \mathcal{O}$?

Definition (equivalence problem)

Given two SWS's τ_1 and τ_2 over the same schemas $\mathcal{R}$, R_{in} and R_{out}, is $\tau_1(D, \mathcal{I}) = \tau_2(D, \mathcal{I})$ for all instances D of $\mathcal{R}$ and input sequences $\mathcal{I}$ of R_{in}?

Non-emptiness problem for SWS's.

Theorem

The non-emptiness problem is

1. undecidable for SWS(FO, FO);
2. EXPTIME-complete for SWS(CQ, UCQ);
3. PSPACE-complete for SWS_{nr}(CQ, UCQ);

- Lower bounds: Reductions from satisfiability of FO queries (1); satisfiability single ground fact sirups (2); and Q3SAT (3).
- Upper bounds: tree-automata techniques (2) and satisfiability test for UCQ's (3).

Validation problem for SWS's.

Theorem

The validation problem is

1. undecidable for SWS(FO, FO);
2. undecidable for SWS(CQ, UCQ); and
3. NEXPTIME-complete for SWS_{nr}(CQ, UCQ).

- Lower bounds: Reductions from satisfiability of FO queries (1); satisfiability of deterministic 2-head machines (2); and halting problem for NEXPTIME Turing machines (3).
- Upper bound: small model property (3).

Equivalence problem for SWS's.

Theorem

The equivalence problem is

1. undecidable for SWS(FO, FO);
2. undecidable for SWS(CQ, UCQ); and
3. coNEXPTIME-complete for SWS_{nr}(CQ, UCQ).

- Similar techniques as for the validation problem.

Summary of results (classic decision problems)

	Non-emptiness	**Validation**	**Equivalence**
$\mathrm{SWS}_{\mathrm{nr}}$(FO, FO)	undecidable	undecidable	undecidable
SWS(CQ, UCQ)	EXPTIME-complete	undecidable	undecidable
$\mathrm{SWS}_{\mathrm{nr}}$(CQ, UCQ)	PSPACE-complete	NEXPTIME-complete	coNEXPTIME-complete

- For non-recursive $\mathrm{SWS}_{\mathrm{nr}}$(FO,FO) all problems are beyond reach;
- Allowing recursion but disallowing negation in FO only helps for the non-emptiness problem;
- Disallowing both recursion and negation in FO make all problems decidable (but still with high complexity); and

Composition of SWS's

How to compose SWS's? Using mediators...

- A mediator coordinates services by routing the output of one service to the input of another;
- A mediator receives and redirects messages but does not access local databases; and
- an SWS mediator also synthesizes actions produced by its component services by means of queries in a query language $\mathcal{L}_{\text{Act}}$.

SWS mediators

Definition (MDT$\mathcal{L}_{\text{Act}}$))

Over a set S of available SWS's defined on $\mathcal{R}$, R_{in} and R_{out}, an SWS mediator in MDT($\mathcal{L}_{\text{Act}}$) is defined as $\pi = (Q, \delta, \sigma, q_0)$ where Q and q_0 are as before and for each $q \in Q$ the transition rules in δ are of the form

$$q \rightarrow (q_1, \textsc{Eval}(\tau_1)), \ldots, (q_k, \textsc{Eval}(\tau_k)).$$

Here $\tau_i \in S$ and is referred to as a component service of π. The synthesis rule $\sigma(q)$ is a query $\psi \in \mathcal{L}_{\text{Act}}$ just as in the case of SWS's, except that at ψ is never allowed to access the local database D.

Run of an SWS mediator

Key differences with the run of a normal SWS are:

1. Msg relations are instantiated with the result of evaluation of a component service, i.e., $\textsc{Eval}(\tau)$ for $\tau \in S$.
2. The index of the last input messages I_j consumed by $\textsc{Eval}(\tau)$ when called in state q is passed on. Below that state, the next input message is I_{j+1}
   - Different branches may consume different inputs....
3. The synthesis query never accesses the local database.

For $\pi \in \mathrm{MDT}(\mathcal{L}_{\mathrm{Act}})$, we denote by $\pi(D, \mathcal{I})$ the result of executing π on D and $\mathcal{I}$.

The composition problem

- Given a class of goal SWS's $\mathcal{G}$;
- Given a class of SWS mediators $\mathcal{M}$; and
- Given a class of component SWS's $\mathcal{C}$.

Definition (CP($\mathcal{G}, \mathcal{M}, \mathcal{C}$))

Given a goal service $\tau \in \mathcal{G}$ and a finite set of component services $S \subset \mathcal{C}$, all defined over the same schemas $\mathcal{R}$, R_{in}, R_{out}, does there exist an SWS mediator $\pi \in \mathcal{M}$ over S such that π and τ are equivalent?

Here, π and τ are equivalent iff $\pi(D, \mathcal{I}) = \tau(D, \mathcal{I})$ for all D and $\mathcal{I}$.

Some remarks

- When considering the goal and mediator SWS's as queries, and the component services as views, then the composition problem is equivalent to the problem of equivalent query rewriting.
  - We leverage on rewriting techniques to establish some of our results;
  - Our result might shed some light on the query rewriting problem...
- Our notion of composition is different from the notion of composition studied e.g., for the Peer model.
  - Due to synthesis, we cannot interleave executions of component services.

Complexity of composition - undecidable cases

Theorem

The composition problem CP($\mathcal{G}, \mathcal{M}, \mathcal{C}$) is

1. undecidable when $\mathcal{G}, \mathcal{C}$ are SWS(FO, FO) and $\mathcal{M}$ is MDT(FO), even when $\mathcal{G}, \mathcal{M}, \mathcal{C}$ are all nonrecursive;

Proof.

Complexity of composition - undecidable cases

Theorem

The composition problem CP($\mathcal{G}, \mathcal{M}, \mathcal{C}$) is

1. undecidable when $\mathcal{G}, \mathcal{C}$ are SWS(FO, FO) and $\mathcal{M}$ is MDT(FO), even when $\mathcal{G}, \mathcal{M}, \mathcal{C}$ are all nonrecursive;

Proof.

1. by reduction from satisfiability of FO queries.

Complexity of composition - undecidable cases

Theorem

The composition problem CP($\mathcal{G}, \mathcal{M}, \mathcal{C}$) is

1. undecidable when $\mathcal{G}, \mathcal{C}$ are SWS(FO, FO) and $\mathcal{M}$ is MDT(FO), even when $\mathcal{G}, \mathcal{M}, \mathcal{C}$ are all nonrecursive;
2. undecidable when $\mathcal{G}, \mathcal{C}$ are SWS(CQ, UCQ) and $\mathcal{M}$ is MDT(UCQ), even when either $\mathcal{M}$ or $\mathcal{C}$ is nonrecursive;

Proof.

1. by reduction from satisfiability of FO queries.

□

Complexity of composition - undecidable cases

Theorem

The composition problem CP($\mathcal{G}, \mathcal{M}, \mathcal{C}$) is

1. undecidable when $\mathcal{G}, \mathcal{C}$ are SWS(FO, FO) and $\mathcal{M}$ is MDT(FO), even when $\mathcal{G}, \mathcal{M}, \mathcal{C}$ are all nonrecursive;
2. undecidable when $\mathcal{G}, \mathcal{C}$ are SWS(CQ, UCQ) and $\mathcal{M}$ is MDT(UCQ), even when either $\mathcal{M}$ or $\mathcal{C}$ is nonrecursive;

Proof.

1. by reduction from satisfiability of FO queries.
2. by reduction from the equivalence problem of SWS (CQ,UCQ)

□

Complexity of composition - decidable case

Theorem

The composition problem $\mathsf{CP}(\mathcal{G}, \mathcal{M}, \mathcal{C})$ is

1. in 2EXPSPACE when $\mathcal{G}, \mathcal{C}$ are $\mathsf{SWS}_{\mathsf{nr}}(\mathsf{CQ}, \mathsf{UCQ})$ and $\mathcal{M}$ is $\mathsf{MDT}_{\mathsf{nr}}(\mathsf{UCQ})$, *i.e.*, when services and mediators are all nonrecursive;

Proof.

□

Complexity of composition - decidable case

Theorem

The composition problem $\mathsf{CP}(\mathcal{G}, \mathcal{M}, \mathcal{C})$ is

1. in 2EXPSPACE when $\mathcal{G}, \mathcal{C}$ are $\mathsf{SWS}_{\mathsf{nr}}$(CQ, UCQ) and $\mathcal{M}$ is $\mathsf{MDT}_{\mathsf{nr}}$(UCQ), *i.e.*, when services and mediators are all nonrecursive;

Proof.

1. Can be reduced to equivalent query rewriting using views for UCQ with $\neq$ in EXPTIME. We show that this rewriting problem is in EXPSPACE (we did not find this result in the literature). □

Complexity of composition - decidable cases?

You might have noticed ... so far data-driven SWS's and recursion together always gives us undecidability.

- In the paper we describe a decidable case with recursive goals and mediators and non-recursive component services.
  - We restrict SWS's such that they correspond to unions of conjunctions of 2-way regular path queries (2RPQs);
  - Leverage on the decidable containment for 2RPQs [Calvanese, De Giacomo and Vardi. 2005]
  - Combine this with Duschka and Genesereth's maximally contained rewriting algorithm for DATALOG.
- We get an 2EXPTIME decidable case.

We don't know any other decidable recursive case yet for data-driven services.

Non-data driven synthesized web services

- We consider a special case of SWS's in which propositional logic PL is used for downward message and upward action passing.
- Since PL is defined over propositional variables, for SWS (PL,PL) and SWS_{nr}(PL,PL) to make sense, we require
  - The local database D of $\mathcal{R}$ to be empty empty;
  - Each I_i in the input sequence $\mathcal{I}$ of R_{in} consists of a set of propositional variables; and
  - Message/action registers, and output are Boolean values.

Relationship between SWS (PL,PL) and automata

- It is easily seen that for every alternating finite state automata (AFA) A, there is an equivalent SWS (PL,PL) τ that can be constructed in time $|A|$.
- Conversely, we need an exponential alphabet to encode all possible input sequence (recall that each I_i consists of a set of propositional variables). However, we can show the following:

Lemma

Any SWS τ in SWS (PL,PL) can be transformed into an NFA in time exponential in the size of τ.

Therefore, SWS (PL,PL) and it non-recursive counterpart $\mathrm{SWS}_{\mathrm{nr}}$(PL,PL) can be regarded as automata (taken into account an exponential blowup).
$\Rightarrow$ Roman model can be modeled in SWS (PL,PL).

Composition problem for DFA's

- In the Roman model, where Web services are modeled by deterministic finite state automata, the following notion of composition is proposed:
- Given n DFA $A_i = (Q_i, \Sigma_i, q_0^i, \delta_i : Q_i \times \Sigma_i \rightarrow Q_i)$, the asynchronous product of the A_i's is the non-deterministic automata:

$$A_1 \otimes \cdots \otimes A_n = (Q, \Sigma, \mathbf{q}, \delta : Q \times \Sigma \rightarrow \mathcal{P}(Q))$$

given by

  - $Q = Q_1 \times \cdots \times Q_n$;
  - $\Sigma = \bigcup_{i=1,\ldots,n} \Sigma_i$;
  - $\mathbf{q} = (q_0^1, \ldots, q_0^n)$; and
  - δ is defined by $\mathbf{t} \in \delta(\mathbf{s}, a)$ iff for some i, $t_i = \delta(s_i, a)$ and for $j \neq i$, $t_j = s_j$.

Composition problem for DFA's

- Given a DFA B, the composition problem is now to determine whether $B \preceq A_1 \otimes \cdots \otimes A_n$.
- Here, $\preceq$ is the standard simulation relation.
- This problem is known to be EXPTIME-complete
  - Lower bound: Muscholl and Walukiewicz'07;
  - Upper bound: Roman model [D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella.,2003]
- Our notion of composition is different since the synthesis requires the runs of the individual automata to complete; moreover, using an MDT(PL)-mediator one can take arbitrary Boolean combinations of the component automata.

Complexity of composition - PL cases

We do not know the answer for CP($\mathcal{G}, \mathcal{M}, \mathcal{C}$) where $\mathcal{G}$, $\mathcal{M}$ and $\mathcal{C}$ are all SWS (PL,PL).

Theorem

The composition problem CP($\mathcal{G}, \mathcal{M}, \mathcal{C}$) is

1. decidable when $\mathcal{G}$ is SWS_{nr}(PL,PL), $\mathcal{C}$ is SWS(PL, PL) and $\mathcal{M}$ is MDT(PL); and
2. decidable when $\mathcal{G}$ is SWS(PL,PL), $\mathcal{C}$ is SWS_{nr}(PL, PL) and $\mathcal{M}$ is MDT_{nr}(PL).

Proof.

Both (1) and (2) rely on a connection between SWS_{nr}(PL,PL) and *k*-prefix recognizable languages. The length of this prefix gives a bound on the mediator which can then be guessed. □

Complexity of composition - PL cases

Let MDT($\vee$) be the subclass of mediators in MDT(PL) that only use disjunctions in the synthesis rules.

Theorem

1. CP(SWS(PL, PL), MDT($\vee$), SWS(PL, PL)) is decidable in 3EXPSPACE;
2. CP(NFA, MDT($\vee$), SWS(PL, PL)) is 2EXPSPACE-complete, while CP(DFA, MDT($\vee$), SWS(PL, PL)) is in EXPSPACE;

- Upper bounds are established using the relationship between SWS (PL,PL) and automata; and
- The regular expression rewriting techniques of Calvanese, De Giacomo, Lenzerini and Vardi [2002] (which is 2EXPSPACE)

Complexity of composition - PL cases

Let $\text{MDT}^b(\text{PL})$ be that class that invokes each component service at most a fixed number of times in all transition rules combined, and that have bounded-size synthesis rules.

Theorem

CP(SWS(PL, PL), MDT^b(PL), SWS(PL, PL)) is in EXPSPACE;
CP(SWS(PL, PL), MDT^b(PL), SWS_{nr}(PL, PL)) is PSPACE-complete.

Upper bound: The boundedness restriction enables us to establish a small model property.

Summary of results (composition)

CP($\mathcal{G}, \mathcal{M}, \mathcal{C}$)			**Complexity**
$\mathcal{G}$: goal service	$\mathcal{M}$: mediator	$\mathcal{C}$:component	
SWS(FO,FO)	MDT(FO)	SWS(FO,FO)	undecidable
SWS_{nr}(FO,FO)	MDT_{nr}(FO)	SWS_{nr}(FO,FO)	undecidable
SWS(CQ,UCQ)	MDT(UCQ)	SWS(CQ,UCQ)	undecidable
SWS(CQ,UCQ)	MDT(UCQ)	SWS_{nr}(CQ,UCQ)	undecidable
SWS(CQ,UCQ)	MDT_{nr}(UCQ)	SWS(CQ,UCQ)	undecidable
SWS_{nr}(CQ,UCQ)	MDT_{nr}(UCQ)	SWS_{nr}(CQ,UCQ)	2EXPSPACE
SWS_{nr}(PL,PL)	MDT(PL)	SWS(PL,PL)	decidable
SWS(PL,PL)	MDT_{nr}(PL)	SWS_{nr}(PL,PL)	decidable
Special cases			
SWS(UC2RPQ)	MDT(UC2RPQ)	$\text{SWS}_{\text{nr}}(\text{CQ}^r)$	2EXPTIME
SWS(PL,PL)	MDT($\vee$)	SWS(PL,PL)	3EXPSPACE
NFA	MDT($\vee$)	SWS(PL, PL)	2EXPSPACE
DFA	MDT($\vee$)	SWS(PL, PL)	EXPSPACE
SWS(PL, PL)	MDT^b(PL)	SWS(PL, PL)	EXPSPACE
SWS(PL, PL)	MDT^b(PL)	SWS_{nr}(PL, PL)	PSPACE

Conclusion

- We provided a model for (data-driven) Web services (SWS's) that generalizes both automata and transducer-based models.
- Settled classical decision problems for various classes SWS $(\mathcal{L}_{\text{Msg}}, \mathcal{L}_{\text{Act}})$
- Defined SWS mediators for composing SWS's and showed various complexity results.
- Complexities are high and we need to look at further restrictions ...
- Matching lower bounds for complexities of composition ... ?
- Find out whether the general PL case decidable is or not.

Thank you!