Complexity and Composition of Synthesized Web Services

Wenfei Fan\textsuperscript{1,2} & Floris Geerts\textsuperscript{1} & Wouter Gelade\textsuperscript{3} & Frank Neve\textsuperscript{3} & Antonella Poggi\textsuperscript{4}

\textsuperscript{1} University of Edinburgh, UK
\textsuperscript{2} Bell Labs, US
\textsuperscript{3} Hasselt University & transnational University Limburg, Belgium
\textsuperscript{4} SAPIENZA Università di Roma, Italy
Before coming to Vancouver...

**Todo:**

1. Book Hotel
2. Book Plane
3. Check schedule bus/trains from/to airport
4. Register for Conference
5. Pack suitcase
Before coming to Vancouver...

1. Book Hotel → Web service
2. Book Plane → Web service
3. Check schedule bus/trains from/to airport → Web service
4. Register for Conference → Web service
5. Pack suitcase → Web service
Prevalent use of Web services highlights the need for studying their properties ...
Web services

standalone

input request → Web service → output

composite

input request → Web service → output

Web service

Web service

Web service
Data-driven Web services

standalone

input
request

Web service

output

composite

input
request

Web service

Web service

Web service
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.

⇒ 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 ⇒ 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
  \( \mathcal{W} = (\mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{A}, \mathbf{Q}_{\text{in}}, \mathbf{Q}_{\text{out}}, \mathbf{R}) \), 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
  - \( \mathbf{R} \) is a set of rules.
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 ...

(only book plane & hotel if you can get there and sleep there and get back home) xor (you can go on a cheap holiday in a sunny location)
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.

(only book plane & hotel if you can get there and sleep there and get back home) xor (you can go on a cheap holiday in a sunny location)
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

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

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 \( R_{out} \).
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 $\text{Msg}$ over the input schema $R_{\text{in}}$.

- $\mathcal{L}_{\text{Act}}$ for **upward action passing** and the generation of the output.

- We denote the results of these queries (i.e., upward actions) by relations $\text{Act}$ over the output schema $R_{\text{out}}$. 
A synthesized Web services (or SWS) $\tau$ over $R$, $R_{in}$, 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;
- $\delta$ consists of the set of transition rules; and
- $\sigma$ consists of the set of synthesis rules.
Synthesized Web services in $\mathcal{SWS}(\mathcal{L}_{\text{Msg}}, \mathcal{L}_{\text{Act}})$

**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 $\sigma(q)$:

$$
\delta(q) : \quad q \rightarrow (q_1, \phi_1(\bar{x}_1)), \ldots, (q_k, \phi_k(\bar{x}_k)).
$$

$$
\sigma(q) : \quad \text{Act}(q) \leftarrow \psi(\bar{y}),
$$

where the $\phi_i$’s are queries in $\mathcal{L}_{\text{Msg}}$ from $\mathcal{R}, \mathcal{R}_{in}, \mathcal{R}_{out}, \text{Msg}(q)$ to $\text{Mes}(q_i)$ and $\psi$ 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 $\mathcal{R}, \mathcal{R}_{in}, \text{Msg}(q)$ to $\text{Act}(q)$ if $k = 0$. 
Synthesised Web services: $\delta$- and $\sigma$-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 $\text{Msg}(q)$; 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. $\text{Act}(q_0)$ is the result of the execution of $\tau$ on $D$ and $\mathcal{I}$, denoted by $\tau(D, \mathcal{I})$. 
1. Downward phase (downward using $\delta$)

input sequence

- $t = 1$
- $t = 2$
- $t = 3$

local database

$q_0$

$\text{Msg}(q_0) = \emptyset$
$\text{Act}(q_0) = \bot$
Run of a synthesized Web Service

1. Downward phase (downward using $\delta$)

\[
\delta(q_0) : q_0 \rightarrow (q_1, \phi_1(\overline{x})), (q_2, \phi_2(\overline{x}))
\]
Run of a synthesized Web Service

1. Downward phase (downward using $\delta$)

\[ \delta(q_0) : q_0 \rightarrow (q_1, \phi_1(x)), (q_2, \phi_2(x)) \]
Run of a synthesized Web Service

1. Downward phase (downward using $\delta$)

\[\delta(q_0) : q_0 \rightarrow (q_1, \phi_1(\bar{x})), (q_2, \phi_2(\bar{x}))\]
Run of a synthesized Web Service

1. Downward phase (downward using $\delta$)

$$\delta(q_0) : q_0 \rightarrow (q_1, \phi_1(\bar{x})), (q_2, \phi_2(\bar{x}))$$
Run of a synthesized Web Service

1. Downward phase (downward using $\delta$)

- $t = 1$
- $t = 3$

Local database

Input sequence

$\delta(q_1) : q_1 \rightarrow (q_3, \phi_3(\bar{x}))$
Run of a synthesized Web Service

1. Downward phase (downward using $\delta$)

\[
\begin{align*}
\delta(q_2) : q_2 &\rightarrow (q_4, \phi_4(\bar{x})), (q_5, \phi_5(\bar{x})) \\
\phi_1(\bar{x}) &\rightarrow q_1 \leftarrow \phi_2(\bar{x}) \\
\phi_3(\bar{x}) &\rightarrow q_3 \leftarrow \phi_4(\bar{x}) \\
\phi_5(\bar{x}) &\rightarrow q_5 \leftarrow \phi_4(\bar{x}) \\
Msg(q_0) & = \emptyset \\
Act(q_0) & = \bot \\
Msg(q_1) & = \bot \\
Act(q_1) & = \bot \\
Msg(q_2) & = \bot \\
Act(q_2) & = \bot \\
Msg(q_3) & = \bot \\
Act(q_3) & = \bot \\
Msg(q_4) & = \bot \\
Act(q_4) & = \bot \\
Msg(q_5) & = \bot \\
Act(q_5) & = \bot
\end{align*}
\]
Run of a synthesized Web Service

1. Downward phase (downward using $\delta$)

\[ t = 1 \]
\[ t = 3 \]

\[ \delta(q_2) : q_2 \rightarrow (q_4, \phi_4(\bar{x})), (q_5, \phi_5(\bar{x})) \]
Run of a synthesized Web Service

1. Downward phase (downward using $\delta$)

```plaintext
input sequence
t = 1

t = 3
local database

Msg(q0) = ∅
Act(q0) = ⊥

φ₁(\overline{x})

Msg(q₁) = ⊥
Act(q₁) = ⊥

φ₂(\overline{x})

Msg(q₂) = ∅
Act(q₂) = ⊥

φ₃(\overline{x})

Msg(q₃) = ⊥
Act(q₃) = ⊥

φ₄(\overline{x})

Msg(q₄) = ⊥
Act(q₄) = ⊥

φ₅(\overline{x})

Msg(q₅) = ⊥
Act(q₅) = ⊥

q₀

q₁

q₂

q₃

q₄

q₅

q₆

q₇

q₈

q₉

q₁₀
```
Run of a synthesized Web Service

1. Downward phase (downward using $\delta$)

<table>
<thead>
<tr>
<th>t = 1</th>
<th>t = 3</th>
<th>local database</th>
</tr>
</thead>
</table>

- $\phi_1(\bar{x})$
- $\phi_2(\bar{x})$
- $\phi_3(\bar{x})$
- $\phi_4(\bar{x})$
- $\phi_5(\bar{x})$

- $\text{Msg}(q_0) = \emptyset$
- $\text{Act}(q_0) = \bot$
- $\text{Msg}(q_1) = \emptyset$
- $\text{Act}(q_1) = \bot$
- $\text{Msg}(q_2) = \emptyset$
- $\text{Act}(q_2) = \bot$
- $\text{Msg}(q_3) = \emptyset$
- $\text{Act}(q_3) = \bot$
- $\text{Msg}(q_4) = \emptyset$
- $\text{Act}(q_4) = \emptyset$
- $\text{Msg}(q_5) = \emptyset$
- $\text{Act}(q_5) = \bot$

- $\text{Msg}(q_6)$
- $\text{Msg}(q_7)$
- $\text{Msg}(q_8)$
- $\text{Msg}(q_9)$
- $\text{Msg}(q_{10})$
Run of a synthesized Web Service

2. Synthesis phase (upward using $\sigma$)

- **Input sequence**
  - $t = 1$
  - $t = 3$

- **Local database**

- **States and Transitions**
  - $q_0$ with $\phi_1(x)$ and $\phi_2(x)$
  - $q_1$ with $\phi_3(x)$
  - $q_2$ with $\phi_4(x)$ and $\phi_5(x)$
  - $q_3$ with $\phi_6(x)$
  - $q_4$ with $\phi_7(x)$
  - $q_5$ with $\phi_8(x)$
  - $q_6$ with $\phi_9(x)$
  - $q_7$ with $\phi_{10}(x)$

- **Messages and Actions**
  - $\text{Msg}(q_0) = \emptyset$
  - $\text{Act}(q_0) = \bot$
  - $\text{Msg}(q_1) = \bot$
  - $\text{Act}(q_1) = \bot$
  - $\text{Msg}(q_2) = \emptyset$
  - $\text{Act}(q_2) = \bot$
  - $\text{Msg}(q_3) = \bot$
  - $\text{Act}(q_3) = \bot$
  - $\text{Msg}(q_4) = \emptyset$
  - $\text{Act}(q_4) = \bot$
  - $\text{Msg}(q_5) = \emptyset$
  - $\text{Act}(q_5) = \bot$
  - $\text{Msg}(q_6) = \emptyset$
  - $\text{Msg}(q_7) = \emptyset$
  - $\text{Msg}(q_8) = \emptyset$
  - $\text{Msg}(q_9) = \emptyset$
  - $\text{Msg}(q_{10}) = \emptyset$

- **Action Assignment**
  - $\sigma(q_6) : \text{Act}(q_6) \leftarrow \psi(y)$
Run of a synthesized Web Service

2. Synthesis phase (upward using $\sigma$)
Run of a synthesized Web Service

2. Synthesis phase (upward using $\sigma$)

- Input sequence:
  - $t = 1$
  - $t = 3$

- Local database:

- Transition Q1:
  - $\phi_1(\bar{x})$
  - $\phi_2(\bar{x})$

- Transition Q2:
  - $\phi_4(\bar{x})$
  - $\phi_5(\bar{x})$

- Transition Q3:
  - $\phi_3(\bar{x})$

- Transition Q6:
  - $\phi_6(\bar{x})$
  - $\phi_7(\bar{x})$

- Transition Q8:
  - $\phi_8(\bar{x})$
  - $\phi_9(\bar{x})$
  - $\phi_{10}(\bar{x})$

- Transition Q10:

- Messages:
  - $Msg(q_0) = \emptyset$
  - $Act(q_0) = \bot$
  - $Msg(q_1) = \emptyset$
  - $Act(q_1) = \bot$
  - $Msg(q_2) = \emptyset$
  - $Act(q_2) = \bot$
  - $Msg(q_3) = \emptyset$
  - $Act(q_3) \neq \bot$
  - $Act(q_6) = \emptyset$
  - $Msg(q_4) = \emptyset$
  - $Act(q_4) = \bot$
  - $Msg(q_5) = \emptyset$
  - $Act(q_5) \neq \bot$
  - $Msg(q_7) = \emptyset$
  - $Act(q_7) = \bot$
  - $Msg(q_8) = \emptyset$
  - $Act(q_8) = \bot$
  - $Msg(q_9) = \emptyset$
  - $Act(q_9) = \bot$
  - $Msg(q_{10}) = \emptyset$
  - $Act(q_{10}) = \bot$
Run of a synthesized Web Service

2. Synthesis phase (upward using $\sigma$)
Run of a synthesized Web Service

2. Synthesis phase (upward using $\sigma$)

- Input sequence $t = 1$
- $t = 3$
- Local database

- $\phi_1(\bar{x})$
- $\phi_2(\bar{x})$
- $\phi_3(\bar{x})$
- $\phi_4(\bar{x})$
- $\phi_5(\bar{x})$

- $\text{Act}(q_0) \neq \bot$
- $\text{Act}(q_1) \neq \bot$
- $\text{Act}(q_2) \neq \bot$
- $\text{Act}(q_3) \neq \bot$
- $\text{Act}(q_4) \neq \bot$
- $\text{Act}(q_5) \neq \bot$
- $\text{Act}(q_6) \neq \bot$
- $\text{Act}(q_7) \neq \bot$
- $\text{Act}(q_8) \neq \bot$
- $\text{Act}(q_9) \neq \bot$
- $\text{Act}(q_{10}) \neq \bot$

- $\text{Msg}(q_0) = \emptyset$
- $\text{Msg}(q_1) = \emptyset$
- $\text{Msg}(q_2) = \emptyset$
- $\text{Msg}(q_3) = \emptyset$
- $\text{Msg}(q_4) = \emptyset$
- $\text{Msg}(q_5) = \emptyset$
- $\text{Msg}(q_6) = \emptyset$
- $\text{Msg}(q_7) = \emptyset$
- $\text{Msg}(q_8) = \emptyset$
- $\text{Msg}(q_9) = \emptyset$
- $\text{Msg}(q_{10}) = \emptyset$
Run of a synthesized Web Service

3. Output

input sequence

\[ t = 1 \]

\[ t = 3 \]

local database

\[ \text{Msg}(q_0) = \emptyset \]
\[ \mathcal{O} = \text{Act}(q_0) \]

\[ \phi_1(\bar{x}) \]

\[ \phi_2(\bar{x}) \]

\[ \phi_3(\bar{x}) \]

\[ \phi_4(\bar{x}) \]

\[ \phi_5(\bar{x}) \]

\[ \text{Msg}(q_1) \]
\[ \text{Act}(q_1) \neq \bot \]

\[ \text{Msg}(q_2) \]
\[ \text{Act}(q_2) \neq \bot \]

\[ \text{Msg}(q_3) \]
\[ \text{Act}(q_3) \neq \bot \]

\[ \text{Msg}(q_4) = \emptyset \]
\[ \text{Act}(q_6) = \emptyset \]

\[ \text{Msg}(q_5) \]
\[ \text{Act}(q_5) \neq \bot \]

\[ \text{Msg}(q_6) \]
\[ \text{Act}(q_6) \neq \bot \]

\[ \text{Msg}(q_7) \]
\[ \text{Act}(q_7) \neq \bot \]

\[ \text{Msg}(q_8) \]
\[ \text{Act}(q_8) \neq \bot \]

\[ \text{Msg}(q_9) \]
\[ \text{Msg}(q_9) \neq \bot \]

\[ \text{Act}(q_9) \neq \bot \]

\[ \text{Msg}(q_{10}) \]
\[ \text{Act}(q_{10}) \neq \bot \]
Different classes of SWS

- SWS \((L_{Msg}, L_{Act})\): parametrized by \(L_{Msg}\) and \(L_{Act}\).
  - 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 $I$ of $R_{in}$ such that $\tau(D, I) \neq \emptyset$?

**Definition (validation problem)**

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

**Definition (equivalence problem)**

Given two SWS’s $\tau_1$ and $\tau_2$ over the same schemas $\mathcal{R}$, $R_{in}$ and $R_{out}$, is $\tau_1(D, I) = \tau_2(D, I)$ for all instances $D$ of $\mathcal{R}$ and input sequences $I$ of $R_{in}$?
The non-emptiness problem is

1. undecidable for SWS(FO, FO);
2. \textsc{Exptime}-complete for SWS(CQ, UCQ);
3. \textsc{Pspace}-complete for SWS\textsubscript{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).
The validation problem is

1. undecidable for SWS(FO, FO);
2. undecidable for SWS(CQ, UCQ); and
3. \textsc{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 \textsc{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)

<table>
<thead>
<tr>
<th></th>
<th>Non-emptiness</th>
<th>Validation</th>
<th>Equivalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWS(_{nr})(FO, FO)</td>
<td>undecidable</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
<tr>
<td>SWS(CQ, UCQ)</td>
<td>EXPTIME-complete</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
<tr>
<td>SWS(_{nr})(CQ, UCQ)</td>
<td>PSPACE-complete</td>
<td>NEXPTIME-complete</td>
<td>coNEXPTIME-complete</td>
</tr>
</tbody>
</table>

- For non-recursive SWS\(_{nr}\)(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
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 $\mathcal{L}_{\text{Act}}$. 
Definition (MDT $\mathcal{L}_{\text{Act}}$))

Over a set $S$ of available SWS’s defined on $R$, $R_{in}$ and $R_{out}$, an SWS mediator in $\text{MDT}(\mathcal{L}_{\text{Act}})$ 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 $\delta$ are of the form

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

Here $\tau_i \in S$ and is referred to as a component service of $\pi$. 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 $\psi$ 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., $\text{Eval}(\tau)$ for $\tau \in S$.

2. The index of the last input messages $I_j$ consumed by $\text{Eval}(\tau)$ 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 $\pi \in \text{MDT}(L_{\text{Act}})$, we denote by $\pi(D, I)$ the result of executing $\pi$ on $D$ and $I$. 
The composition problem

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

**Definition (CP($G, M, C$))**

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

Here, $\pi$ and $\tau$ are equivalent iff $\pi(D, I) = \tau(D, I)$ for all $D$ and $I$. 
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.
The composition problem \( CP(\mathcal{G}, \mathcal{M}, \mathcal{C}) \) is

1. **undecidable** when \( \mathcal{G}, \mathcal{C} \) are \( \text{SWS(FO, FO)} \) and \( \mathcal{M} \) is \( \text{MDT(FO)} \),
even when \( \mathcal{G}, \mathcal{M}, \mathcal{C} \) are all nonrecursive;
Complexity of composition - undecidable cases

Theorem

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

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

Proof.

1. by reduction from satisfiability of FO queries.
Theorem

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

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

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

Proof.

1. by reduction from satisfiability of FO queries.
Theorem

The composition problem \( CP(G, M, C) \) is

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

2. **undecidable** when \( G, C \) are \( SWS(CQ, UCQ) \) and \( M \) is \( MDT(UCQ) \), even when either \( M \) or \( C \) is nonrecursive;

Proof.

1. by reduction from satisfiability of FO queries.
2. by reduction from the equivalence problem of SWS (CQ, UCQ)
The composition problem $\text{CP}(G, M, C)$ is

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

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

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

Proof.

1. Can be reduced to equivalent query rewriting using views for $\text{UCQ}$ with $\neq$ in $\text{EXPTIME}$. We show that this rewriting problem is in $\text{EXPSPACE}$ (we did not find this result in the literature). □
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.
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 $R$ to be empty empty;
- Each $I_i$ in the input sequence $I$ of $R_{in}$ consists of a set of propositional variables; and
- Message/action registers, and output are Boolean values.
Relationship between $SWS (\mathcal{PL}, \mathcal{PL})$ and automata

- It is easily seen that for every alternating finite state automata (AFA) $A$, there is an equivalent $SWS (\mathcal{PL}, \mathcal{PL}) \tau$ 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 $SWS \tau$ in $SWS (\mathcal{PL}, \mathcal{PL})$ can be transformed into an NFA in time exponential in the size of $\tau$.

Therefore, $SWS (\mathcal{PL}, \mathcal{PL})$ and its non-recursive counterpart $SWS_{nr}(\mathcal{PL}, \mathcal{PL})$ can be regarded as automata (taken into account an exponential blowup).

$\Rightarrow$ Roman model can be modeled in $SWS (\mathcal{PL}, \mathcal{PL})$. 
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 $A_i = (Q_i, \Sigma_i, q_i^0, \delta_i : Q_i \times \Sigma_i \rightarrow Q_i)$, the asynchronous product of the $A_i$’s is the non-deterministic automata:

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

given by

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

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

**Theorem**

The composition problem \( CP(G, M, C) \) is
1. **decidable** when \( G \) is \( SWS_{nr}(PL,PL) \), \( C \) is \( SWS(PL, PL) \) and \( M \) is \( MDT(PL) \); and
2. **decidable** when \( G \) is \( SWS(PL,PL) \), \( C \) is \( SWS_{nr}(PL, PL) \) and \( M \) 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.
Let $\text{MDT}(\lor)$ be the subclass of mediators in $\text{MDT}(\text{PL})$ that only use disjunctions in the synthesis rules.

**Theorem**

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

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

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

Upper bound: The boundedness restriction enables us to establish a small model property.
Summary of results (composition)

<table>
<thead>
<tr>
<th>Complexity</th>
<th>CP((G, M, C))</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>undecidable</td>
<td>SWS(FO,FO)</td>
<td>MDT(FO)</td>
</tr>
<tr>
<td></td>
<td>SWS_{nr}(FO,FO)</td>
<td>MDT_{nr}(FO)</td>
</tr>
<tr>
<td>undecidable</td>
<td>SWS(CQ,UCQ)</td>
<td>MDT(UCQ)</td>
</tr>
<tr>
<td></td>
<td>SWS(CQ,UCQ)</td>
<td>MDT(UCQ)</td>
</tr>
<tr>
<td></td>
<td>SWS_{nr}(CQ,UCQ)</td>
<td>MDT_{nr}(UCQ)</td>
</tr>
<tr>
<td>2EXPSPACE</td>
<td>SWS_{nr}(PL,PL)</td>
<td>MDT(PL)</td>
</tr>
<tr>
<td></td>
<td>SWS(PL,PL)</td>
<td>MDT_{nr}(PL)</td>
</tr>
<tr>
<td>decidable</td>
<td>SWS(UC2RPQ)</td>
<td>MDT(UC2RPQ)</td>
</tr>
<tr>
<td>2EXPSPACE</td>
<td>SWS(PL,PL)</td>
<td>MDT(\lor)</td>
</tr>
<tr>
<td>3EXPSPACE</td>
<td>NFA</td>
<td>MDT(\lor)</td>
</tr>
<tr>
<td>2EXPSPACE</td>
<td>DFA</td>
<td>MDT(\lor)</td>
</tr>
<tr>
<td>EXPSPACE</td>
<td>SWS(PL, PL)</td>
<td>MDT^{b}(PL)</td>
</tr>
<tr>
<td>EXPSPACE</td>
<td>SWS(PL, PL)</td>
<td>MDT^{b}(PL)</td>
</tr>
<tr>
<td>PSPACE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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!