Specification and Verification of Data-driven Web Services

Alin Deutsch, Liying Sui, Victor Vianu
UCSD

presented by: Grigoris Karvounarakis
Univ. of Pennsylvania
CIS 650
November 4, 2004
Data-driven Web Services

- Interconnected interactive Web pages, built on top of a database (e.g., online shopping sites)
  - User input through text boxes, buttons and pull-down menus, usually dynamically generated
  - Access information in databases (e.g., product inventory of online store)
  - Action: transition to a new web page posting information that is output by previous step + (possibly) new menus etc for more input.
Example

button("clear")

HOMEPAGE ($W_0$)

login
password

login  register  clear

button("register")

REGISTRATION PAGE

button("login") \&
user(login, passwd)

button("login") \&
\neg user(login, passwd)

CUSTOMER PAGE

user-profile(login)
menu from product-list

user

product-list

<table>
<thead>
<tr>
<th>login</th>
<th>passwd</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>price</td>
</tr>
</tbody>
</table>
Example

HOMEPAGE (\(W_0\))

- button("clear")

- button("register")

- button("login") \&
  - user(login, passwd)

REGISTRATION PAGE

- menu from product-list

CUSTOMER PAGE

- user-profile(login)

ERROR PAGE (\(W_e\))

- "internal" errors

user-profile(login)

<table>
<thead>
<tr>
<th>user</th>
<th>product-list</th>
</tr>
</thead>
<tbody>
<tr>
<td>login</td>
<td>name</td>
</tr>
<tr>
<td>passwd</td>
<td>price</td>
</tr>
</tbody>
</table>
Outline

- High-level Specification model
- Automatic Verification
  - FO-LTL
  - CTL/CTL*
- Conclusions - Discussion
Model for data-driven Web services

- A web service $W$ is a set of web page schemas as well as
  - A global database $D$ (no updates)
  - State relations $S$ (representing the current state of a corresponding infinite state machine)
  - Input relations $I$
  - Action relations $A$
  - A home page schema $W_0$
  - An error page schema $W_e$ (captures “internal”/unrecoverable errors in that prohibit correct execution, not application-semantic errors, e.g., invalid passwd)
Web page schemas

- Input choices (e.g., options of a pull-down menu)
  - Options = \( \varphi(\text{Database, State, Past input}) \)
- Actions (i.e., output of transition)
  - \( \varphi(\text{Database, State, Input+Past input}) \)
- Web Page Transitions
  - Target Webpage: \( \varphi(\text{Database, State, Input+Past input}) \)
- State changes
  - insertions/deletions := \( \varphi(\text{Database, State, Input+Past input}) \)
  - CWA: non-existence = negation

- \( \varphi \): FO query
WP schema of example

• Input: login, passwd, button

• Input Rules:
  \[ \text{Options}_{\text{button}}(x) \leftarrow x=\text{"login"} \lor x=\text{"register"} \lor x=\text{"clear"} \]

• State Rules (this example: one state relation \text{error}):  
  \[ \text{error(\"failed login\")} \leftarrow \text{button(\"login\") \land \neg \text{user}(\text{login}, \text{passwd})} \]

• Target WP: HomePage, RegPage, CustPage

• Target Rules:
  \[ \text{HomePage} \leftarrow \text{button(\"clear\")} \]
  \[ \text{RegPage} \leftarrow \text{button(\"register\")} \]
  \[ \text{CustPage} \leftarrow \text{button(\"login\") \land \text{user}(\text{login}, \text{passwd})} \]
  \[ \text{HomePage} \leftarrow \text{button(\"login\") \land \neg \text{user}(\text{login}, \text{passwd})} \]
Web Service Runs

- An infinite sequence of pairs of the “current” Web page with the corresponding snapshots of the db relations (input, previous_input, state, action):

- A run goes through the special page $W_e$ iff in the previous state:
  - Some formula required some input before it was provided
  - The user is required to enter some input more than once
  - The target rules do not identify uniquely a target WP

- At step $i+1$, prev_input is updated with the input of the step $i$ and the new input, action, state is the result of applying the corresponding rules to the contents of the db at step $i$
Desirable Properties

- **Basic Soundness**
  - Every run is error-free (i.e., does not go through $W_e$)

- **Semantic (application-specific)**
  - No product is delivered before correct payment is received
  - No user can cancel an order that has already been shipped

- **Navigational properties**
  - There is a way to reach the home page from any page
Verification of Web Services

- Different expressiveness is required for different properties

- Linear-time properties (LTL-FO)
  - Satisfied by every run (e.g., basic and semantic properties in previous slide)

- Branching-time properties (CTL/CTL*)
  - Involve quantification over runs (e.g., navigational property in previous slide)
First-Order Linear-time Temporal Logic

- Temporal operators
  - $\mathbf{X}p$: $p$ holds in the next time
  - $p \mathbf{U} q$: $p$ holds until $q$ holds
  - $\mathbf{F}p$: $p$ holds eventually
  - $\mathbf{G}p$: $p$ always holds
  - $p \mathbf{B} q$: either $q$ always holds or $p$ holds before $q$ fails
Example

- No product is delivered before correct payment is received

\[ \forall \text{pid, uname, price} \left[ \xi(\text{pid, uname, price}) \land \neg \text{Ship(uname, pid)} \right] \]

where \( \xi(\text{pid, uname, price}) \) is the formula:

\[ \text{PaymentPage} \land \text{pay(price)} \land \text{button(“authorize payment“)} \land \text{pick(pid, price)} \land \exists \text{pname catalog(pid, pname, price)} \]

Color code: input state action data
Decidability of verification

- Even just deciding whether a Web service is error-free is undecidable
- Need restrictions: Input-boundedness
  - Input rules: \( \exists^* \text{FO} \) with ground state atoms
  - All other rules: quantification only on variables that are bounded by expressions on (current or past) input relations

Given an input bounded Web Service \( W \) and an input bounded LTL-FO formula \( \varphi \), checking \( W \models \varphi \) is PSPACE-complete
Not input-bounded formula

- No product is delivered before correct payment is received

\[ \forall \text{pid, uname, price } [\xi(\text{pid, uname, price}) \implies \neg \text{Ship(uname, pid)}] \]

where \( \xi(\text{pid, uname, price}) \) is the formula:

- \( \text{PaymentPage} \land \text{pay(price)} \land \text{button(“authorize payment“)} \land \text{pick(pid, price)} \land \exists \text{pname catalog(pid, pname, price)} \)

Color code: input state action data
Input-bounded formula

• No product is delivered before correct payment is received

\[ \forall \text{pid, uname, price} \ [\xi(\text{pid, uname, price}) \mathcal{B} \neg\text{Ship(uname, pid)}] \]

where \(\xi(\text{pid, uname, price})\) is the formula:

\[
\text{PaymentPage} \land \text{pay(price)} \land \text{button(“authorize payment”) \land}
\]

\[
\text{pick(pid, price)} \land \text{prod-price(pid, price)}
\]

Note: pick is a state relation, but it essentially keeps track of previous input(?)
Boundaries of decidability

- Relaxing the requirement that state atoms must be ground in formula defining the input options, by allowing state atoms with variables.

  **Reduction:** Does TM halt on input epsilon?

- Lifting the input-bounded requirement by allowing projection in state rules (i.e., existential quantification over state relations).

  **Reduction:** Implication for FDs and IDs

- Allowing Prev_I to record all previous input to I rather than the most recent one.

  **Reduction:** Trakhtenbrot’s Theorem

- Extend the LTL-FO formula with path quantification.

  **Reduction:** validity of $\exists^*\forall^*FO$ formulas
Branching-time Temporal Logic

- **CTL (Computation Tree Logic)** introduces path quantifiers:
  - $A$: “for every path”
  - $E$: “there exists a path”

- **Examples**
  - From every page (i.e., the target of every path) there always exists a path that eventually reaches HomePage:
    \[ A \ G \ E \ F \ \text{HomePage} \]
  - From every path, always if the current page is HomePage and login is selected, then there exists a path leading eventually to a page where one can authorize payment:
    \[ A \ G \ [ \ (\text{HP} \ \land \ \text{button(“login”)}) \ \rightarrow \ E \ F \ (\text{button(“authorize”)}) ] \]
Verification of CTL-FO

- Input-boundedness is not enough for decidability of CTL-FO verification
- Further restriction: Propositional input-bounded Web Services
  - Actions and states are propositional (no data values)
  - No rules can use PrevI atoms
  - Input rules need not be propositional
- Propositional CTL formulas
  - Use input, action, state relation names and WP symbols (viewed as propositions)
Input-bounded propositional CTL-FO

- Verification of $W \models \varphi$ for CTL(*) formulas for propositional Web services:
  - $\text{CO-NEXPTIME}$ for CTL
  - $\text{EXPSPACE}$ for CTL*
Other restrictions

- Restrict formulas to navigational properties (i.e. only including web page symbols - no relations)
  - \textbf{PSPACE} for CTL*
- Don't allow relations in input rules (i.e. specification does not use database at all)
  - \textbf{PSPACE} for CTL*
- Input-driven search: allow limited use of Prev_I
  - \textbf{EXPTIME}-complete for CTL
  - \textbf{2-EXPTIME}-complete for CTL*
Conclusions

- Extend the model and verification results to allow:
  - multiple users
  - custom-defined sessions to be verified (instead of all possible runs) - would allow updates (e.g., inventory updates after purchase) between sessions
  - interacting Web services

- Practical verification algorithms
  - Heuristics could help in achieving good performance
Discussion

• Similar results would be interesting/useful for other service models as well
• Proposed model is quite limited (although more powerful than Spielmann’s)
  ■ Still, it is limited even further to achieve decidability of verification (e.g., input-bounded, propositional, not using previous inputs)
  ■ What kinds of practical applications can/cannot be expressed?
• Other verification approaches (e.g., approximation: sound but not complete) could possibly be employed to deal with undecidability in models of practical interest
Discussion

- Important idea: coupling logic/verification with databases/Datalog
  - Using a database to represent the state and transition function of a machine that manipulates data
  - Useful ideas that can be applied in other models
- Useful results:
  - Introduces a model that is suitable an interesting kind of Web services
  - Proposes restrictions to achieve decidability of the verification of properties requiring different levels of expressiveness and provides thorough complexity results
  - Identifies boundaries of decidability of the verification of properties requiring different levels of expressiveness