Lecture 9

Training in GenieScript

By Monica Lam
Genie for MultiWOZ

- Methodology
  - Get a WOZ dataset for a task
  - Extract a state machine manually
  - Use Genie templates to write a generic state machine
    - With varieties, including co-references and ellipses
  - The same state machine is used for synthesis and execution
- Results:
  - 98% of MultiWOZ user utterances representable in ThingTalk (2% out of domain)
  - 84% of MultiWOZ turns are synthesizable by a generic state machine

Quiz: How expensive is this approach?
Quiz: How generally applicable is the MultiWOZ-derived state machine?
Key Hypothesis in GenieScript

*Our Motto: “To understand every relevant statement”.*

- A conversational agent
  - Factored into domain schemas and agent policy

- Can train contextual semantic parser with data synthesized from only the schemas, for *any* agent policy

- Write the generic state machine (GSM) ONCE by EXPERTS
  - To exercise all the conversational transitions
    - with user & agent initiatives
  - GSM used in synthesis
  - GSM without agent initiatives used to handle user initiatives
Developers’ Task

• Developers write the agent policy
  • In GenieScript (JavaScript) with regular control constructs (not a state machine)
  • Can be updated without resynthesizing data and training!

• GenieScript creates a bootstrap agent automatically
  • Test the agent to refine agent policy
  • Use the agent to annotate the few-shot only for failed parses
Key Concepts

- Conversational semantics:
  Add sentence state: the essence of the current statement
- Synthesis of conversational data
Database Abstraction (single command)

DB Queries factored into

Relational Operators
Implemented by
DB writers

Domain Knowledge: Schemas
Defined by
App developers

σ, Π, U, −, ×

Natural language
constructs

Natural language query factored into

<table>
<thead>
<tr>
<th>Restaurant</th>
<th>User</th>
<th>City</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Email</td>
<td>Geo</td>
<td>Name</td>
</tr>
<tr>
<td>Price</td>
<td>Age</td>
<td>Country</td>
<td>UPC</td>
</tr>
<tr>
<td>Cuisine</td>
<td></td>
<td></td>
<td>Price</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Implemented by DB writers
Defined by App developers

Domain Knowledge: Schemas

Natural language constructs

Natural language query factored into

Domain Knowledge: Schemas

Natural language constructs

Natural language query factored into

Domain Knowledge: Schemas

Natural language constructs

Natural language query factored into

Domain Knowledge: Schemas

Natural language constructs

Natural language query factored into
Database Abstraction (Conversational)

DB Queries factored into

Relational Operators
Implemented by DB writers

Domain Knowledge: Schemas
Defined by App developers

σ, π, U, −, ×

Formal Context

Incomplete natural language constructs

Natural language query factored into

Restaurant
- Name
- Price
- Cuisine

City
- Name
- Geo
- Country

User
- Name
- Email
- Age

Product
- Name
- UPC
- Price
## Introduce: Sentence State (SS)

CSP: contextual semantic parser; Apply: algorithmic;

<table>
<thead>
<tr>
<th>Context (c)</th>
<th>English</th>
<th>ThingTalk</th>
</tr>
</thead>
<tbody>
<tr>
<td>I’d like a French restaurant in Menlo Park</td>
<td>@com.yelp.restaurant() filter cuisine == ‘French’ &amp;&amp; location == Location(‘Menlo Park’)</td>
<td></td>
</tr>
<tr>
<td>Sentence state $s = CSP (u)$</td>
<td>What is its address?</td>
<td>[address] of @com.yelp.restaurant()</td>
</tr>
<tr>
<td>Dialogue state $d = Apply (s, c)$</td>
<td>What is the address of a French restaurant in Menlo Park</td>
<td>[address] of @com.yelp.restaurant() filter cuisine == ‘French’ &amp;&amp; location == Location(‘Menlo Park’)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Context (c)</th>
<th>English</th>
<th>ThingTalk</th>
</tr>
</thead>
<tbody>
<tr>
<td>I want a 4-star restaurant in Palo Alto.</td>
<td>@com.yelp.restaurant() filter rating == ‘4’ &amp;&amp; location == Location(‘Palo Alto’)</td>
<td></td>
</tr>
<tr>
<td>Sentence state $s = CSP (u)$</td>
<td>What is its address?</td>
<td>[address] of @com.yelp.restaurant()</td>
</tr>
<tr>
<td>Dialogue state $d = Apply (s, c)$</td>
<td>What is the address of a 4-star restaurant in Palo Alto.</td>
<td>[address] of @com.yelp.restaurant() filter rating == ‘4’ &amp;&amp; location == Location(‘Palo Alto’)</td>
</tr>
</tbody>
</table>

The same English utterance and domain $\rightarrow$ same sentence state
Design Principles of Sentence State

- No new syntax: It’s a ThingTalk statement
  Guarantees representation completeness

- Sentence-State: \( s = \text{CSP} \ (u, c) \)
  The smallest complete ThingTalk statement
  (borrowing from context \( c \))
  that includes all information in the new utterance \( u \).

- Dialogue-State \( d: = \text{apply} \ (s, c) \)
  Include as much information from context \( c \)
  as possible without generating a conflict with \( s \), rendering a null result.
Apply Algorithm

\[ s: \text{Expression with input parameters and a result} \]

- Add constraints from \( c \) on the input parameters that do not conflict

\[
\begin{align*}
\text{c:} & \text{ @restaurant (ratings = 3), cuisine = “Mexican”} \\
\text{How about 4-star ones?} \\
\text{s:} & \text{ @restaurant (ratings = 4)} \\
\text{Apply (s, c) =} \\
\end{align*}
\]

- Add computation from \( c \) to the result

\[
\begin{align*}
\text{Who is the tallest U. S. president?} \\
\text{c:} & \text{ sort height desc of @people(), filter profession = `US President'[1]} \\
\text{How about the Ukrainian presidents?} \\
\text{s:} & \text{ @people() filter profession = ‘Ukraine President’} \\
\text{Apply (s, c) =} \\
\end{align*}
\]
## Co-References and Ellipses

<table>
<thead>
<tr>
<th>Co-reference</th>
<th>Last Turn and Context</th>
<th>User Utterance and Sentence State</th>
<th>Dialogue State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Co-reference</td>
<td>What is the phone number of Left Bank? [phonenumber] of @restaurant() filter id == ‘Left Bank’</td>
<td>What is its address? (Projection) [address] of @restaurant()</td>
<td>What is the address of Left Bank? [address] of @restaurant() filter cuisine = ”French”</td>
</tr>
<tr>
<td>Ellipsis</td>
<td>What is the phone number of Left Bank? [phonenumber] of @restaurant() filter id == ‘Left Bank’</td>
<td>How about the address? (Replace projection) [address] of @restaurant()</td>
<td>What is the address of Left Bank? [address] of @restaurant() filter id == ‘Left Bank’</td>
</tr>
<tr>
<td>Co-reference</td>
<td>Suggest a French restaurant @restaurant() filter cuisine = ”French”</td>
<td>I want one in Palo Alto (Add filter) @restaurant(), filter geo = ‘Palo Alto’</td>
<td>I want a French restaurant in Palo Alto @restaurant() filter geo = ‘Palo Alto’. cuisine = ”French”</td>
</tr>
<tr>
<td>Ellipsis</td>
<td>Who are the U. S. president? @people(), filter profession = ‘US President’</td>
<td>Who is the shortest? (Add sort) sort height asc of @people()[1]</td>
<td>Who is the shortest U. S. President? sort height asc of @people(), filter profession = ‘US President’[1]</td>
</tr>
<tr>
<td>Ellipsis</td>
<td>Who is the tallest U. S. president? sort height desc of @people(), filter profession = ‘US President’[1]</td>
<td>How about the shortest? (Change order of sort) sort height asc of @people()[1]</td>
<td>Who is the shortest U. S. President? sort height asc of @people(), filter profession = ‘US President’[1]</td>
</tr>
<tr>
<td>Ellipsis</td>
<td>Who is the tallest U. S. president? sort height desc of @people(), filter profession = ‘US President’[1]</td>
<td>How about the Ukranian presidents? (Change filter) @people() filter profession = ‘Ukrainian President’</td>
<td>Who is the tallest Ukranian president? sort height desc of @people() filter profession = ‘Ukrainian President’</td>
</tr>
</tbody>
</table>

Co-references and ellipses have different natural language constructs. Ellipses can be used for any part of the sentence!
Ellipses often used for error corrections as well

- Different natural language forms

<table>
<thead>
<tr>
<th>Last Turn and Context</th>
<th>User Utterance and Sentence State</th>
<th>Dialogue State</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ellipses</strong> - useful for CSP errors</td>
<td>Who is the tallest Ukrainian president? <strong>ERROR</strong>: sort height desc of @people(), filter profession = ‘US President’[1]</td>
<td>Who is the tallest Ukrainian president? sort height desc of @people() filter profession = ‘Ukrainian President’</td>
</tr>
<tr>
<td></td>
<td>I mean Ukraine president @people() filter profession = ‘Ukrainian President’</td>
<td></td>
</tr>
</tbody>
</table>
Key Concepts

• Conversational semantics:
  Add sentence state: the essence of the current statement
• Synthesis of conversational data
## Co-References

A template for each parameter in a query template

<table>
<thead>
<tr>
<th>Operation</th>
<th>Context from Last Turn</th>
<th>New User Statement</th>
<th>Sentence State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projection</td>
<td>@restaurant() filter cuisine = &quot;chinese&quot;</td>
<td>Can you tell me its address?</td>
<td>[address] of @restaurant()</td>
</tr>
<tr>
<td>Aggregation</td>
<td>@restaurant() filter cuisine =~ &quot;chinese&quot;</td>
<td>how many such restaurants are there?</td>
<td>count(@restaurant())</td>
</tr>
<tr>
<td>Verification</td>
<td>@restaurant() filter id == ‘Left Bank’</td>
<td>Does it have a 4 star rating?</td>
<td>[ratings ==4] of @restaurant()</td>
</tr>
<tr>
<td>Computation</td>
<td>@restaurant() filter id == ‘Left Bank’</td>
<td>How far is it?</td>
<td>[distance, $location.current_location] of @restaurant()</td>
</tr>
<tr>
<td>Join</td>
<td>@restaurant() filter id == ‘Left Bank’</td>
<td>How many reviews does it have?</td>
<td>count(@restaurant() join @reviews())</td>
</tr>
<tr>
<td>Action</td>
<td>@restaurant() filter id == ‘Left Bank’</td>
<td>Please book it</td>
<td>@book_restaurant(id = ‘Left Bank’)</td>
</tr>
</tbody>
</table>
Ellipses
A template for each possible component to substitute

<table>
<thead>
<tr>
<th>Operation</th>
<th>Context from Last Turn</th>
<th>New User Statement</th>
<th>Sentence State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct a domain</td>
<td>@play-movie(), title=&quot;Bohemian Rhapsody&quot;</td>
<td>I want to play the song, not the movie.</td>
<td>@play-song()</td>
</tr>
<tr>
<td>Add filter</td>
<td>@restaurant() filter cuisine = &quot;chinese&quot;</td>
<td>I only want 4 star rated ones</td>
<td>@restaurant() filter rating &gt;= 4</td>
</tr>
<tr>
<td>Add filter to subquery</td>
<td>@review() filter id == any(@restaurant())</td>
<td>Only reviews from 4 star restaurants</td>
<td>@review() filter id == any(@restaurant() filter ratings == 4)</td>
</tr>
<tr>
<td>Add filter to projection</td>
<td>[cuisine] of @restaurant()</td>
<td>Only those with 4 star rating</td>
<td>@restaurant() filter rating == 4</td>
</tr>
<tr>
<td>Remove a filter</td>
<td>@restaurant(), cuisine = &quot;italian&quot;</td>
<td>it does not have to be Italian.</td>
<td>@restaurant(), true(cuisine)</td>
</tr>
<tr>
<td>Negate a filter</td>
<td>@restaurant(), cuisine = &quot;italian&quot;</td>
<td>I don't want Italian</td>
<td>@restaurant(), cuisine != &quot;italian&quot;</td>
</tr>
<tr>
<td>Relax a filter</td>
<td>@restaurant(), cuisine = &quot;pizza&quot;</td>
<td>Italian restaurants also work</td>
<td>$or @restaurant(), cuisine = &quot;italian&quot;</td>
</tr>
<tr>
<td>Correct a param value</td>
<td>@restaurant(), cuisine = &quot;japanese&quot;</td>
<td>I would rather have Italian food.</td>
<td>@restaurant(), cuisine = &quot;italian&quot;</td>
</tr>
<tr>
<td>Correct the parameter</td>
<td>@song(), album_title = &quot;taylor swift&quot;</td>
<td>The artist name is Taylor Swift, not the album title.</td>
<td>@song(), album_title != &quot;taylor swift&quot; &amp;&amp; artist_name=&quot;taylor swift&quot;</td>
</tr>
<tr>
<td>Correct filter</td>
<td>@train(), departure_time &gt;= 9:00</td>
<td>I mean the train need to depart before 9 am</td>
<td>@train(), departure_time &lt;= 9:00</td>
</tr>
<tr>
<td>Correct a projection</td>
<td>[address] of @restaurant()</td>
<td>I wanted the phone number</td>
<td>[phone_number] of @restaurant()</td>
</tr>
<tr>
<td>Reverse sort</td>
<td>sort(rating desc of @restaurant())</td>
<td>I want the restaurants with the lowest rating</td>
<td>sort(rating asc of @restaurant())[1]</td>
</tr>
<tr>
<td>Correct index</td>
<td>sort(rating desc of @restaurant())[1]</td>
<td>I want to see the 2nd highest rated</td>
<td>sort(rating desc of @restaurant())[2]</td>
</tr>
</tbody>
</table>
Completeness of Synthesis (wrt co-references and ellipses)

Add:

- A primitive template for each query operation, substitute a parameter with a co-reference to the context
- A primitive template for each component of a query, just say the component, using the context as elided content
- Compounding primitive templates covers all combinations
Rationale of the Design

• The sentence state (SS) is shorter, matching human speech
  • The dialogue state gets longer with the conversation
  • CSP, contextual semantic semantic parser, predicts only what is mentioned
  • Easier to annotate, easier to learn?
• The agent can tell the focus of the new statement
  • Can emphasize the new addition in reporting results and in confirmations
• NOTE: The sentence states are relatively simple, but the dialogue states after “apply” are large
Quiz

- Potential downside: does it need a neural network to “apply”?
Generic State Machine

Synthesis: All the states
Execution: GSM minus Agent initiative

Run-time system:
Interleaves GSM with program policy
Synthesize Conversations

• Simulate the state machine
  • Synthesize user statement
  • Execute: Generate arbitrary results
  • Synthesize agent response
• A small state machine
  • but the context from previous state limits what can be synthesized as the next step
• Note: after “apply” the dialogue state has a lot of variety
## Quiz: Comparison with Genie’s MultiWOZ

<table>
<thead>
<tr>
<th></th>
<th>Genie’s MultiWOZ</th>
<th>GenieScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target representation for CSP (contextual semantic parser)</td>
<td>Dialogue State in ThingTalk</td>
<td></td>
</tr>
<tr>
<td>Ambiguity in utterance</td>
<td>Resolved by neural network</td>
<td></td>
</tr>
<tr>
<td>Conversational templates</td>
<td>Extract from MultiWOZ dialogues (very few changes of mind)</td>
<td></td>
</tr>
<tr>
<td>Agent policy</td>
<td>Hard coded: same as synthesis state machine - Very hard to write/modify</td>
<td></td>
</tr>
<tr>
<td>Ease of updating agent policy</td>
<td>Need to resynthesize and train with changes to agent policy</td>
<td></td>
</tr>
<tr>
<td>Generality</td>
<td>Transactional, sensitive to WOZ instructions</td>
<td></td>
</tr>
</tbody>
</table>
Summary

*Our Motto: “To understand every relevant statement”.*

- A conversational agent
  - Factored into domain schemas and agent policy

- Can train contextual semantic parser with data synthesized from only the schemas, for *any* agent policy

- Write the generic state machine (GSM) ONCE by EXPERTS
  - To exercise all the conversational transitions
    - with user & agent initiatives
  - GSM used in synthesis
  - GSM without agent initiatives used to handle user initiatives