A Formal Language & Data Engineering Approach to Multi-Turn Dialogues

CS294S/W FALL 2020, LECTURE 4

Giovanni Campagna
(Work done in collaboration with Sina Semnani, Ryan Kearns, Lucas Sato, Agata Foryciarz, Mehrad Moradshahi, Silei Xu & others in OVAL under supervision of Prof. Monica Lam)
(Genie Logo by SK Lim)
The “State of The Art” in VA Tools: Slot Elicitation

U: I want to make a restaurant reservation for Terun
A: For how many people?
   U: 3 people
A: At what time?
   U: 7pm
A: Ok, I’m going to make a restaurant reservation for 3 people at Terun at 7pm.

• One slot / parameter at a time
• Fixed, developer-provided questions per parameter
• No error path: if the user doesn’t answer the question, tough luck
Real World Dialogues [Budzianowski et al. 2018]

U: Could you give me information about a restaurant called Panahar?

A: Absolutely. Panahar is a more expensive Indian restaurant. It is located at 8 Norfolk Street city centre. Their postal code is cb12lf. Their phone number is 01223355012.

U: Could you book a table for me there on Friday?

A: Yes, I can, for how many people and what time on Friday?

U: There will be 4 of us and we would like to eat at 11:00.

A: Booking was unfortunately unsuccessful. Can we try another day or time slot?

U: Sure, how about 10:00?
Transaction (Slot-Filling) Dialogues

- A subset of *task-oriented dialogues* (participants trying to “do things”)
- **User** introduces the *transaction* & drives the conversation
- **Agent** provides answers & suggestions + elicits info to complete actions

- Superset of interactive search & Q&A (informational)
- Covers all dialogues that execute user-driven actions
  - Purchases
  - Reservations
  - Tickets
  - Simple customer support: changing/cancelling orders, paying bills, scheduling repairs/returns, etc.
Challenges of Transaction Dialogues

- Carrying over of contextual information
- Multiple *slots* per turn
- Error correction and recovery

- Long studied field
- First notable work: **Dialogue State Tracking Challenge** (2011)

Can we solve transaction dialogues *once and for all?*
The Practical Modular Approach To Dialogues

User Utterance → NLU → Intent & Slots → Amorphous Blob of Domain-Specific Code → Backend → API calls → Agent Reply

Training Data → NLU

Stanford University
The Academic Modular Approach To Dialogues

Complete Dialogue History

Neural State Tracker

Intent & Slots

Training Data

Amorphous Blob of Domain-Specific Code

API calls

Backend

Agent Reply
State of the Art: Manually Annotated Conversations

- Dialogues are vast, complex and very varied \(\rightarrow\) **need a lot of data to train**

- Alexa: 10k employees, millions of manually annotated sentences

- *MultiWOZ* dataset [Budzianowski et al.]:
  - \~10k hand annotated dialogues in 5 domains
  - \~100k turns in total

- State of the art: about 55% *joint accuracy*

- About 30% of the errors are misannotations [Zhou and Small]
Our Approach

- Formal Dialogue State
- User Utterance
- Neural NLU & State Tracking
  - Executable ThingTalk Code
  - ThingTalk Runtime
    - API calls
    - Backend
      - Synthesis & Automatic Paraphrasing
      - New Dialogue State
  - Domain-Independent Dialogue State Machine
    - Results
    - Neural Language Generation
      - Agent Reply

Campagna, Semnani, Kearns, Koba, Sato, Lam, ArXiv
Key Insights

• Formal, *executable* representation for dialogue states
• State as input & output of neural network
• Domain-independent dialogue state machine to specify agent behavior
• Synthesis approach to training data
Lecture Outline

1. The last state machine for transaction dialogues
2. Combining language understanding & state tracking
3. How to specify a dialogue agent
4. From specification to a complete agent
5. Experimental results (and how to push them)
But First, A Bit of Terminology

- **Utterance**: each phrase said by the user or agent
- **Turn**: a pair of (agent, user) interactions
- **Dialogue**: well-formed sequence of turns
- **History**: sequence of all turns up to the current point
- **State**: formal representation of the dialogue, up to a certain point
  - *User state*: right after the user speaks
  - *Result state*: right after execution
  - *Agent state*: right after the agent speaks
- **Abstract State**: family of states, as defined by the dialogue state machine
Lecture Outline

1. The last state machine for transaction dialogues
2. Combining language understanding & state tracking
3. How to specify a dialogue agent
4. From specification to a complete agent
5. Experimental results (and how to push them)
U: I am looking for a restaurant. The restaurant should serve world food and should be in the centre.
A: Unfortunately, I could not find a restaurant that serves that kind of food in that area.
U: What expensive restaurants are there?
A: The Cambridge Chop House is expensive and it’s British food. Is that okay?
U: No. Try looking for a restaurant that serves Korean food.
A: I found the Little Seoul restaurant. Would you like me to make a reservation?
U: Yes, a table for 2.
A: I would be happy to reserve for you, what day and time are you looking for?
U: Thank you, I need a reservation for 2 people at 13:15 on Monday.
A: gg3dcufd is your confirmation code, may I help you with anything else?
U: No that was everything. Thanks
Beyond Restaurant Reservations

U: I would like to watch a movie.
A: What movie would you like? Do you like comedy?
U: I prefer action movies
A: How about Spiderman?
U: Sure I like that
A: Opening Netflix…

U: I need to buy new shoes.
A: Are you looking for sneakers or leather shoes?
U: Probably a new pair of Nikes
A: How about the new Air Jordans? They are $299 on nike.com

U: They look cool! I’ll take them.
A: Would you like them shipped at your usual address?
U: Yes please
A: Alright, payment is processing. With 2-day standard shipping, they will arrive Tuesday.

U: Hey Genie, order me a grande latte
A: Where would you like to pick it up?
U: Stanford & El Camino
A: Alright, ordering 1 grande latte from Starbucks. That will be $3.65 + tax. It will be ready in 10 minutes.
A State Machine For Transaction Dialogues

Campagna, Foryciarz, Moradshahi, Lam, ACL 2020
Transaction Dialogues in Practice

• All dialogues look the same - so why do people pay for skill developers?

• In Alexa (& DialogFlow, Rasa, Watson, LUIS...): **dialogue tree**
  • Replace *dialogue act* (domain indep.) with *intent* (domain dep.)
  • Write lots of sentences for each intent
  • Hard-code the agent behavior for each intent

• Conflates:
  • *Semantics* (what it means)
  • *Policy* (what to do with it)
  • *Execution* (how to do it)
Concrete State Representation

- Previously: domain + abstract dialogue act + slots
- Slot: “latest mention of an entity from the user”
- Ill-defined

U: I’m looking for an Italian restaurant.
    [ food = “Italian” ]

A: I found Terun. Would you like a reservation?

U: Yes please!
    [ food = “Italian”, name = ??? ]

- Contrast: formal **ThingTalk executable semantics**
  - Straightforward denotational semantics through relational algebra
  - It either gives you the answer, or it doesn’t!
The Restaurant Example

I’m looking for an Italian restaurant

NLU (contextual semantic parsing)

$dialogue execute:
@Restaurant(), food == “Italian”

ThingTalk Runtime

{ name = “Terun”, price_range = moderate, geo = “California Ave”, … }

State Machine & Language Generation

I have found Terun. Would you like a reservation?
The Language of Dialogue States (User Side)

$dialogue @org.thingpedia.dialogue.transaction.execute;

now => @com.yelp.Restaurant(), food == “italian” => notify
# [results=[
    { name = “Terun”, price_range = moderate, ... },
    ...
];

now => @com.yelp.Restaurant(), food == “italian” &&
    price_range == enum(cheap) => notify;

now => @com.yelp.make_reservation(restaurant=$?, ...);
The Language of Dialogue States (Agent Side)

$dialogue @org.thingpedia.dialogue.transaction.sys_rec_one ;

now => @com.yelp.Restaurant(), food == “italian” => notify
#[results=[
    { name = “Terun”, price_range = moderate, … },
    …
];

now => @com.yelp.make_reservation(restaurant=$?, …);

now => @com.yelp.make_reservation(restaurant=“Terun”, …)
#[confirm=enum(proposed)];
How do we know that this representation is sufficient?
User & Agent Dialogue Act Labels

- greet
- execute
- learn_more
- ask_recommend
- cancel
- end
- sys_greet
- sys_search_question(param)
- sys_generic_search_question
- sys_slot_fill(param)
- sys_recommend_one
- sys_recommend_two
- sys_recommend_three
- sys_propose_refined_query
- sys_learn_more_what
- sys_empty_search_question(param)
- sys_empty_search
- sys_action_success
- sys_action_error
- sys_anything_else
- sys_goodbye
Lecture Outline

1. The last state machine for transaction dialogues
2. Combining language understanding & state tracking
3. How to specify a dialogue agent
4. From specification to a complete agent
5. Experimental results (and how to push them)
You’ve Seen This Picture Before

I’m looking for an Italian restaurant

$dialogue execute: @Restaurant(), food == “Italian”
I'm looking for an Italian restaurant

```plaintext
$dialogue execute:
@Restaurant(),
food == "Italian"
```
Adding The Dialogue State

Previous ThingTalk

$dialogue
sys_search_question(food):
  @Restaurant(),
  price_range == moderate

Natural Language

I’m looking for an Italian restaurant

Neural Semantic Parser

Next ThingTalk

$dialogue execute:
  @Restaurant(),
  food == “Italian” &&
  price_range == moderate
The Neural Model (BERT-LSTM)
What are the advantages & disadvantages of the contextual NLU model vs. training with the full dialogue history?
Pros & Cons of Contextual NLU

• Positive: **Minimal information to disambiguate input**
  • Formal state removes “extra” information
  • Controllable amount of history to show to network
  • Reduces needed training data

• Positive: **Formal guarantees of agent behavior**
  • Can prevent “bad states” with tools of formal verification

• Negative: **Difficult to capture uncertainty**
  • Hard sample of one dialogue state as output of the network
  • Inherent ambiguity must be expressed in formalism
  • Statistical ambiguity is lost
Lecture Outline

1. The last state machine for transaction dialogues
2. Combining language understanding & state tracking
3. **How to specify a dialogue agent**
4. From specification to a complete agent
5. Experimental results (and how to push them)
The Dialogue Agent, Again

Current Dialogue State → User Utterance → Neural NLU & State Tracking
→ Executable State → ThingTalk Runtime
→ API calls → Backend
→ Result State → Agent Policy
→ New Dialogue State → Neural Language Generation
→ Agent Reply
Specifying The Dialogue Agent

- Domain-independent state machine
- Domain dependent schema + APIs + annotations
- Agent Policy
- Synthesis (Simulation)
- Contextual NLU
- Neural NLG

Stanford University
Reminder: The Transaction Dialogue State Machine
Zooming In…

- **SearchRequest**
  - LargeResult
    - SlotFillQuestion
  - SmallResult
    - ProposeN
  - SingleResult
    - ProposeOne
  - EmptyResult
    - EmptySearchError
  - AskAction
  - SearchRefine
  - InfoQuestion
    - IncompleteAction
      - SlotFillQuestion
    - SuccessfulAction
      - ActionResult
    - FailedAction
      - ActionError

User state
Agent state
Result state
Specifying Dialogue State Machines

**Result state**

**State function**

**Abstract result state**

**User template**

**Agent template**

**ProposeN**
I have ___ and ___. Both are ___.

**ProposeOne**
I have ___. It is a ___ ___ that ___.

**AskAction**
I like that. Can you help me ___ it?

**InfoQuestion**
What is the ___ of ___?

**SearchRefine**
I don’t like ___. Do you have something ___?

**IncompleteAction**

**SlotFillQuestion**
___ what ___ would you like it?
Specifying Dialogue State Machines

**State function**: map dialogue state (ThingTalk code + result) to *abstract state*

**Transition templates**: triple of *(abstract state, agent act, user act)* + validation code

**Agent templates**: express agent act as sentence + target state

**User templates**: express user act as sentence + target state
The Transaction Dialogue State Machine

*State function*: function(state) { if (state.current.result !== null) return StateResult; … }

*Transition templates:*
StateResult: ProposeOne → AskAction
StateResult: ProposeOne → SearchRefine
StateResult: ProposeN → InfoQuestion

*Agent templates*
ProposeOne = [“I have” | “I found”] Name [ “Would you like to” CorefAction “?” | InfoPhrase ]
ProposeN = [“I have” | “I found”] Name “and” Name “Both are”
  [ AdjectiveSlot | PassiveVerbSlot ]

*User templates*
AskAction = [“I like that” | “Sounds good”] “Can you help me” CorefAction “?”
InfoQuestion = [“Can you tell me the”] Param “of” Name “?”
Specifying A Skill

**Database Schema**

Restaurant
- name: “Curry Prince”, “Pizza Express”, …
- area: “north”, “south”, “east”, …
- price: “cheap”, “moderate”, “expensive”
- food: “Italian”, “Chinese”, “Indian”, …
- address
- phone
- postcode

**API Actions**

Make Reservation
- book_people: 1 ... 7
- book_day: “Monday”, “Tuesday”, …
- book_time: 7:00, 8:00, …
- confirmation_number

Stanford University
Referring To Slots In Natural Language

**DOMAIN:**
[ “restaurant”, “food place” ]

**Slot “food”:**
- **NOUN:** [ “food”, “cuisine” ]
- **ADJECTIVE:** [ “$value” ]
- **HAS-NP:** [ “$value food”, “$value cuisine” ]
- **VERB:** [ “serves $value food” ]

what <NOUN> does this <DOMAIN> have?  
what food does this restaurant have?  
what <NOUN> would you like?  
what cuisine would you like?

Xu, Campagna, Li, Lam, CIKM 2020
Referring To Slots In Natural Language

**DOMIAN:**
[ “restaurant”, “food place” ]

**Slot “food”**:  
- **NOUN**: [ “food”, “cuisine” ]
- **ADJECTIVE**: [ “$value” ]
- **HAS-NP**: [ “$value food”, “$value cuisine” ]
- **VERB**: [ “serves $value food” ]

```
<ADJECTIVE> <DOMAIN>  
<DOMAIN> that <VERB>  
<DOMAIN> with <HAS-NP>  
<DOMAIN> that has <HAS-NP>  

Italian restaurant, Italian food place  
restaurant that serves Italian food  
restaurant with Italian food  
restaurant that has Italian cuisine
```
# Specifying The Domain For A Transaction Dialogue

## Query (Schema)

**Restaurant** [ “restaurant”, “food place” ]
- *id*: Entity(Restaurant)
- *geo*: Location
  - [ “address”, “in #”, “near #”, “around #” ]
- *price*: Enum(cheap, moderate, expensive)
  - [ “# -ly priced “, “#” ]
- *rating*: Number [min=1, max=5]
  - [ “rated #” ]
- *cuisines*: Array(Entity(Cuisine))
  - [ “# food”, “serves # food“ ]
- …

## Actions

**MakeReservation**
- [ “reserve #”, “book #” ]
- restaurant*: Entity(Restaurant)
- *book_people*: Number [min=1]
  - [ “for #”, “for # people” ]
- *book_day*: Date [ “for #” ]
- confirmation_number*: String
  - [ “confirmation number” ]
In ThingTalk Syntax

class @com.yelp {
    query restaurant(out id: Entity(com.yelp:restaurant),
        out food: String
        #[canonical={
            base=["cuisine", "food"],
            property=["# cuisine", "# food"]
        }]
        #[prompt=["what would you like to eat"]],
    ...);

    action make_reservation(in req restaurant: Entity(com.yelp:restaurant),
        in req book_day : Date
        #_[canonical={
            base=["day", "date"],
            preposition=["for ", "on "]
        }], ...
    }
}
The General Form of Transactional Domains

- One query + one or more actions
  - Query: the subject of the discourse
  - Actions: what you can do with it

- Query with `id` parameter of Entity type
- Query must have the same name as the Entity type
- Other parameters are the properties of the object
  - Some are searchable (`#([filterable=true]`)  

- Actions have one parameter with the same type as the query ID
  - Ties query and actions together
Concretely Doing Things: The Backend

```javascript
const Tp = require('thingpedia');

module.exports = class extends Tp.BaseDevice {
    get_restaurant() {
        // return the content of the database
        // to perform the query
        return ...;
    }

    do_make_reservation({ restaurant, book_day, ... }) {
        // call the API to make it happen!
    }
}
```
Putting It All Together

SmallResult
Restaurant, price == moderate && geo == “Palo Alto”
{ id = “Terun”, price = moderate, cuisines = [“pizza”], … }
{ id = “Coconuts”, price = moderate, cuisines = [“caribbean”], … }

ProposeOne
I have Terun. It is a moderately priced restaurant that serves pizza.

ProposeN
I have Terun and Coconuts. Both are moderately priced.

AskAction
I like that. Can you help me book it?

SearchRefine
I don’t like pizza. Do you have something Caribbean?

IncompleteAction

InfoQuestion
What is the address of Terun?

SlotFillQuestion
For what date would you like it?
What are examples of dialogues that cannot be expressed as transactions?
Designing State Machines (For Your Projects)

1. State with few manually written dialogues
2. Annotate fully
3. Collapse into abstract states, agent acts and user acts (on paper)
4. Draw the state machine (on paper)
5. Separate domain-independent & domain-dependent parts
6. Code the state machine in Genie
7. Generate a sample dataset
8. Observe, refine, iterate
Designing State Machines (For Your Projects)

- Write state function
  - Given concrete state of the dialogue, capture abstract state
  - Ideally, should work for all possible states
- Write agent templates
  - Both sentence & formal state
- Given agent sentence, write as many user templates as meaningful
  - Templates will depend on the specifics of the agent sentence!
  - Capture pragmatics: why is the user saying something?
  - Collapse or distinguish meaning accordingly
  - OK to write a little nonsense
- Helpful to look at existing human-human conversations
Tips & Tricks for Dialogue Design

• Be *transparent* to your users
  • It’s an incomplete information game
  • Reveal as much as possible of what you know
• Let the user *be in the charge*
  • The customer is always right
Lecture Outline

1. The last state machine for transaction dialogues
2. Combining language understanding & state tracking
3. How to specify a dialogue agent
4. From specification to a complete agent
5. Experimental results (and how to push them)
Reminder: Specifying The Dialogue Agent

Domain-independent state machine

Domain dependent schema + APIs + annotations

Agent Policy

Synthesis (Simulation)

Contextual NLU

Neural NLG
Training the NLU

• Step 1: synthesize as many dialogues as possible
  • Enumerate many possible next states
  • Simulate execution
  • Iterate until enough dialogues generated

• Step 2: extract one *training sample* per turn
  • Old state + user utterance + new state
Constructing the Agent Policy

- Dialogue state from execution with real policy
  - Sample *transition*
  - Sample *agent template*
  - Output: *state after agent speaks + synthetic utterance*

- (Future) neural NLG to improve synthetic utterance
How do you compare the Genie way to write the agent vs. dialogue trees?
Lecture Outline

1. The last state machine for transaction dialogues
2. Combining language understanding & state tracking
3. How to specify a dialogue agent
4. From specification to a complete agent
5. Experimental results (and how to push them)
Experiment 1: Quality of Synthesis (Domain Transfer)

- **MultiWOZ 2.1 benchmark** [Budzianowski et al. 2018, Eric et al. 2019]
  - ~10k hand annotated dialogues in 5 domains
  - Annotation: Domain + Slot: not mentioned / value / don’t care

- Training set:
  - 4 annotated domains
  - Synthesized data for the 5th domain
  - *Domain adaptation*: convert dialogues of similar domains

- Test set with new domain (existing test data)

- Evaluation metric: *Joint Accuracy* (all slots correct given partial dialogue)

- Models:
  - TRADE [Wu et al. 2019] – fully trained RNN + ptr-gen
  - SUMBT [Lee et al. 2019] – pretrained BERT + ontology match
Zero-Shot Transfer Learning (TRADE)

Joint Accuracy on MultiWOZ 2.1 Test Set

- Attraction
- Hotel
- Restaurant
- Taxi
- Train

Full Dataset
- Zero-shot Approach by Wu et al.
- Our Approach
- Campagna, Foryciarz, Moradshahi, Lam, ACL 2020
Zero-Shot Transfer Learning

Joint Accuracy on MultiWOZ 2.1 Test Set

- Attraction
- Hotel
- Restaurant
- Taxi
- Train

- Full Dataset (TRADE)
- Full Dataset (SUMBT)
- Zero-shot Approach by Wu et al.
- Our Approach (TRADE)

Campagna, Foryciarz, Moradshahi, Lam, ACL 2020
Zero-Shot Transfer Learning

Joint Accuracy on MultiWOZ 2.1 Test Set

Attraction
Hotel
Restaurant
Taxi
Train

- Full Dataset (TRADE)
- Full Dataset (SUMBT)
- Zero-shot Approach by Wu et al.
- Our Approach (TRADE)
- Our Approach (SUMBT)

Campagna, Foryciarz, Moradshahi, Lam, ACL 2020
Zero- and Few-Shot Transfer Learning

Joint Accuracy on MultiWOZ 2.1 Test Set

Campagna, Foryciarz, Moradshahi, Lam, ACL 2020
Experiment 2: Performance of Contextual Semantic Parsing

• Experimental setting: **MultiWOZ 2.1 dev and test, reannotated**
  • Add dialogue states for agent, correct annotations

• Training set: synthesized + ~1k turns few-shot (2% of original training)

• Evaluation metric: *Turn-By-Turn Accuracy* (correct next state given current)
  • Measures the ability to continue the dialogue

• Model: BERT-LSTM
Results

![Bar chart showing validation and test results for Exact Match and Slots Only.

- Validation:
  - Exact Match: 70%
  - Slots Only: 80%

- Test:
  - Exact Match: 60%
  - Slots Only: 70%

Campagna, Semnani, Kearns, Koba, Sato, Lam, ArXiv
Contextual Semantic Parsing vs Dialogue State Tracking

TRADE (Wu et al. 2019)
SUMBT (Lee et al. 2019)
DSTQA (Zhou and Small, 2019)
DST-Picklist (Zhang et al., 2019a)
SST (Chen et al., 2020)
TripPy (Heck et al., 2020)
SimpleTOD (Hosseini-Asl et al., 2020)
Contextual Semantic Parsing

Campagna, Semnani, Kearns, Koba Sato, Lam, ArXiv
Ablation Study

Turn-by-turn accuracy on validation set

- **Exact Match**
  - Synthesized Only
  - Few-Shot Only
  - Synthesized + Few-Shot

- **Slots Only**
  - Synthesized Only
  - Few-Shot Only
  - Synthesized + Few-Shot

Campagna, Semnani, Kearns, Koba Sato, Lam, ArXiv
Takeaways

• Synthesized data is a **cheap** and **effective** way to build training sets
  • Easy to obtain with a general *state machine* + few domain templates
  • **+21% accuracy** over zero-shot SOTA in DST

• Contextual semantic parsing is a better way to build a dialogue agent
  • Needs more precise annotations than DST (agent state)
  • But easy to obtain with synthesized and few shot
  • **+14% accuracy** turn by turn compared to DST
Recap: Dialogues The Genie Way™

• Formal, executable state representation

• Contextual NLU for state tracking: (current state, user input) → new state

• Data engineering of both agent and user at the same time
  • Domain-independent state machine factored out
  • A lot more state transitions possible
  • Domain-specific in the form annotations (generated in the future?)
How hard is it to write new dialogue state machines?