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

CS294S/W 2020, LECTURE 3

Giovanni Campagna
(Work done in collaboration with Agata Foryciarz, Mehrad Moradshahi,
Sina Semnani, 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

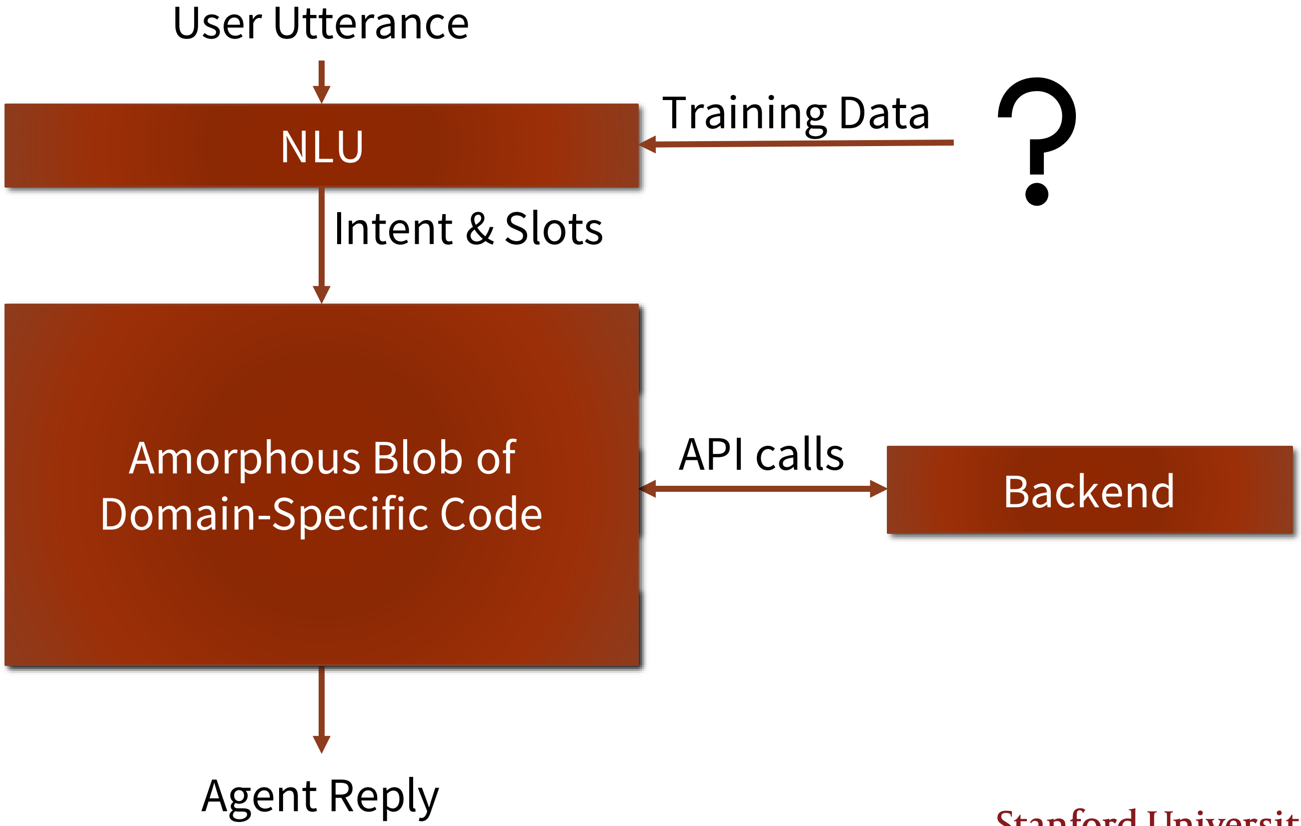
- A subset of *task-oriented dialogues* (participants trying to “do things”)
- **User** introduces the *transaction* & drives the conversation
- **Agent** provides answers & suggestions + elicits info to complete actions

- 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

Training Data

?

Intent & Slots

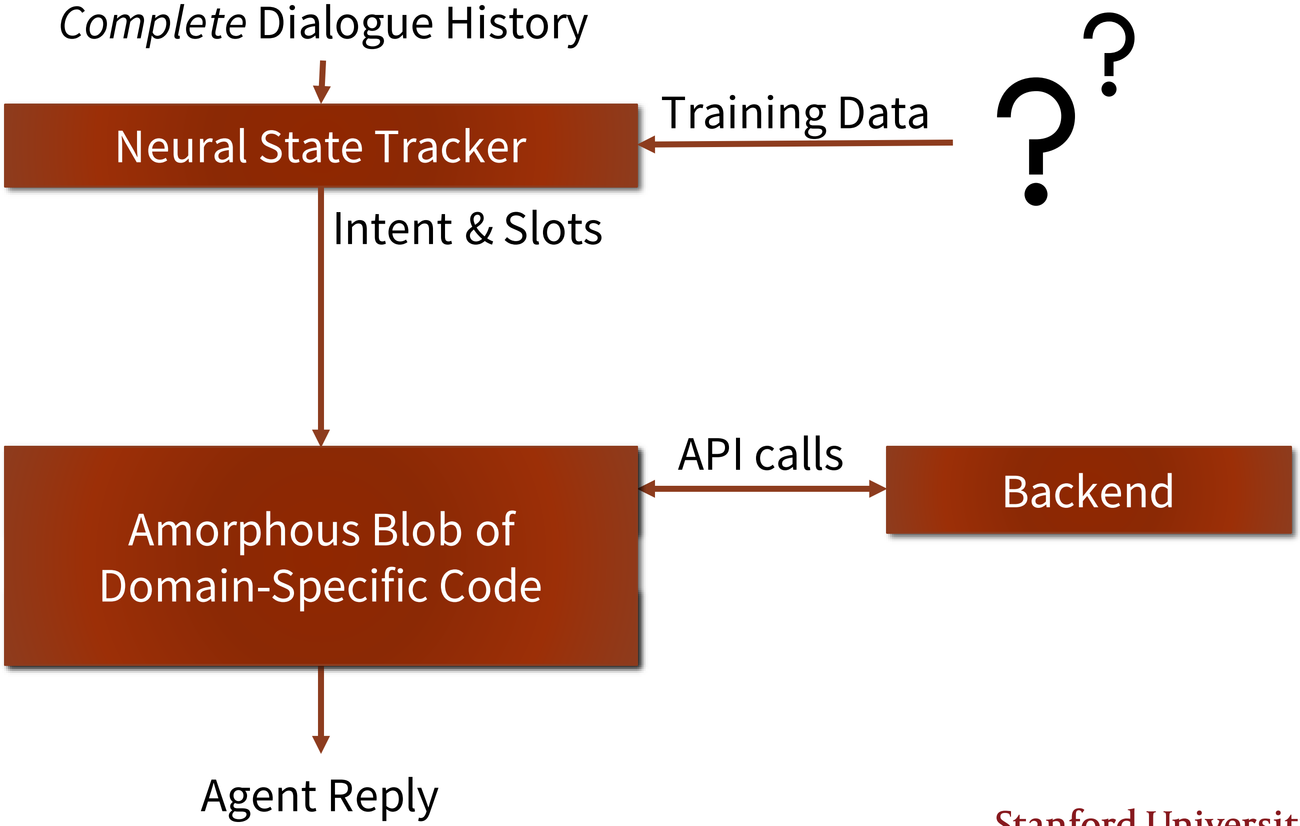
Amorphous Blob of Domain-Specific Code

API calls

Backend

Agent Reply

The Academic Modular Approach To Dialogues


Complete Dialogue History

Neural State Tracker

Training Data

Intent & Slots

Amorphous Blob of Domain-Specific Code

API calls

Backend

Agent Reply

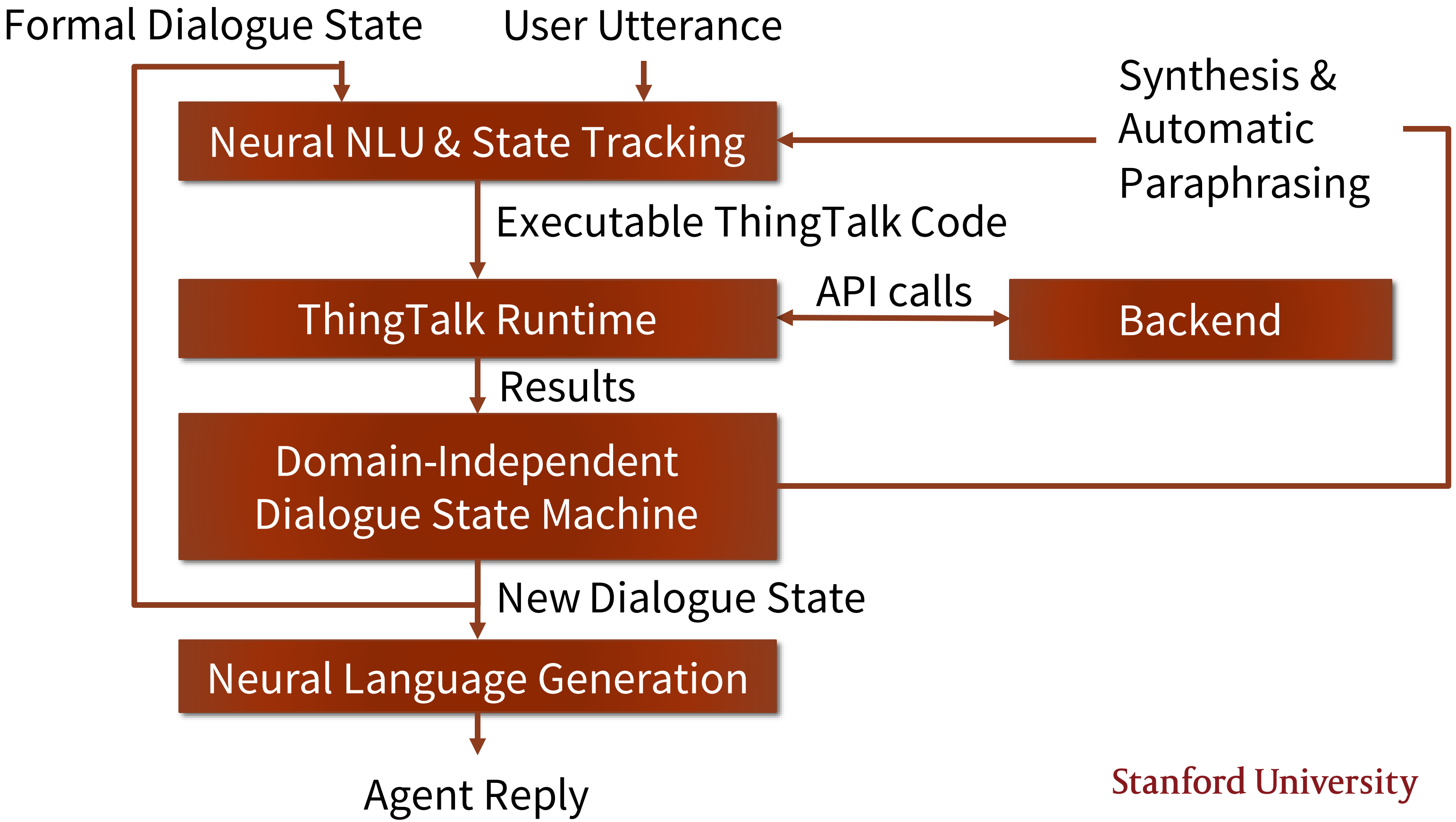
State of the Art: Manually Annotated Conversations

- Dialogues are vast, complex and very varied → **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 70% of the errors are misannotations [Zhou and Small]

Our Approach



Formal Dialogue State

User Utterance

Neural NLU & State Tracking

Synthesis & Automatic Paraphrasing

Executable ThingTalk Code

ThingTalk Runtime

API calls

Backend

Results

Domain-Independent Dialogue State Machine

New Dialogue State

Neural Language Generation

Agent Reply

Key Insights

- Formal, *executable* representation for dialogue states
- Fed to & generated by neural network
- 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
- **Dialogue Act**: pair of utterance and state after the utterance
- **Abstract Dialogue Act**: family of dialogue acts

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 High-Level Structure of Transaction Dialogues

A: Hello, how can I help you?	Greet
U: Hello, I am looking for an Italian restaurant in Palo Alto. A: How about Terun? It’s a highly rated pizza restaurant. U: Is it good for children? A: I think so. 120 reviewers say the ambience is “family friendly”.	Search
U: Ok, can I make a reservation for Friday at 9pm? A: How many people? U: 30 people	Action Slot Fill
A: I am sorry, I cannot find you a table at that time. How about 9:30pm? U: Sure, that works. A: Alright, your reservation is confirmed for Friday 1/24 at 9:30 pm, 30 people.	Execution
Anything else I can help you with?	

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: 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.

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

<Picture>

U: They look cool! I'll take them.

A: Would you like them shipped at your usual address?

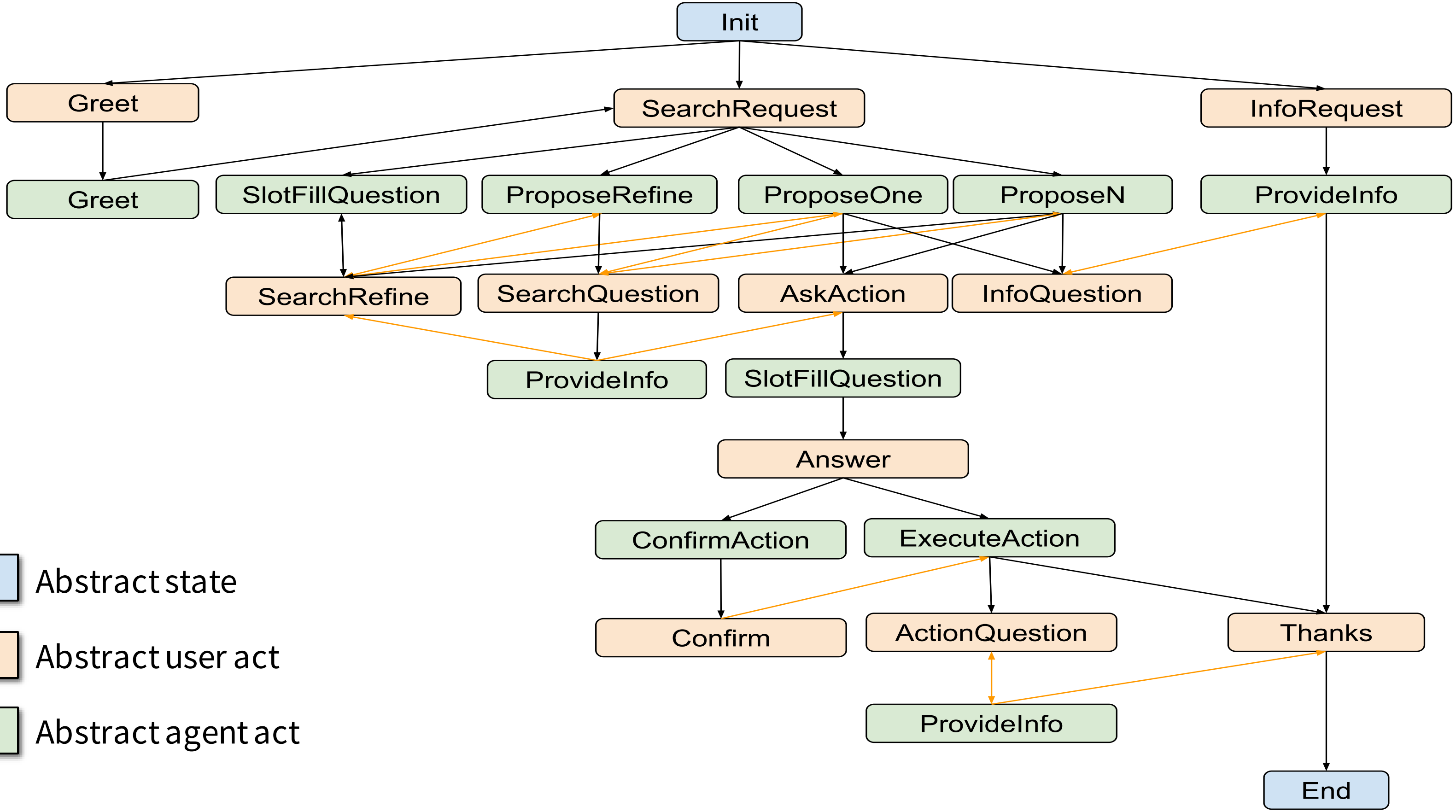
U: Yes please

A: Alright, payment is processing. With 2-day standard shipping, they will arrive Tuesday.

Why Are Transactions Important?

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

A State Machine For Transaction Dialogues



🟦 Abstract state

🟧 Abstract user act

🟩 Abstract agent act

Executable Representations

- 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" ]
```

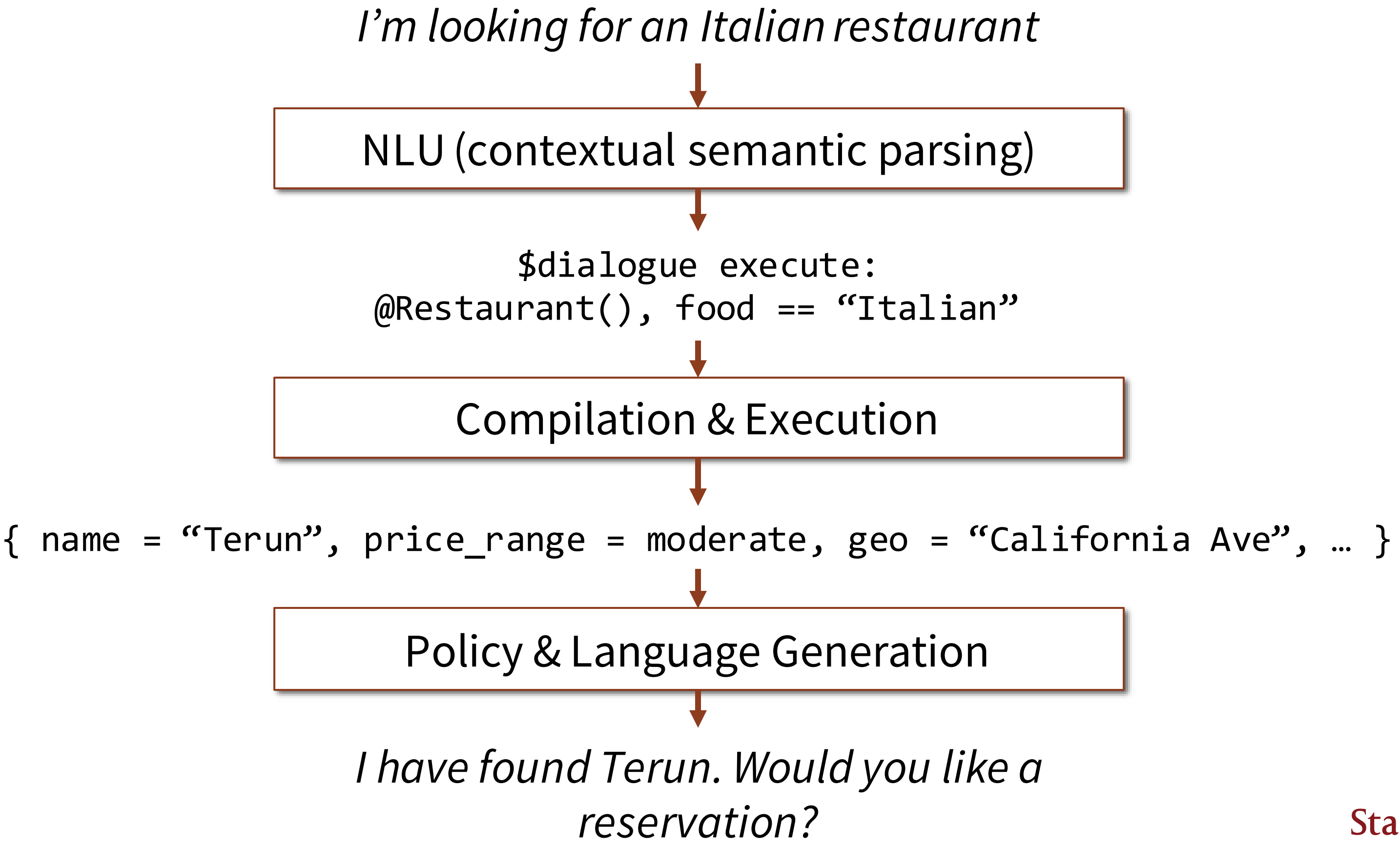
A: I found Terun. Would you like a reservation?

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”
```

Compilation & Execution

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

Policy & 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)];
```

User & Agent Dialogue Act Labels

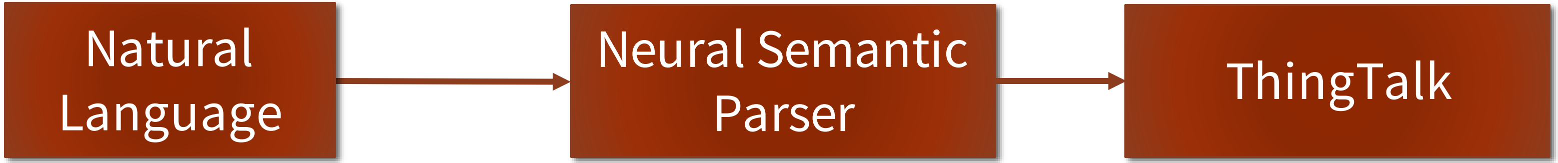
- greet
- execute
- learn_more
- ask_recommend
- cancel
- end

- 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)

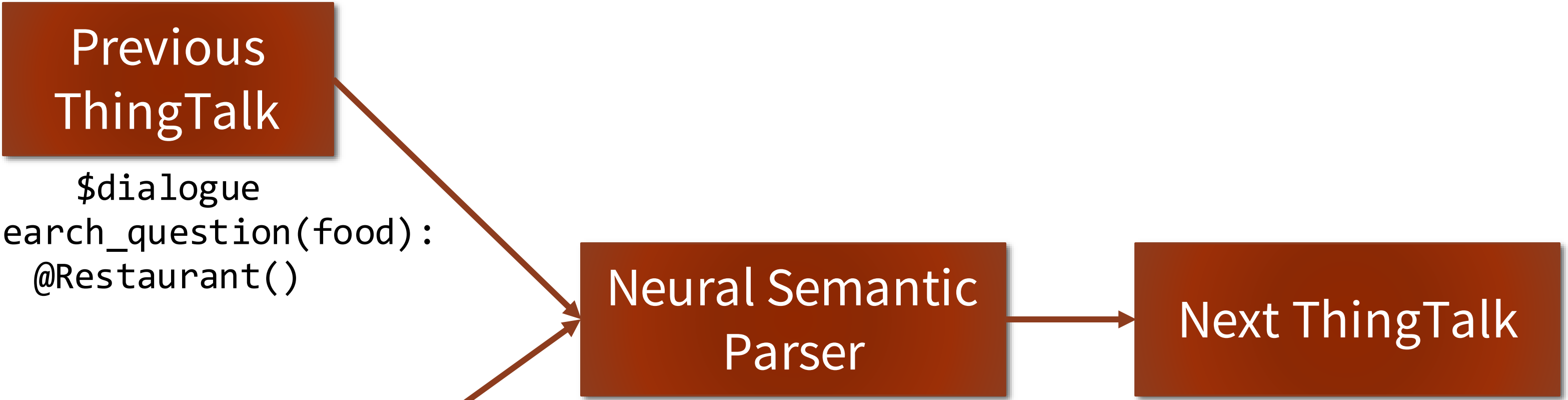
You’ve Seen This Picture Before



I’m looking for an Italian restaurant

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

Adding The Dialogue State

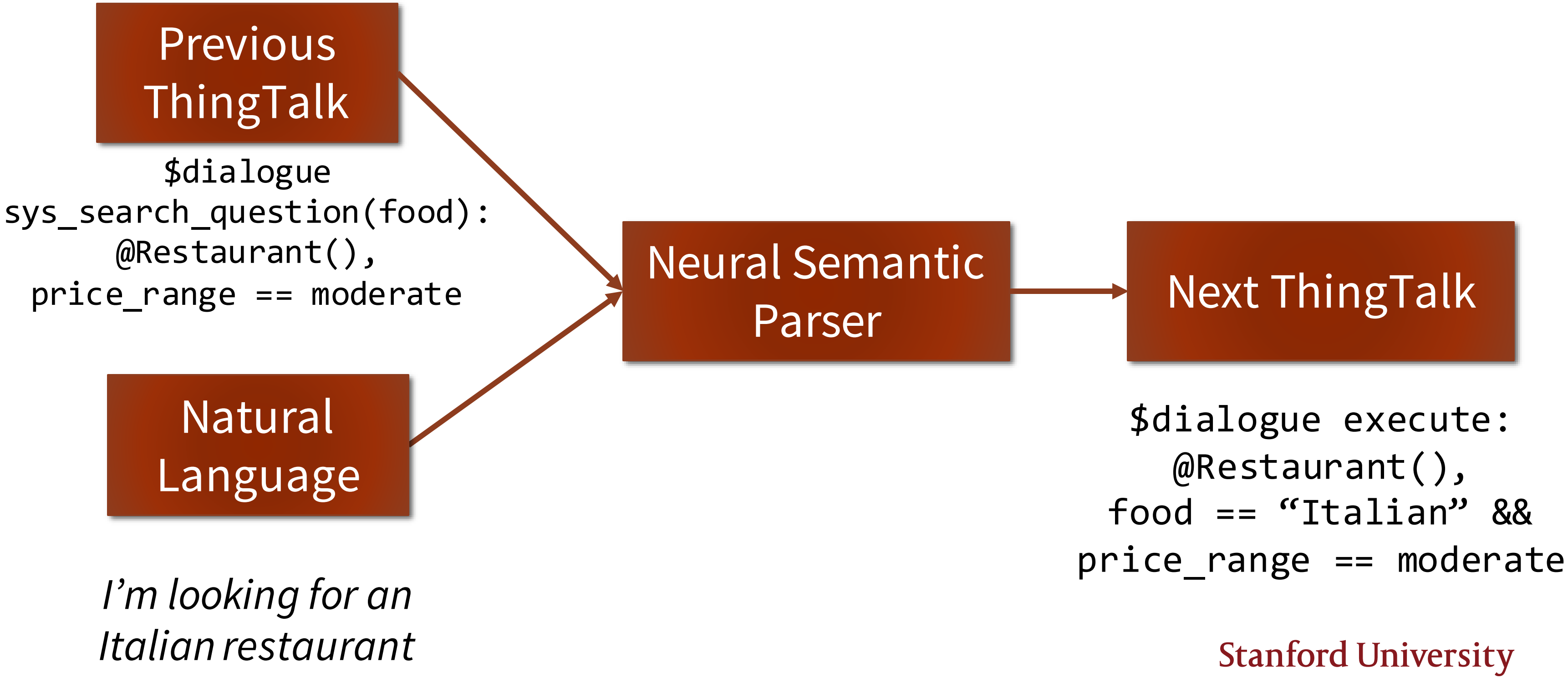


```
$dialogue
sys_search_question(food):
@Restaurant()
```

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

I’m looking for an Italian restaurant

Adding The Dialogue State



Previous ThingTalk

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

Natural Language

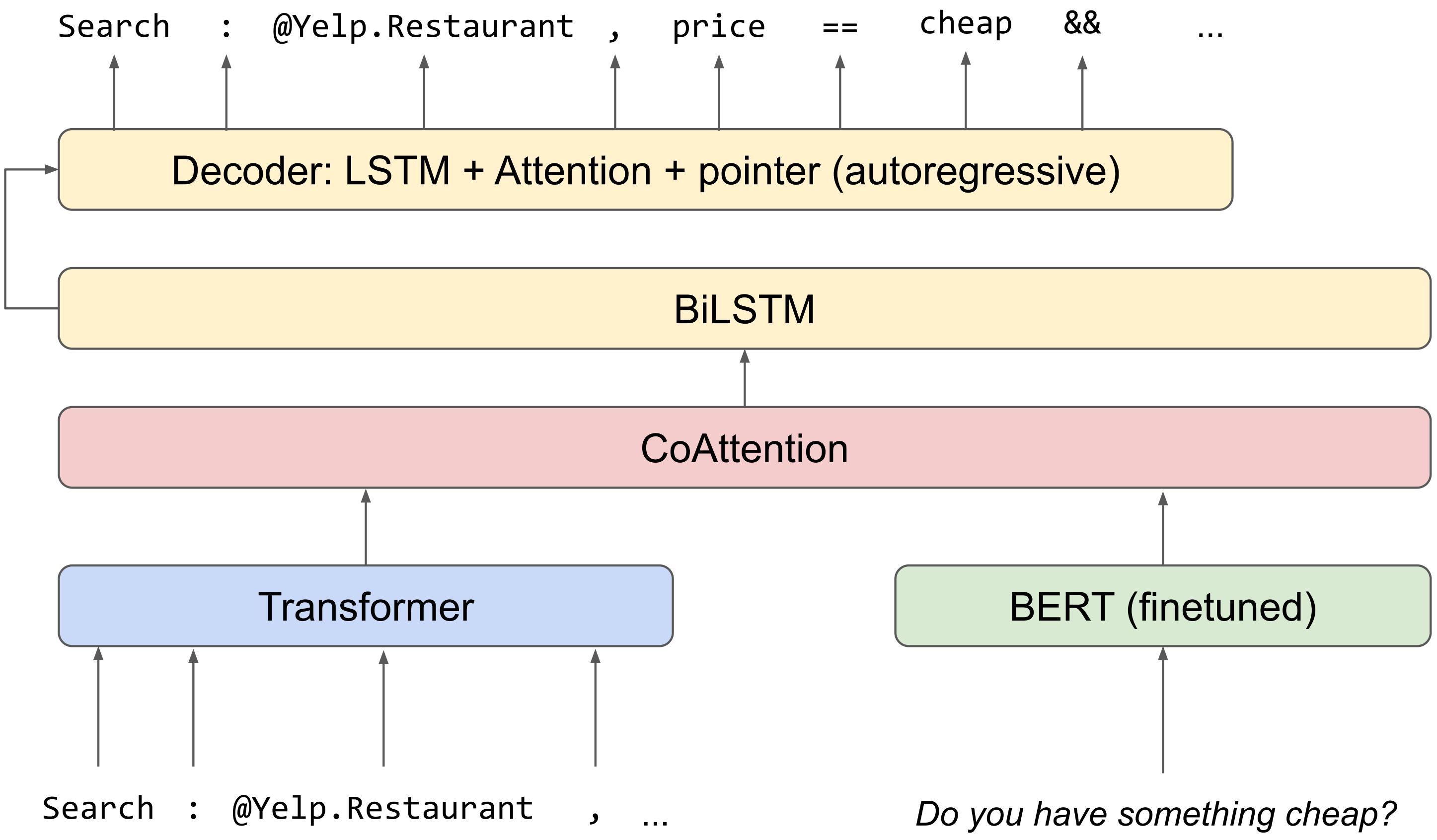
I’m looking for an Italian restaurant

Neural Semantic Parser

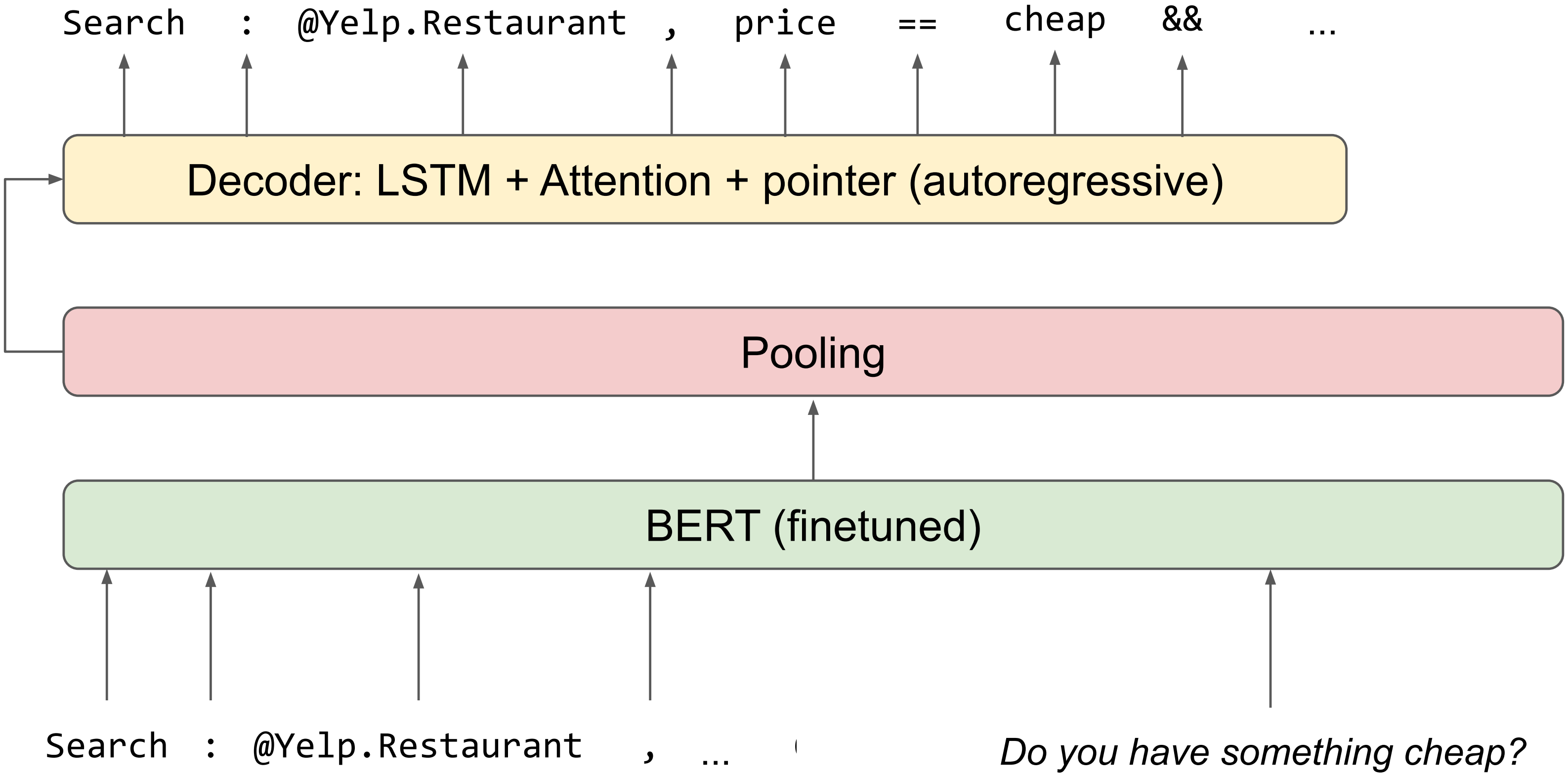
Next ThingTalk

```
$dialogue execute:
@Restaurant(),
food == “Italian” &&
price_range == moderate
```

The Neural Model (Proposal A)


Search : @Yelp.Restaurant , price == cheap && ...
Decoder: LSTM + Attention + pointer (autoregressive)
BiLSTM
CoAttention
Transformer
BERT (finetuned)
Search : @Yelp.Restaurant , ...
Do you have something cheap?

The Neural Model (Proposal B)


Search : @Yelp.Restaurant , price == cheap && ...
Decoder: LSTM + Attention + pointer (autoregressive)
Pooling
BERT (finetuned)
Search : @Yelp.Restaurant , ...
Do you have something cheap?

Quiz 1

What are the advantages & disadvantages of the contextual NLU model vs. training with the full dialogue history?

Pros & Cons of Contextual NLU

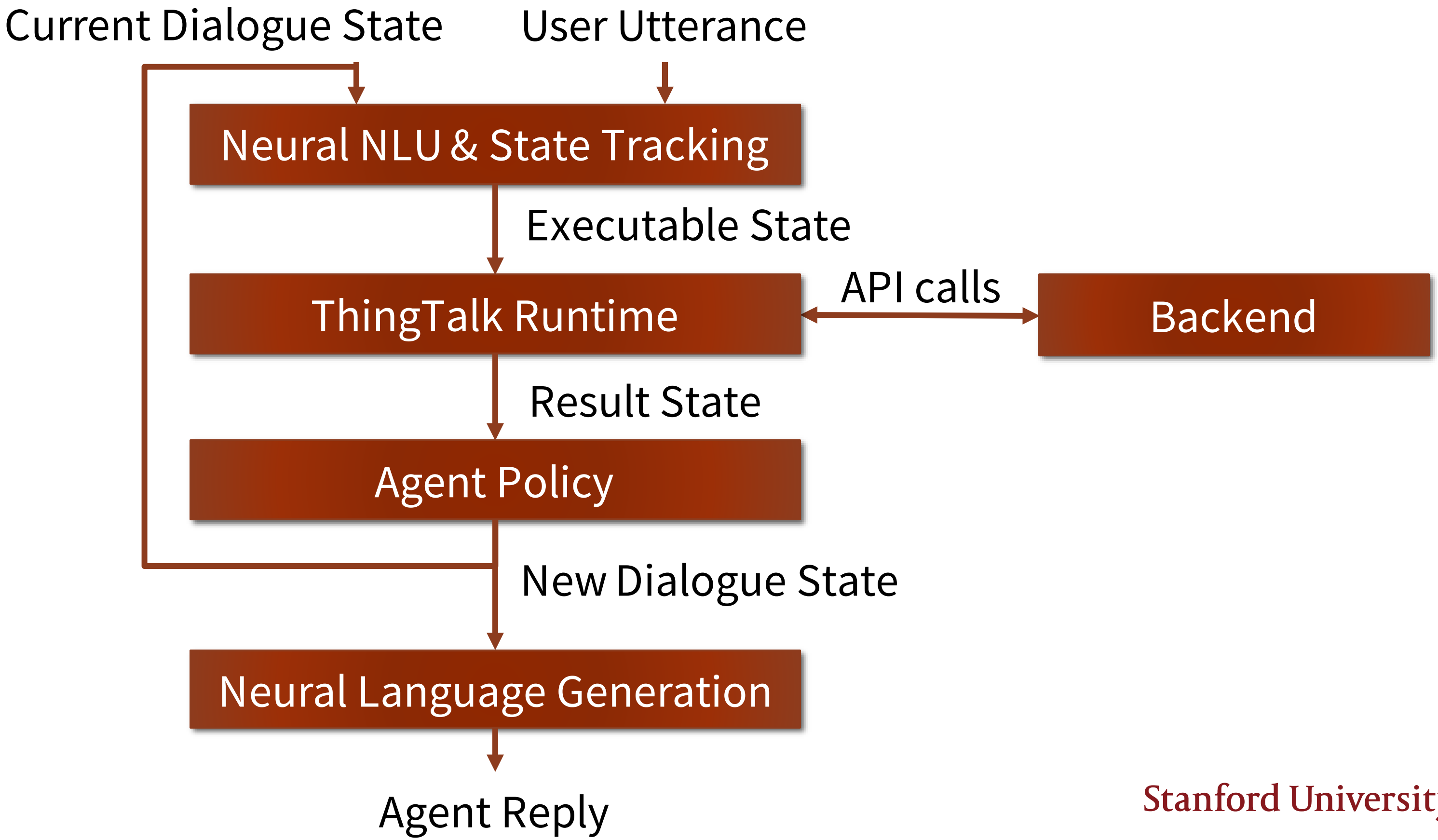
- Positive: **Minimal information to disambiguate input**
    - Formal state removes “extra” information
    - Controllable amount of history to show to network
    - Reduces needed training data
- Positive: **Formal guarantees of agent behavior**
    - Can prevent “bad states” with tools of formal verification

- Negative: **Uncertainty cannot be captured**
    - 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

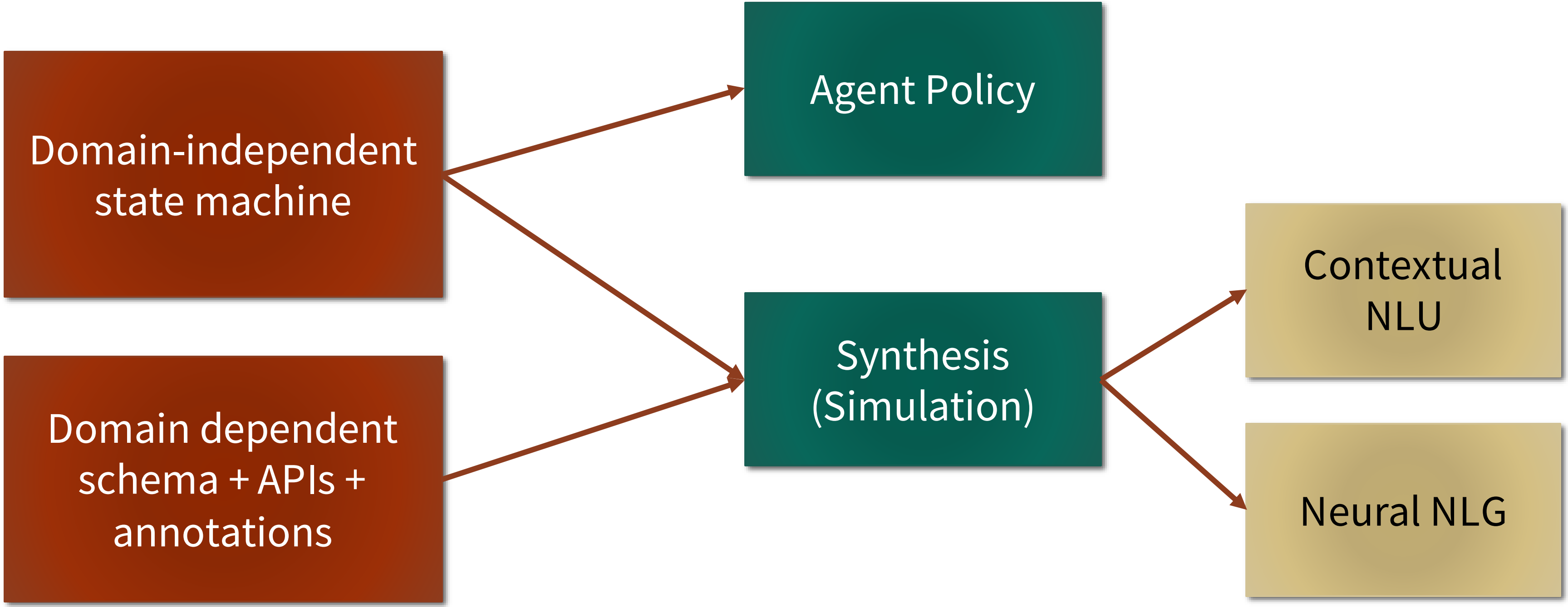
Agent Policy

New Dialogue State

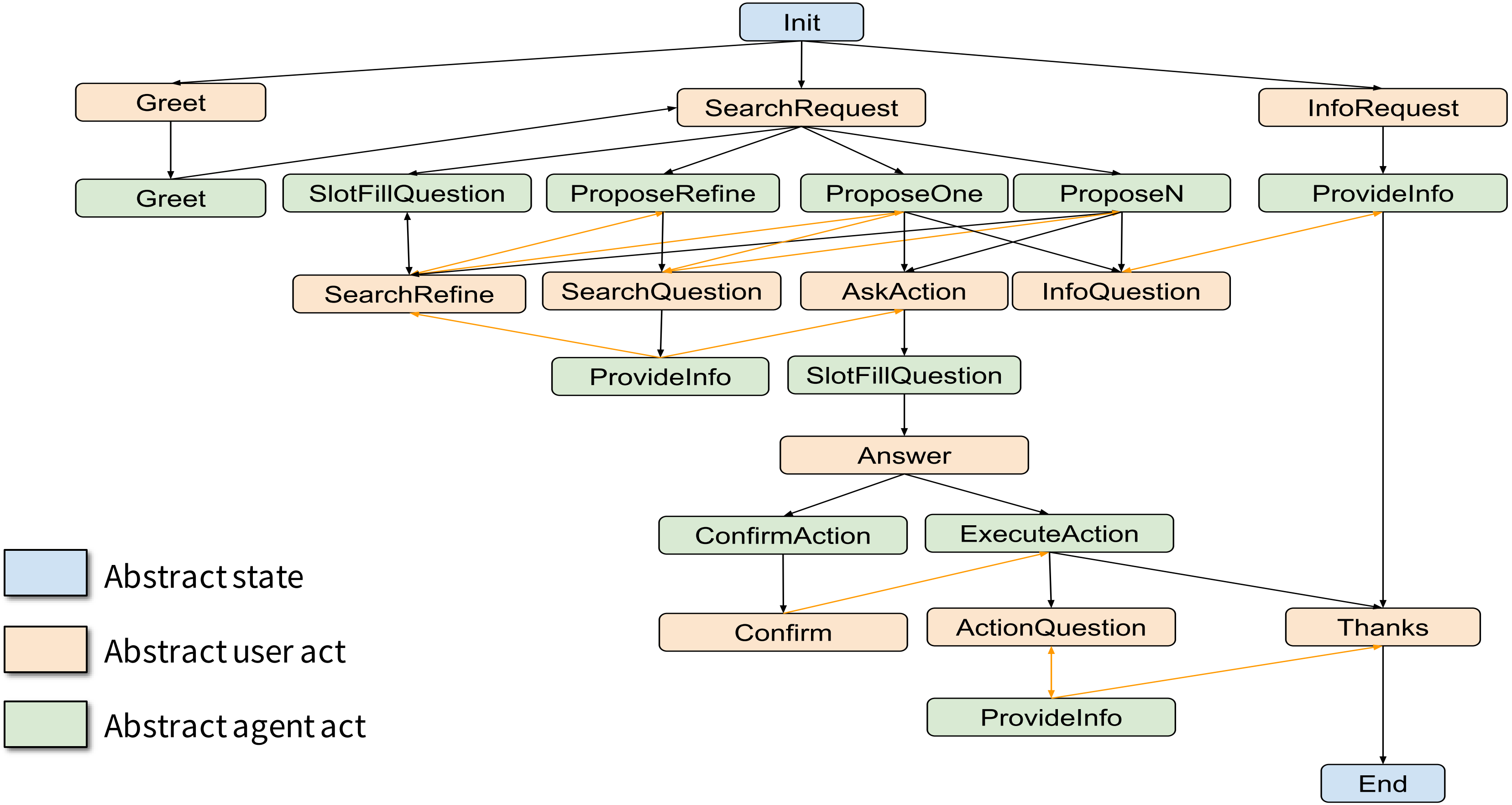
Neural Language Generation

Agent Reply

Specifying The Dialogue Agent


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

Reminder: The Transaction Dialogue State Machine

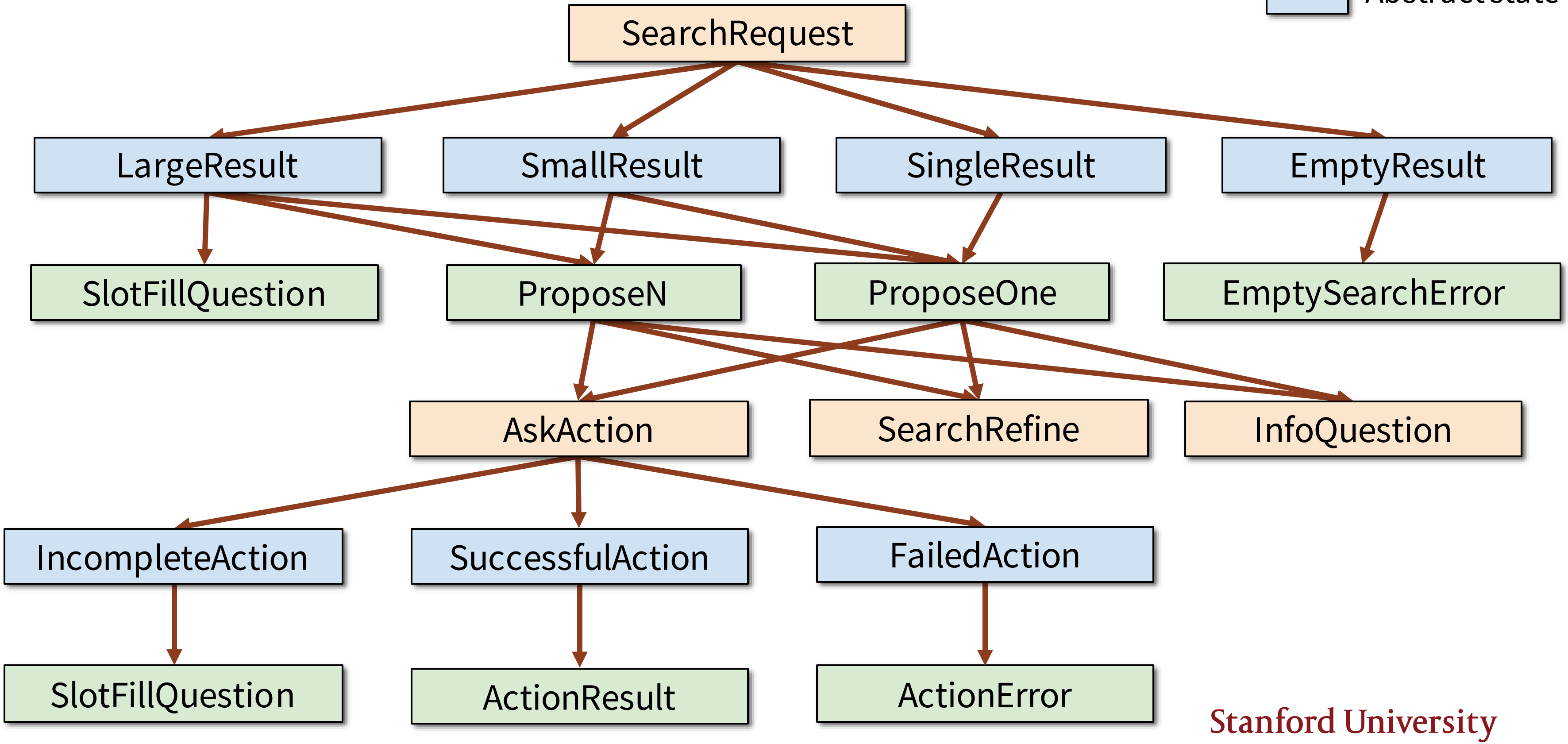


Zooming In…

🟧 User act

🟩 Agent act

🟦 Abstract state



SearchRequest

LargeResult SmallResult SingleResult EmptyResult

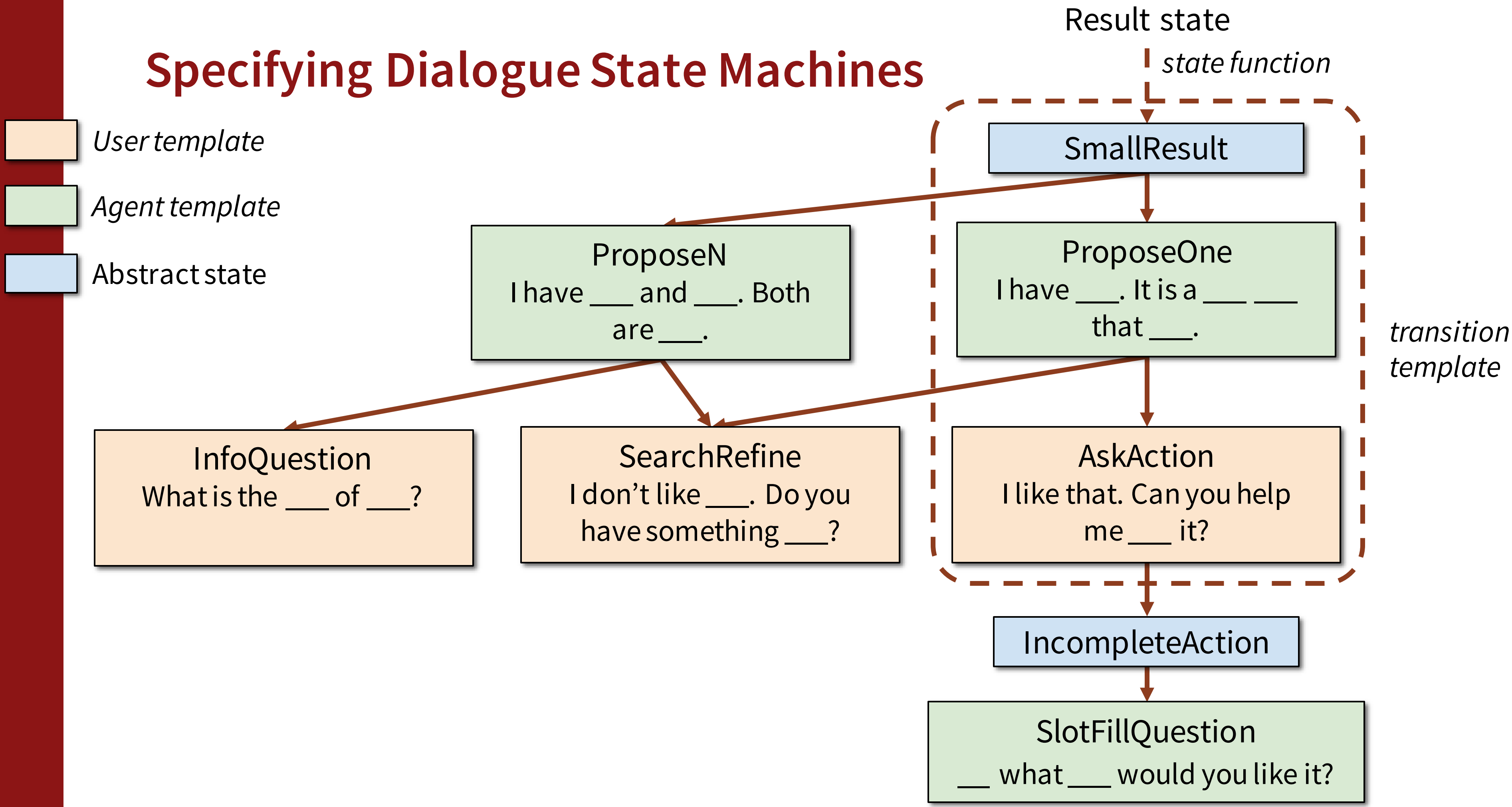
SlotFillQuestion ProposeN ProposeOne EmptySearchError

AskAction SearchRefine InfoQuestion

IncompleteAction SuccessfulAction FailedAction

SlotFillQuestion ActionResult ActionError

Specifying Dialogue State Machines



User template

Agent template

Abstract state

Result state

state function

transition template

SmallResult

ProposeN
I have ___ and ___. Both are ___.

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

InfoQuestion
What is the ___ of ___?

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

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

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

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 #”]
- book_time : Time [“at #”, “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

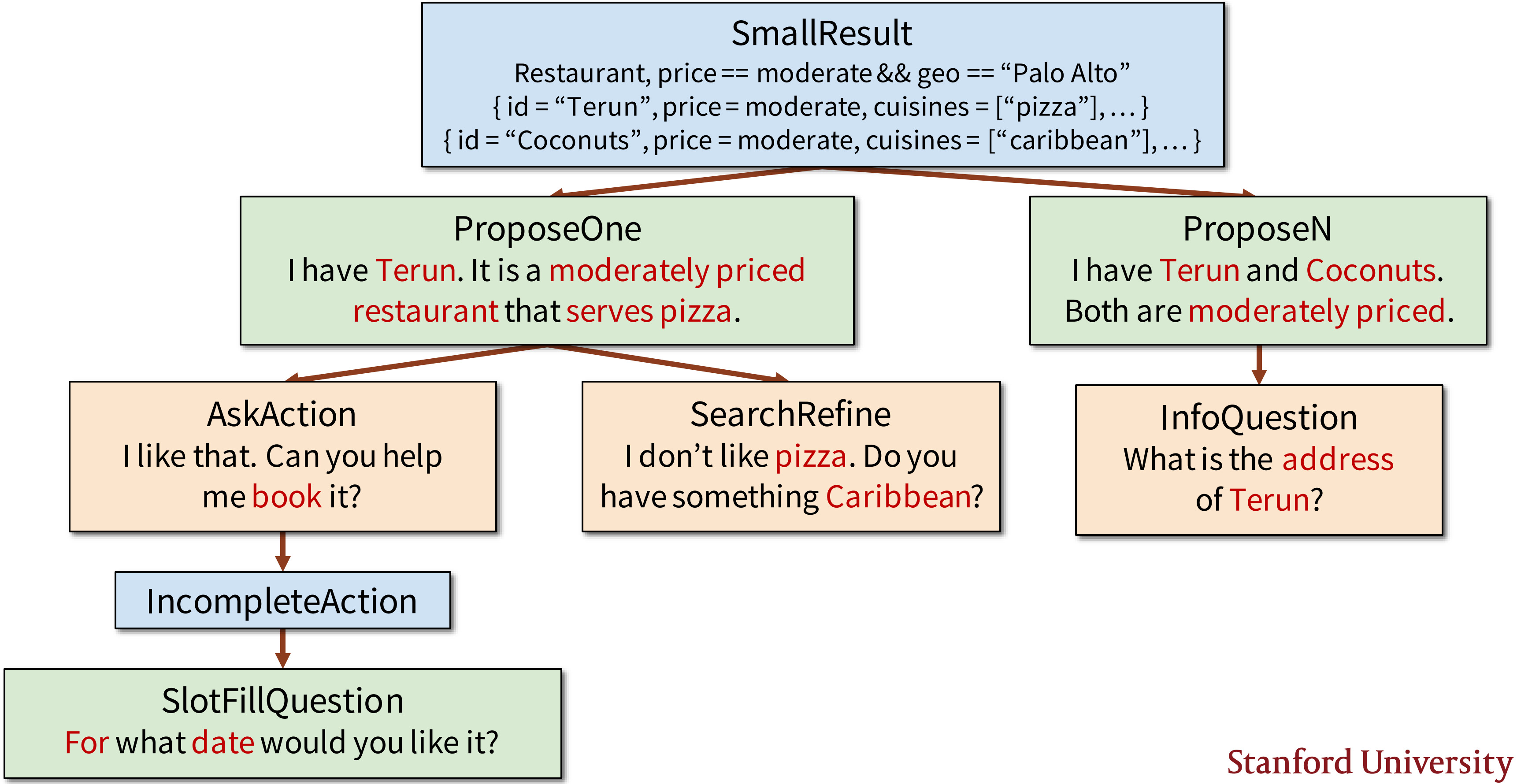
Concretely Doing Things: The Backend

```
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?

InfoQuestion

What is the address of Terun?

IncompleteAction

SlotFillQuestion

For what date would you like it?

Quiz 3

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

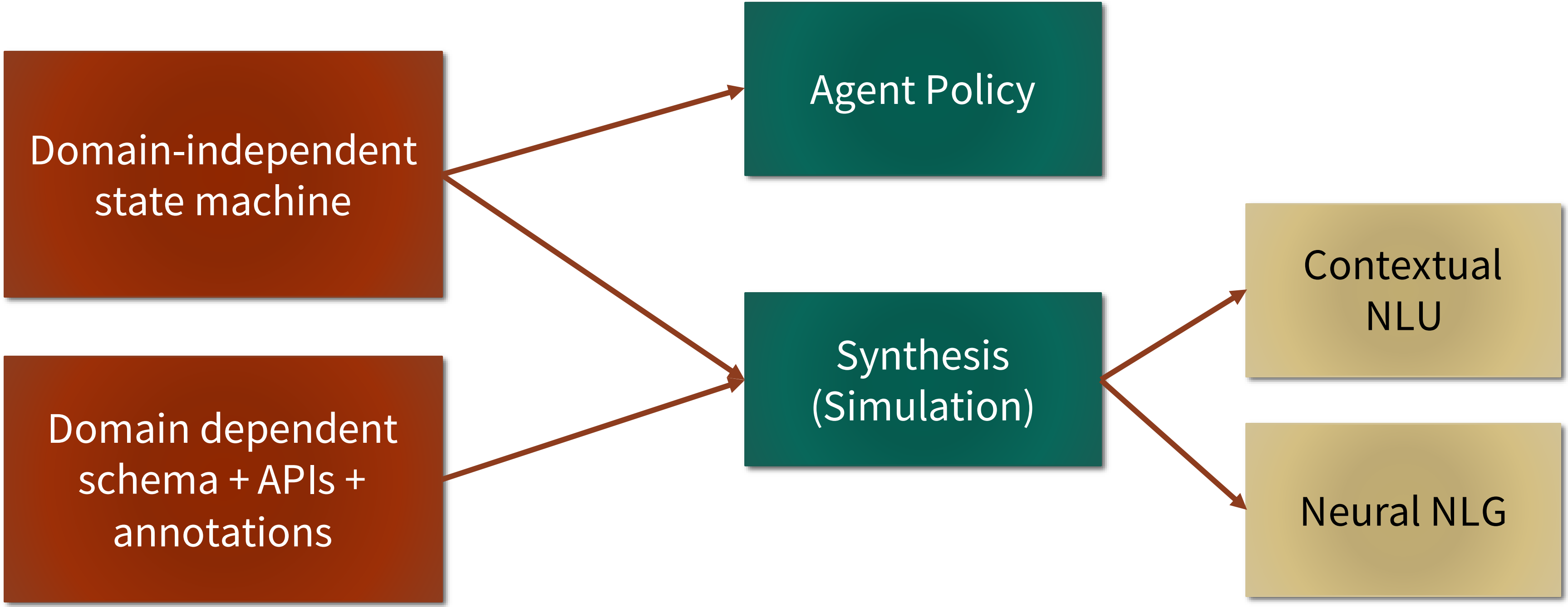
Designing State Machines (For Your Projects)

- 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

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

Quiz 4

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)**

A Preparatory Experiment on MultiWOZ

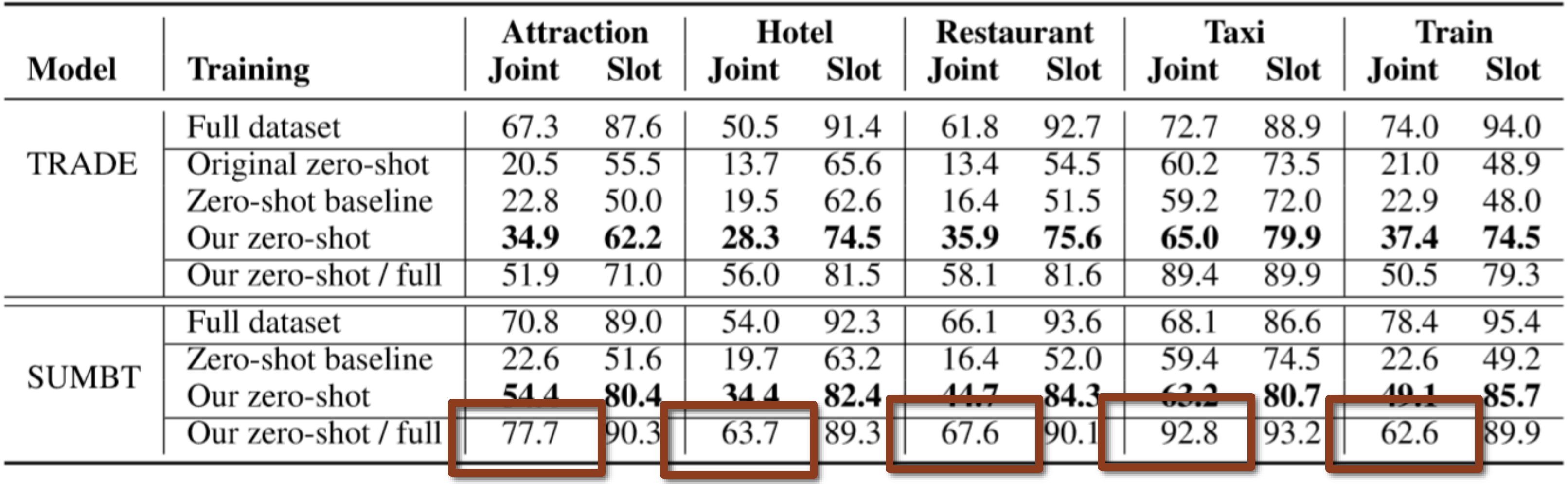
- **Zero-shot domain extension**
  - Train on 4 domains, test on the 5th
  - Zero-shot as in “no new human-written training data”
  - Ok to retrain
  - Ok to beef up the training data with automatically generated data
  - *Domain adaptation*: convert e.g. taxi dialogues to train ticket dialogues

- Representation: **domain + slots**
  - Reuse the existing training data
  - Comes with all the “quirks” (30% bad annotations)

- Models:
  - TRADE [Wu et al.]
  - SUMBT [Lee et al.]

history-based neural dialogue state trackers

Results



Model	Training	Attraction		Hotel		Restaurant		Taxi		Train	
		Joint	**Slot**	**Joint**	**Slot**	**Joint**	**Slot**	**Joint**	**Slot**	**Joint**	**Slot**
TRADE	Full dataset	67.3	87.6	50.5	91.4	61.8	92.7	72.7	88.9	74.0	94.0
	Original zero-shot	20.5	55.5	13.7	65.6	13.4	54.5	60.2	73.5	21.0	48.9
	Zero-shot baseline	22.8	50.0	19.5	62.6	16.4	51.5	59.2	72.0	22.9	48.0
	Our zero-shot	**34.9**	**62.2**	**28.3**	**74.5**	**35.9**	**75.6**	**65.0**	**79.9**	**37.4**	**74.5**
	Our zero-shot / full	51.9	71.0	56.0	81.5	58.1	81.6	89.4	89.9	50.5	79.3
SUMBT	Full dataset	70.8	89.0	54.0	92.3	66.1	93.6	68.1	86.6	78.4	95.4
	Zero-shot baseline	22.6	51.6	19.7	63.2	16.4	52.0	59.4	74.5	22.6	49.2
	Our zero-shot	**54.4**	**80.4**	**34.4**	**82.4**	**44.7**	**84.3**	**63.2**	**80.7**	**49.1**	**85.7**
	Our zero-shot / full	77.7	90.3	63.7	89.3	67.6	90.1	92.8	93.2	62.6	89.9

ThingTalk Results (Work In Progress)

- Manually annotated single-domain restaurant dialogues
- *Turn-by-turn accuracy*

- With only synthesized data: **61%**
  - (Actually fluctuates a lot – 51-55%)
- With *automatic paraphrasing*: **65%**

- Help us boost it up!

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?)

- Cheap, fast, powerful!

Quiz 3 (Open Ended)

How hard is it
to write new dialogue state machines?