NPFL099 - Statistical dialogue systems

Spoken language understanding I

Filip Jurčíček

Institute of Formal and Applied Linguistics
Charles University in Prague
Czech Republic

Home page: http://ufal.mff.cuni.cz/~jurcicek

Version: 12/03/2013

Outline

- Spoken language understanding
- Meaning representation in a dialogue system
- Parsers
  - Phoenix parser
  - Transformation based learning for SLU
- Data preprocessing

Spoken language understanding

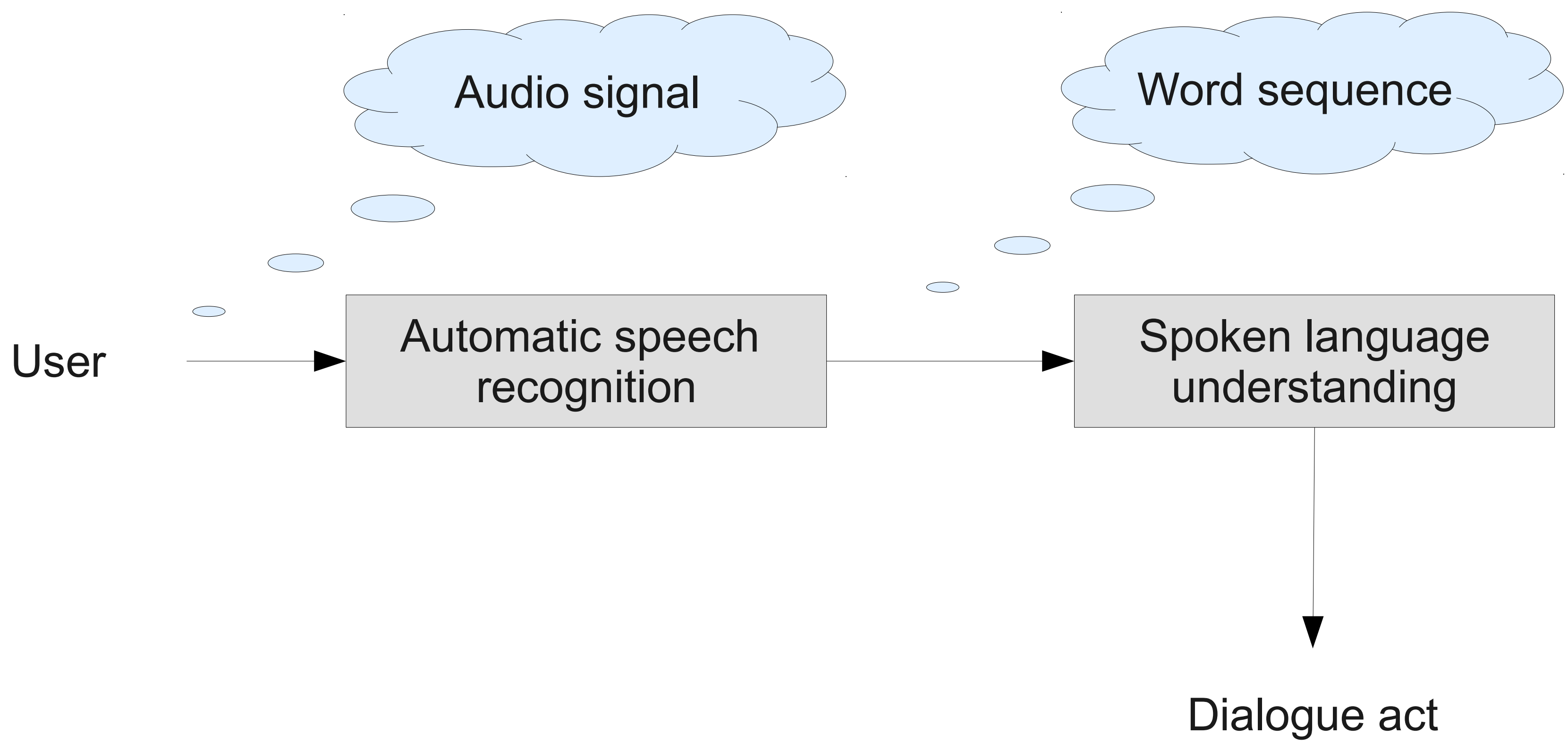
- Definition
  - SLU converts recognised speech into meaning
- For SDS, only basic basic meaning is necessary
  - I am looking for a Chinese restaurant
  - inform(venue=restaurant)&inform(food=Chinese)
- Mostly in the form of dialogue acts

Meaning representation

- Dialogue acts are composed of:
  - a dialogue act type:
    - inform, request, confirm, select, affirm, deny, hello, bye, repeat, help, request_alternatives, etc.
  - semantic information:
    - attribute value pairs
    - domain dependent
    - usually defined by ontology

    - venue=restaurant
    - food=Chinese

SLU in an SDS



inform(venue=restaurant)&inform(food=Chinese)

Example: TownInfo application

- Queries about
  - restaurants, bars, and hotels
- Search constraints
  - area, price range, stars
- Provides
  - address, postcode, phone number

Typical conversation

Turn	Transcription	Dialogue act
System	Hello. How may I help you?	hello()
User	Hi, I am looking for a restaurant.	inform(venue=restaurant)
System	What type of food would you like?	request(food)
User	I want Italian.	inform(food=Italian)
System	Did you say Italian	confirm(food=Italian)

Real user input

User	0.4 hi I am looking for a restaurant 0.2 uhm am looking for a bar	0.7 inform(venue=restaurant) 0.3 inform(venue=bar)
System	Did you say that you are looking for a restaurant?	confirm(venue=restaurant)

Example: TrainInfo application

- Queries about
  - departures, arrivals
- Search constraints
  - station of a query, from, to, through, planned time
- Provides
  - platform number, delay, real time of departure, real time of arrival

Frame based approach

- Utterances are composed of frames
- Frame(s) is a hierarchical structure

- Frame is composed of
  - Slots
  - Or other Frames

- All of this is equivalent to CFG

Example: Semantic frames

DARPA Communicator

```
clause:
{ display
    topic:
    { flight
        number: pl
        predicate:
        { from
            topic: { city name:Boston }
        }
        predicate:
        { to
            topic: { city name:Denver }
        }
    }
}
```

interpretation of “Show me flights from Boston to Denver”

Design of meaning representation

- Aim to capture all important aspects in an utterance
- Transform ambiguous natural redundant input into a unambiguous formal representation
- Every SDS has usually its own meaning representation
  - a set of DAs

Information State Update

- Meaning representation provides instruction to SDS's how to update dialogue state
- Dialogue state
  - A collection of variables used to track progress in a dialogue

Dialogue state

Dialogue state is used to track the progress of the dialogue

- Turn 1:
  - S: How may I help you?
  - Dialogue state:
    - venue = None
    - food = None
    - price = None

User says

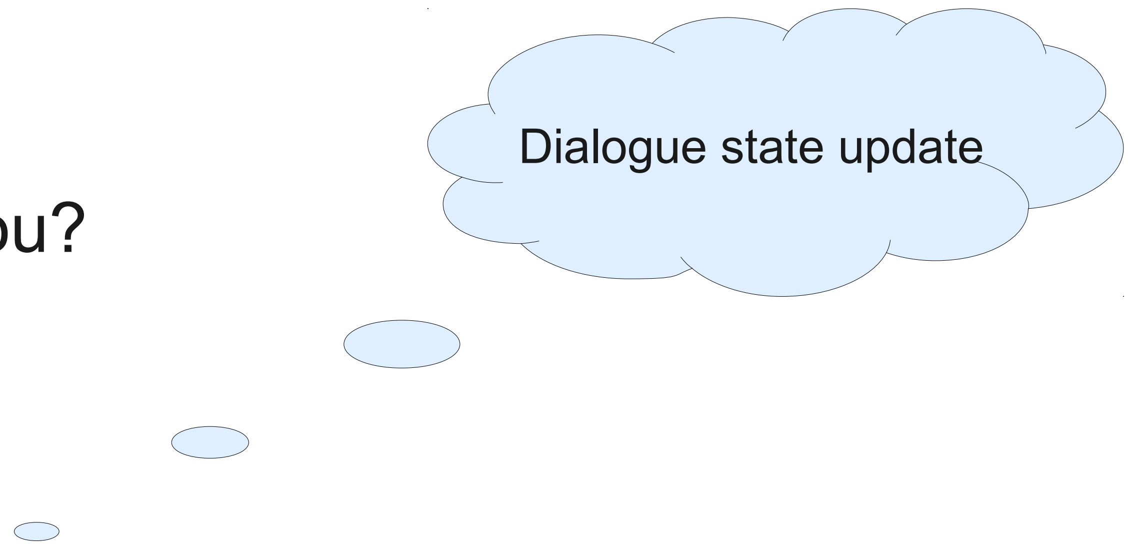
Dialogue state is used to track the progress of the dialogue

- Turn 1:
  - S: How may I help you?
  - Dialogue state:
    - venue = None
    - food = None
    - price = None
  - U: inform(venue = restaurant)

Dialogue state update

Dialogue state is used to track the progress of the dialogue

- Turn 1:
  - S: How may I help you?
  - Dialogue state:
    - venue = restaurant
    - food = None
    - price = None
  - U: inform(venue = restaurant)



Dialogue act set

- Slot level DAs
  - inform – I want Chinese restaurant
  - deny – I do not want Chines
  - request – What is the phone number
  - confirm – Is it cheap
  - select – Is it cheap or expensive (S)

- Others
  - hello
  - bye
  - thankyou

Dialogue act set

- Others
  - ack – back-channel: uhm, fine
  - affirm – Yes
  - negate – No
  - reqalts – Do you have anything else
  - reqmore – Can you give me more details
  - repeat
  - help
  - restart
  - null – does nothing, uninterpretable input

Challenges of SLU

- Repetitions
  - Erm, I want I want something in the city centre.
- Irrelevant content
  - If it is not too much trouble I would be very grateful of some one could tell me whether there is a Chinese restaurant which is not very expensive and close to the city centre, thank you.
- Missing content
  - Chinese city centre

Types of SLU components

- Handcrafted
  - Rule and grammar based
- Data driven
  - Rules and grammar based
  - Kernel techniques such as SVM
  - Probabilistic
    - FSM
    - Logistic Regression
    - CRF
    - DBN

Phoenix parser

- Allows for:
  - Robust parsing
  - Parses what is important
  - Ignore irrelevant bits
  - Follows frame based approach

- Based on robust combination of multiple CFGs
- Allows garbage between consecutive CFGs
- Greedy
  - Tries to match as little CFG as possible
  - Prefers frames where all slots are presented

Phoenix grammar example

```
# reserve hotel room
FRAME: Hotel
NETS:
  [hotel_request]
  [hotel_name]
  [hotel_period]
  [hotel_location]
  [Room_Type]
  [Arrive_Date]
  [want]
;
```

```
[hotel_request]
  (*[want] *a HOTEL)
HOTEL
  (hotel)
  (accommodations)
  (place to stay)
;

[want]
  (*I WANT)
  I
  (i)
  (we)
WANT
  (want)
  (would like)
;
```

Grammar notation

- The grammar is composed of slots
  - slot names are in square brackets
  - in between are strings of words
- The words there are of three types:
  - **standard words:** these are natural language words, they are always written in lower case
  - **slot names:** slots are defined recursively, you can use slots within other slots
  - **variables**: are all-caps words, they behave like slots but are only defined within particular slot definition

Grammar notation

- "*" indicates 0 or 1 word repetitions
- "+" indicates 1 or more repetitions
- "+*" indicates 0 or more repetitions

```
# reserve hotel room
FRAME: Hotel
NETS:
  [hotel_request]
  [hotel_name]
  [hotel_period]
  [hotel_location]
  [Room_Type]
  [Arrive_Date]
  [want]
;
```

```
[hotel_request]
  (*[want] *a HOTEL)
HOTEL
  (hotel)
  (accommodations)
  (place to stay)
;

[want]
  (*I WANT)
  I
  (i)
  (we)
WANT
  (want)
  (would like)
;
```

Phoenix parser output

- Input
  - I would like a hotel room
- Output
  - [hotel_request]([want](i would like) a hotel room)

Phoenix summary

- Phoenix
  - CFGs can be shared
  - only accepts things in the grammar
  - can be restrictive, e.g. not accepting valid input

Transformation based learning

- Based on an idea of inferring a set of self correcting rules
- Initially used in part-of-speech tagging

- Advantages
  - comparable to the state-of-the-art statistical methods
  - results in a small compact set rules
  - it is fast !!! / it is not probabilistic

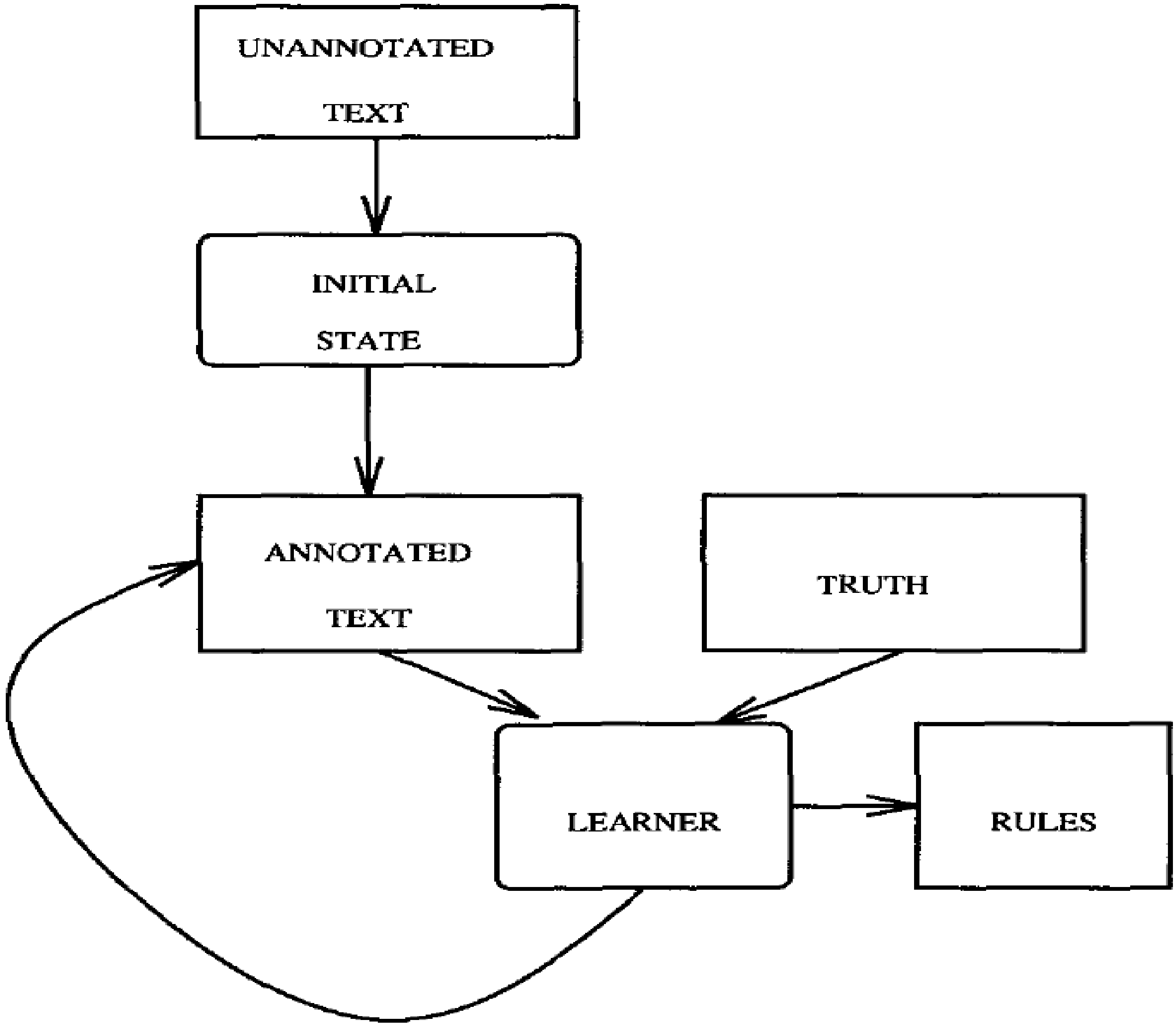
F. Jurčíček, M. Gašić, S. Keizer, F. Mairesse, B. Thomson, K. Yu, S. Young: Transformation-based learning for semantic parsing. In: Proc. Interspeech, Brighton, United Kingdom, 2009.

Basic idea on POS tagging

- Input
  - Example input for the Brill tagger
- Output
  - Example/NN input/NN for/IN the/DT Brill/NNP tagger/NN

- Uses an ordered list of rules
  1. if w_t = example then t_t = NN
  2. if w_t = the then t_t = DT
  3. if w_t = for then t_t = IN
  4. ..
  5. if w_t = input then t_t = VB
  6. ..
  7. if t_t = VB & w_{t-1} = example & t_{t+1} = IN then t_t = NN

Training procedure



TBL for SLU

- Transforms an initial semantic hypothesis into the correct semantics
  - by applying an ordered list of transformation rules
- Initial semantic hypotheses
  - inform()
- In each iteration
  - a transformation rule corrects some of the remaining errors in the semantics

Rules

- Have two components
  - trigger
  - transformation

- Trigger
  - matched against both the utterance and the semantic hypothesis

- Transformation
  - only if the trigger successfully matched
  - it is applied to the current hypothesis

Trigger

- Trigger contains one or more conditions as follows:
  - the utterance contains N-gram N
  - the dialogue act type equals D
  - and the semantics contains slot S
  - all included conditions must be satisfied
- N-gram triggers can be
  - unigrams, bigrams, trigrams
  - skip-ping bigrams which can skip up to 3 words
    - looking * * * bar

Transformation

- Available operations
  - replace the dialogue act type
  - add a slot
  - delete a slot,
  - replace a slot

Trigger	Transformation
I want	replace DAT by ``inform''
can * give & DAT=inform	replace DAT by ``request''
cheap	add the slot ``pricerange=cheap''
centre	add the slot ``area=centre''
near	replace the slot ``area=*'' by ``near=*''

Parsing example

- Text:
  - I am at the west side shopping centre could you tell me a nearby hotel
- Initial semantics
  - DAT = inform

#	trigger	transformation
1	hotel	add the slot type=hotel
2	centre	add the slot area=centre

- Partial semantics
  - DAT = inform
  - type = hotel
  - area = centre

Parsing example

- Text:
  - I am at the west side shopping centre could you tell me a nearby hotel
- Partial semantics
  - DAT = inform
  - type = hotel
  - area = centre

#	trigger	transformation
3	west side shopping” & area=centre	replace the slot “area=centre” by “near=west side shopping”

- Final semantics
  - DAT = inform
  - type = hotel
  - near = west side shopping

Long-range dependencies

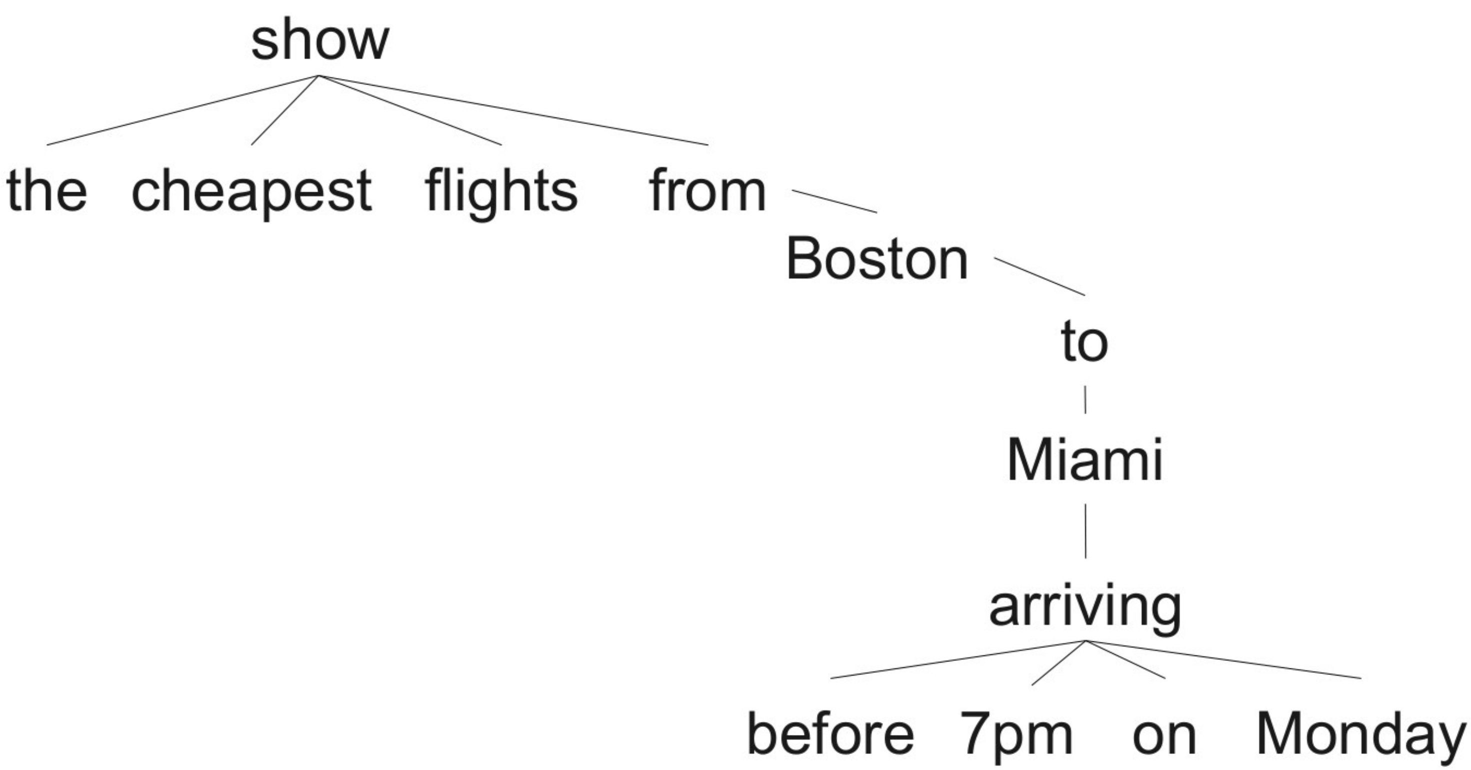
- Bigrams and trigrams are not good in capturing long range dependencies between words

- In general, N-grams fragment data
  - I am looking for a restaurant
  - I am looking for a cheap restaurant
  - I am looking for a cheap beautiful comfortable restaurant

- Use dependency trees
  - long-range dependencies from an utterance tend to be local in a dependency tree

Dependency tree

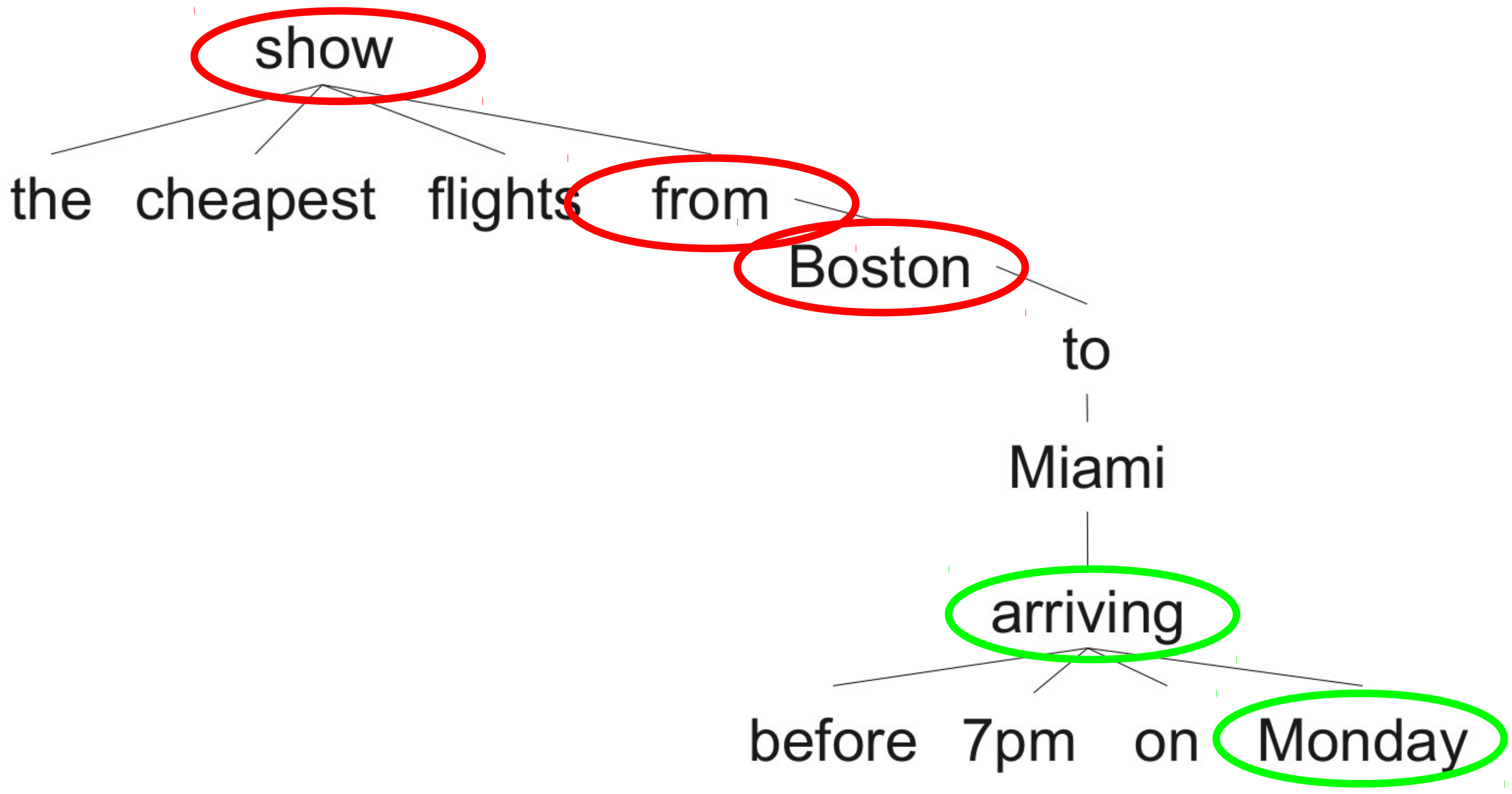
- Each word is viewed as the dependant of one other word, with the exception of the root.
- Dependency links represent grammatical relationships between words



show the cheapest flights from Boston to Miami arriving before 7pm on Monday

Features from a dep. tree

- Bigrams, trigrams, ...
  - following the structure of the tree



Training procedure

Input: a set of (utterance, semantic tree) pairs

Output: a classifier of the input utterance

1. Assign initial semantics to each utterance.
2. Repeat as long as the number of errors on the training set decreases:
   a) Generate all rules which correct at least one error in the training set.
   b) Measure the number of errors corrected minus the number of errors introduced by each rule.
   c) Select the rule with the largest number of corrected errors.
   d) Stop if the number of corrected errors is smaller than threshold T.
   e) Add the selected rule to the end of the rule list and apply it to the current state of the training set.

Text input pre-processing

- Remove
  - uhm, err, uh output from ASR
- Convert
  - I'm → I am
  - …

Text input pre-processing

- Remove filler words

It was used on CUED DAs.

- Replace surface forms of slot values with their category labels, e.g. slot names
  - affirm(area="central",type="hotel")
  - yes i'd like a hotel in the centre of town
- to
  - affirm(area=AREA-0,type=TYPE-0)
  - yes i'd like a TYPE-0 in the AREA-0 of town

  - TYPE-0 = hotel
  - AREA-0 = centre

Text input pre-processing

- For Phoenix, it does not matter
  - it is handcrafted anyway

- In the case of data driven approaches
  - it significantly helps for low price
  - e.g. 93.2% → 94.2% in F-measure in TownInfo domain

Summary

- Meaning representation in a dialogue system
- Parsers
  - Phoenix parser
  - Transformation based learning for SLU
- Data preprocessing
  - category label substitution
- Processing multiple hypotheses

Thank you!

Filip Jurčíček

Institute of Formal and Applied Linguistics
Charles University in Prague
Czech Republic

Home page: http://ufal.mff.cuni.cz/~jurcicek

Processing multiple hypotheses

- ASR provides N-best list
  - `0.33 – I am looking for a bar`
  - `0.26 – I am looking for the bar`
  - `0.11 – I am looking for a car`
  - `0.09 – I am looking for the car`
  - `...`
- How do we get?
  - `0.59 – inform(task=find, venue=bar)`
  - `0.20 – null()`
  - `...`

Processing multiple hypotheses

- Semantic parser: $P(d|w)$
- Automatic speech recognition: $P(w|a)$
- We want to get: $P(d|a)$
- where
  - d – dialogue act
  - w – word sequence
  - a – audio signal

Processing multiple hypotheses

- ASR provides multiple word sequence hypotheses
  - we have to sum over them

$$P(d|a)=\sum_{w} P(d|w)\,P(w|a)$$

- Algorithm
  - Compute semantic interpretation for every word seq.
  - Weight them by the prob. of the word sequence
  - Merge the same dialogue acts and sum their probs.

Alternative

- ASR provides $P(w|a)$
  - map directly from probability distribution to dialogue acts

$$P(d|a) = P(d|P(w|a))$$

$$P(d|a) \approx e^{\theta^T \cdot \Phi_d(P(w|a))}$$

- This approach
  - will be explained in the next lecture