NLU & Advanced Dialog Models

Ling575
Spoken Dialog Systems
April 24, 2013

Meaning Representations for Spoken Dialog

- Typical model: Frame-slot semantics
  - Majority of spoken dialog systems
    - Almost all deployed spoken dialog systems

Meaning Representations for Spoken Dialog

- Typical model: Frame-slot semantics
  - Majority of spoken dialog systems
    - Almost all deployed spoken dialog systems

- Frame:
  - Domain-dependent information structure
  - Set of attribute-value pairs
  - Information relevant to answering questions in domain

Natural Language Understanding

- Most systems use frame-slot semantics
  Show me morning flights from Boston to SFO on Tuesday

  - SHOW:
  - FLIGHTS:
    - ORIGIN:
      - CITY: Boston
      - DATE:
        - DAY-OF-WEEK: Tuesday
      - TIME:
        - PART-OF-DAY: Morning
    - DEST:
      - CITY: San Francisco

Another NLU Example

- Sagae et 2009
- Utterance (speech): we are prepared to give you guys generators for electricity downtown
- ASR (NLU input): we up apparently give you guys generators for a letter city don town
- Frame (NLU output):
  - **<s>.mood declarative**
  - **<s>.sem.agent kirk**
  - **<s>.sem.event deliver**
  - **<s>.sem.modal.possibility can**
  - **<s>.sem.speechact.type offer**
  - **<s>.sem.theme power-generator**
  - **<s>.sem.type event**

Question

- Given an ASR output string, how can we tractably and robustly derive a meaning representation?

Question

- Given an ASR output string, how can we tractably and robustly derive a meaning representation?
- Many approaches:
  - Shallow transformation:
    - Terminal substitution

Question

- Given an ASR output string, how can we tractably and robustly derive a meaning representation?
- Many approaches:
  - Shallow transformation:
    - Terminal substitution
  - Integrated parsing and semantic analysis
    - E.g. semantic grammars

Question

- Given an ASR output string, how can we tractably and robustly derive a meaning representation?
- Many approaches:
  - Shallow transformation:
    - Terminal substitution
  - Integrated parsing and semantic analysis
    - E.g. semantic grammars
  - Classification or sequence labeling approaches
    - HMM-based, MaxEnt-based

Grammars

- Formal specification of strings in a language
- A 4-tuple:
  - A set of terminal symbols: Σ
  - A set of non-terminal symbols: N
  - A set of productions P: of the form A -> α
  - A designated start symbol S
- In regular grammars:
  - A is a non-terminal and α is of the form $\{N\}\Sigma^*$
- In context-free grammars:
  - A is a non-terminal and α in $(\Sigma \cup N)^*$

Simple Air Travel Grammar

- LIST -> show me | I want | can I see|…
- DEPARTTIME -> (after|around|before) HOUR| morning | afternoon | evening
- HOUR -> one|two|three…|twelve (am|pm)
- FLIGHTS -> (a) flight|flights
- ORIGIN -> from CITY
- DESTINATION -> to CITY
- CITY -> Boston | San Francisco | Denver | Washington

Shallow Semantics

- Terminal substitution
  - Employed by some speech toolkits, e.g. CSLU

Shallow Semantics

- Terminal substitution
  - Employed by some speech toolkits, e.g. CSLU
- Rules convert terminals in grammar to semantics
  - LIST -> show me | I want | can I see|…

Shallow Semantics

- Terminal substitution
  - Employed by some speech toolkits, e.g. CSLU
- Rules convert terminals in grammar to semantics
  - LIST -> show me | I want | can I see|…
    - e.g. show -> LIST

Shallow Semantics

- Terminal substitution
  - Employed by some speech toolkits, e.g. CSLU
- Rules convert terminals in grammar to semantics
  - LIST -> show me | I want | can I see|…
    - e.g. show -> LIST
    - see -> LIST
    - I -> ε
    - can -> ε
    - * Boston -> Boston

Shallow Semantics

- Terminal substitution
  - Employed by some speech toolkits, e.g. CSLU
- Rules convert terminals in grammar to semantics
  - LIST -> show me | I want | can I see|…
    - e.g. show -> LIST
    - see -> LIST
    - I -> ε
    - can -> ε
    - * Boston -> Boston
- Simple, but…
  - VERY limited, assumes direct correspondence

Semantic Grammars

- Domain-specific semantic analysis

Semantic Grammars

- Domain-specific semantic analysis
- Syntactic structure:
  - Context-free grammars (CFGs) (typically)
    - Can be parsed by standard CFG parsing algorithms
      - e.g. Earley parsers or CKY

Semantic Grammars

- Domain-specific semantic analysis
- Syntactic structure:
  - Context-free grammars (CFGs) (typically)
    - Can be parsed by standard CFG parsing algorithms
      - e.g. Earley parsers or CKY
- Semantic structure:
  - Some designated non-terminals correspond to slots
    - Associate terminal values to corresponding slot

Semantic Grammars

- Domain-specific semantic analysis
- Syntactic structure:
  - Context-free grammars (CFGs) (typically)
    - Can be parsed by standard CFG parsing algorithms
      - e.g. Earley parsers or CKY
- Semantic structure:
  - Some designated non-terminals correspond to slots
    - Associate terminal values to corresponding slot
- Frames can be nested
- Widely used: Phoenix NLU (CU, CMU), vxml grammars

Show me morning flights from Boston to SFO on Tuesday

- LIST -> show me | I want | can I see|…
- DEPARTTIME -> (after| around|before) HOUR| morning | afternoon | evening
- HOUR -> one|two|three…| twelve (am|pm)
- FLIGHTS -> (a) flight|flights
- ORIGIN -> from CITY
- DESTINATION -> to CITY
- CITY -> Boston | San Francisco | Denver | Washington

- SHOW:
- FLIGHTS:
  - ORIGIN:
    - CITY: Boston
    - DATE:
      - DAY-OF-WEEK: Tuesday
    - TIME:
      - PART-OF-DAY: Morning
  - DEST:
    - CITY: San Francisco

Semantic Grammars: Issues

- Issues:

Semantic Grammars: Issues

- Issues:
  - Generally manually constructed
    - Can be expensive, hard to update/maintain

Semantic Grammars: Issues

- Issues:
  - Generally manually constructed
    - Can be expensive, hard to update/maintain
  - Managing ambiguity:
    - Can associate probabilities with parse & analysis
    - Build rules manually, then train probabilities w/data

Semantic Grammars: Issues

- Issues:
  - Generally manually constructed
    - Can be expensive, hard to update/maintain
  - Managing ambiguity:
    - Can associate probabilities with parse & analysis
    - Build rules manually, then train probabilities w/data
  - Domain- and application-specific
    - Hard to port

VoiceXML

Simple VoiceXML Example

- Minimal form:

```
<form>
  <field name="transporttype">
     <prompt>
        Please choose airline, hotel, or rental car.
     </prompt>
     <grammar type="application/x=nuance-gsl">
       [airline hotel "rental car"]
     </grammar>
  </field>
  <block>
    <prompt>
     You have chosen <value expr="transporttype">.
    </prompt>
  </block>
</form>
```

Basic VXML Document

- Main body: <form></form>
  - Sequence of fields: <field></field>

Basic VXML Document

- Main body: <form></form>
  - Sequence of fields: <field></field>
    - Correspond to variable storing user input
      - <field name="transporttype">

Basic VXML Document

- Main body: <form></form>
  - Sequence of fields: <field></field>
    - Correspond to variable storing user input
      - <field name="transporttype">
    - Prompt for user input
      - <prompt> Please choose airline, hotel, or rental car.</prompt>
        - Can include URL for recorded prompt, backs off

Basic VXML Document

- Main body: <form></form>
  - Sequence of fields: <field></field>
    - Correspond to variable storing user input
      - <field name=“transporttype”>

    - Prompt for user input
      - <prompt> Please choose airline, hotel, or rental car.</prompt>
        - Can include URL for recorded prompt, backs off

    - Specify grammar to recognize/interpret user input
      - <grammar>[airline hotel “rental car”]</grammar>

Other Field Elements

- Context-dependent help:
  - <help>Please select activity.</help>

Other Field Elements

- Context-dependent help:
  - <help>Please select activity.</help>

- Action to be performed on input:
  - <filled>
    - <prompt>You have chosen <value exp=“transporttype”>.
    - </prompt></filled>

Control Flow

- Default behavior:
  - Step through elements of form in document order

Control Flow

- Default behavior:
  - Step through elements of form in document order
- Goto allows jump to:
  - Other form: <goto next=“weather.xml”>
  - Other position in form: <goto next=“#departdate”>
-

Control Flow

- Default behavior:
  - Step through elements of form in document order
- Goto allows jump to:
  - Other form: <goto next=“weather.xml”>
  - Other position in form: <goto next=“#departdate”>
- Conditionals:
  - <if cond=“varname==‘air’”>....</if>

Control Flow

- Default behavior:
  - Step through elements of form in document order
- Goto allows jump to:
  - Other form: <goto next=“weather.xml”>
  - Other position in form: <goto next=“#departdate”>
- Conditionals:
  - <if cond=“varname==‘air’”>....</if>
- Guards:
  - Default: Skip field if slot value already entered

General Interaction

- ‘Universals’:
  - Behaviors used by all apps, specify particulars
  - Pick prompts for conditions

General Interaction

- ‘Universals’:
  - Behaviors used by all apps, specify particulars
  - Pick prompts for conditions
- <noinput>:
  - No speech timeout

General Interaction

- ‘Universals’:
  - Behaviors used by all apps, specify particulars
  - Pick prompts for conditions
- <noinput>:
  - No speech timeout
- <nomatch>:
  - Speech, but nothing valid recognized

General Interaction

- ‘Universals’:
  - Behaviors used by all apps, specify particulars
  - Pick prompts for conditions
- <noinput>:
  - No speech timeout
- <nomatch>:
  - Speech, but nothing valid recognized
- <help>:
  - General system help prompt

Complex Interaction

- Preamble, grammar:

```
<noinput>    I'm sorry, I didn't hear you.  <reprompt/>  </noinput>

<nomatch> I'm sorry, I didn't understand that.  <reprompt/> </nomatch>

<form>
   <grammar type="application/x=nuance-gsl">
    <![ CDATA[
    Flight (   ?[
             (i [wanna (want to)] [fly go])
             (i'd like to [fly go])
             ([(i wanna)(i'd like a)] flight)
          ]
          [
            ( [from leaving departing] City:x) {<origin $x>}
            ( [(?going to)(arriving in)] City:x) {<destination $x>}
            ( [from leaving departing] City:x
              [(?going to)(arriving in)] City:y) {<origin $x> <destination $y>}
          ]
          ?please
     )
     City [ [(san francisco) (s f o)] {return( "san francisco, california")}
            [(denver) (d e n)] {return( "denver, colorado")}
            [(seattle) (s t x)] {return( "seattle, washington")}
     ]
     ]]> </grammar>

  <initial name="init">
     <prompt> Welcome to the consultant. What are your travel plans?  </prompt>
  </initial>
```

Mixed Initiative

- With guard defaults

```
<field name="origin">
   <prompt> Which city do you want to leave from?  </prompt>
   <filled>
      <prompt> OK, from <value expr="origin"> </prompt>
   </filled>
</field>
<field name="destination">
   <prompt> And which city do you want to go to?  </prompt>
   <filled>
      <prompt> OK, to <value expr="destination"> </prompt>
   </filled>
</field>
<block>
   <prompt> OK, I have you are departing from  <value expr="origin">
            to  <value expr="destination">.  </prompt>
  send the info to book a flight...
</block>
</form>
```

Complex Interaction

- Preamble, external grammar:

```
<?xml version="1.0"?>
<vxml version = "2.0">

<form id="F1">

  <field name="F_1">
      <grammar src="NameGram.xml"
type="application/grammar-xml" />
      <prompt>
      Please tell me your full name so I can verify you
      </prompt>
  </field>

  <filled mode="all" namelist="F_1">
      <prompt>
        Your name is <value expr="F_1"/>
        <break strength="medium"/>
      </prompt>
    </filled>
</form>
</vxml>
```

Multi-slot Grammar

```
<?xml version= "1.0"?>
 <grammar xml:lang="en-US" root = "TOPLEVEL">
  <rule id="TOPLEVEL" scope="public">
   <item>

  <!-- FIRST NAME RETURN -->

    <item repeat="0-1">
     <ruleref uri="#FIRSTNAME"/>
     <tag>out.firstNameSlot=rules.FIRSTNAME.firstNameSubslot;</tag>
    </item>
  <!-- MIDDLE NAME RETURN -->

    <item repeat="0-1">
     <ruleref uri="#MIDDLENAME"/>
     <tag>out.middleNameSlot=rules.MIDDLENAME.middleNameSubslot;</tag>
    </item>
  <!-- LAST NAME RETURN -->

     <ruleref uri="#LASTNAME"/>
     <tag>out.lastNameSlot=rules.LASTNAME.lastNameSubslot;</tag>
    </item>

 <!-- TOP LEVEL RETURN-->
   <tag> out.F_1= out.firstNameSlot + out.middleNameSlot + out.lastNameSlot; </tag>
  </rule>
```

Multi-slot Grammar II

```
<rule id="FIRSTNAME" scope="public">
  <one-of>
    <item> matt<tag>out.firstNameSubslot="matthew";</tag></item>
    <item> dee <tag> out.firstNameSubslot="dee ";</tag></item>
    <item> jon <tag> out.firstNameSubslot="jon ";</tag></item>
    <item> george <tag>out.firstNameSubslot="george ";</tag></item>
    <item> billy <tag> out.firstNameSubslot="billy ";</tag></item>
  </one-of>
  </rule>

  <rule id="MIDDLENAME" scope="public">
  <one-of>
    <item> bon <tag>out.middleNameSubslot="bon ";</tag></item>
    <item> double ya <tag> out.middleNameSubslot="w ";</tag></item>
    <item> dee <tag> out.middleNameSubslot="dee ";</tag></item>
  </one-of>
  </rule>

  <rule id="LASTNAME" scope="public">
  <one-of>
    <item> henry <tag> out.lastNameSubslot="henry "; </tag></item>
    <item> ramone <tag> out.lastNameSubslot="dee ";  </tag></item>
    <item> jovi <tag> out.lastNameSubslot="jovi ";  </tag></item>
    <item> bush <tag> out.lastNameSubslot=""bush ";  </tag></item>
    <item> williams <tag> out.lastNameSubslot="williams "; </tag></item>
  </one-of>
  </rule>

</grammar>
```

Augmenting VoiceXML

- Don't write XML directly
  - Use php or other system to generate VoiceXML
    - Used in 'Let's Go Dude' bus info system

Augmenting VoiceXML

- Don't write XML directly
  - Use php or other system to generate VoiceXML
    - Used in 'Let's Go Dude' bus info system
- Pass input to other web services
  - i.e. to RESTful services

Augmenting VoiceXML

- Don’t write XML directly
  - Use php or other system to generate VoiceXML
    - Used in ‘Let’s Go Dude’ bus info system
- Pass input to other web services
  - i.e. to RESTful services
- Access web-based audio for prompts

Advanced Dialog Models

- Information State Models
- Statistical Dialog Models

Information State Models

- Challenges in dialog management
  - Difficult to evaluate
    - Hard to isolate from implementations
    - Integration inhibits portability
  - Wide gap between theoretical and practical models
    - Theoretical: logic-based, BDI, plan-based, attention/intention
    - Practical: mostly finite-state or frame-based
    - Even if theory-consistent, many possible implementations
  - Implementation dominates

Why the Gap?

- Theories hard to implement
  - Underspecified
  - Overly complex, intractable
    - e.g. inferring all user intents
- Theories hard to compare
  - Employ diff't basic units
  - Disagree on basic structure

- Implementation is hard
  - Driven by technical limitations, optimizations
  - Driven by specific tasks
- Most approaches simplistic
  - Not focused on model details

Information State Approach

- Approach to formalizing dialog theories
- Toolkit to support implementation (Trindikit)
  - Designed to abstract out dialog theory components
- Example systems & related tools

Information State Theory of Dialog

- Components:
  - Informational components:
    - Common context and internal models (belief, goals, etc)
  - Formal representations:

  - Dialog moves: recognition and generation
    - Trigger state updates
  - Update rules:
    - Describe update given current state, moves, etc
  - Update strategy:
    - Method for selecting rules if more than one applies
      - Simple or complex

Example Dialog

- S: Welcome to the travel agency!
- U: flights to paris
- S: Okay, you want to know about price. A flight. To Paris. Let's see. What city do you want to go from?

$$
\left[\begin{array}{lll}
\text{PRIVATE} & = & \left[\begin{array}{lll}
\text{BEL} & = & \{\} \\
\text{AGENDA} & = & \langle\rangle \\
\text{PLAN} & = & \left\langle \begin{array}{l} \text{findout}(?\text{x.dept-month}(\text{x})), \\ \text{findout}(?\text{x.dept-day}(\text{x})), \\ \text{findout}(?\text{x.class}(\text{x})), \\ \text{consultDB}(?\text{x.price}(\text{x})) \end{array} \right\rangle \\
\text{TMP} & = & \ldots \\
\text{NIM} & = & \ldots
\end{array}\right] \\
\text{SHARED} & = & \left[\begin{array}{lll}
\text{COM} & = & \{\text{dest-city}(\text{paris}),\text{how}(\text{plane})\} \\
\text{ISSUES} & = & \langle ?x.\text{dept-city}(x),\ ?x.\text{price}(x)\ \rangle \\
\text{QUD} & = & \langle ?x.\text{dept-city}(x)\rangle \\
\text{PU} & = & \ldots \\
\text{LU} & = & \langle \text{ask}(\text{sys},\ ?x.\text{dept-city}(x)),\ \ldots\rangle
\end{array}\right]
\end{array}\right]
$$

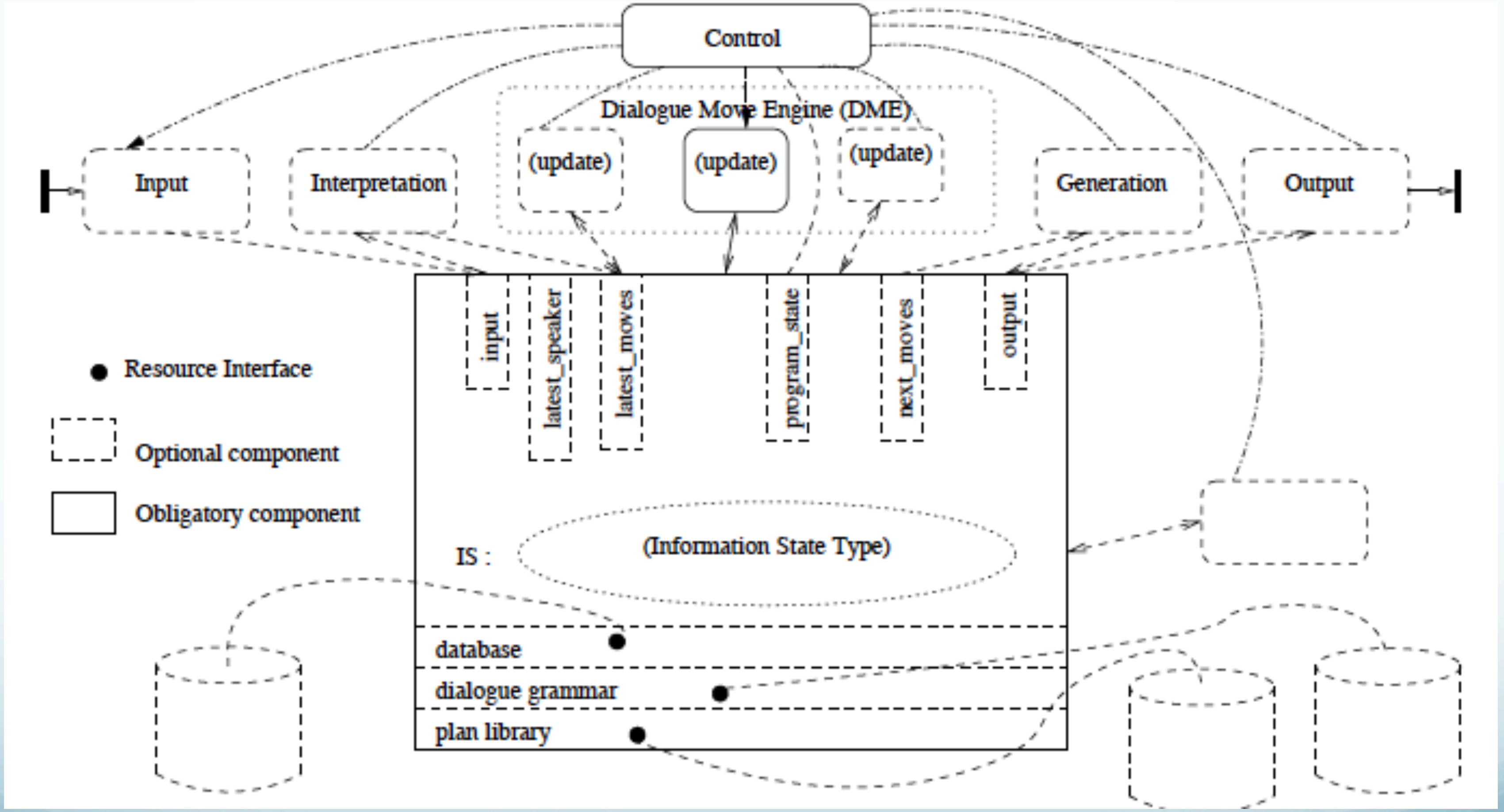
Example Update Rule

$$
\begin{array}{ll}
\textsc{u-rule:} & \textbf{accommodateQuestion}(Q, A) \\
\textsc{pre:} & \left\{ \begin{array}{l} \text{in}(\textsc{shared.lu},\ \text{answer}(\text{usr}, A)), \\ \text{in}(\textsc{private.plan},\ \text{findout}(Q)) \\ \text{domain} :: \text{relevant}(\ A, Q\ ) \end{array} \right. \\
\textsc{eff:} & \left\{ \begin{array}{l} \text{del}(\textsc{private.plan},\ \text{findout}(Q)) \\ \text{push}(\textsc{shared.qud},\ Q) \end{array} \right.
\end{array}
$$

Implementation

- Dialog Move Engine (DME)
  - Implements an information state dialog model
  - Observes/interprets moves
  - Updates information state based on moves
  - Generates new moves consistent with state
- Full system requires: DME+
  - Input/output components
  - Interpretation: determine what move made
  - Generation: produce output for ‘next move’
  - Control system to manage components

Trindikit Architecture



Multi-level Architecture

- Separates types of design expertise, knowledge
- Domain & language resources → Domain system
- Dialog theory → Abstract DME
  - IS, update rules, etc
- Software Engineering → Trindikit
  - basic types, control

Statistical Dialog Management

New Idea: Modeling a dialogue system as a probabilistic agent

- A conversational agent can be characterized by:
  - The current knowledge of the system
    - A set of states S the agent can be in
  - a set of actions A the agent can take
  - A goal G, which implies
    - A success metric that tells us how well the agent achieved its goal
    - A way of using this metric to create a strategy or policy π for what action to take in any particular state.

What do we mean by actions A and policies π?

- Kinds of decisions a conversational agent needs to make:
  - When should I ground/confirm/reject/ask for clarification on what the user just said?
  - When should I ask a directive prompt, when an open prompt?
  - When should I use user, system, or mixed initiative?

A threshold is a human-designed policy!

- Could we learn what the right action is
  - Rejection
  - Explicit confirmation
  - Implicit confirmation
  - No confirmation
- By learning a policy which,
  - given various information about the current state,
  - dynamically chooses the action which maximizes dialogue success

Another strategy decision

- Open versus directive prompts
- When to do mixed initiative

- How we do this optimization?
- Markov Decision Processes

Review: Open vs. Directive Prompts

- Open prompt
  - System gives user very few constraints
  - User can respond how they please:
  - "How may I help you?" "How may I direct your call?"
- Directive prompt
  - Explicit instructs user how to respond
  - "Say yes if you accept the call; otherwise, say no"

Review: Restrictive vs. Non-restrictive gramamrs

- Restrictive grammar
  - Language model which strongly constrains the ASR system, based on dialogue state
- Non-restrictive grammar
  - Open language model which is not restricted to a particular dialogue state

Kinds of Initiative

- How do I decide which of these initiatives to use at each point in the dialogue?

Grammar	Open Prompt	Directive Prompt
Restrictive	*Doesn't make sense*	System Initiative
Non-restrictive	User Initiative	Mixed Initiative

Modeling a dialogue system as a probabilistic agent

- A conversational agent can be characterized by:
  - The current knowledge of the system
    - A set of states S the agent can be in

Modeling a dialogue system as a probabilistic agent

- A conversational agent can be characterized by:
  - The current knowledge of the system
    - A set of states S the agent can be in
  - a set of actions A the agent can take

Modeling a dialogue system as a probabilistic agent

- A conversational agent can be characterized by:
  - The current knowledge of the system
    - A set of states S the agent can be in
  - a set of actions A the agent can take
  - A goal G, which implies
    - A success metric that tells us how well the agent achieved its goal
    - A way of using this metric to create a strategy or policy π for what action to take in any particular state.

Goals are not enough

- Goal: user satisfaction
- OK, that's all very well, but

Goals are not enough

- Goal: user satisfaction
- OK, that's all very well, but
  - Many things influence user satisfaction
  - We don't know user satisfaction til after the dialogue is done
  - How do we know, state by state and action by action, what the agent should do?
- We need a more helpful metric that can apply to each state

Utility

- A utility function
  - maps a state or state sequence
  - onto a real number
  - describing the goodness of that state
  - I.e. the resulting "happiness" of the agent

Utility

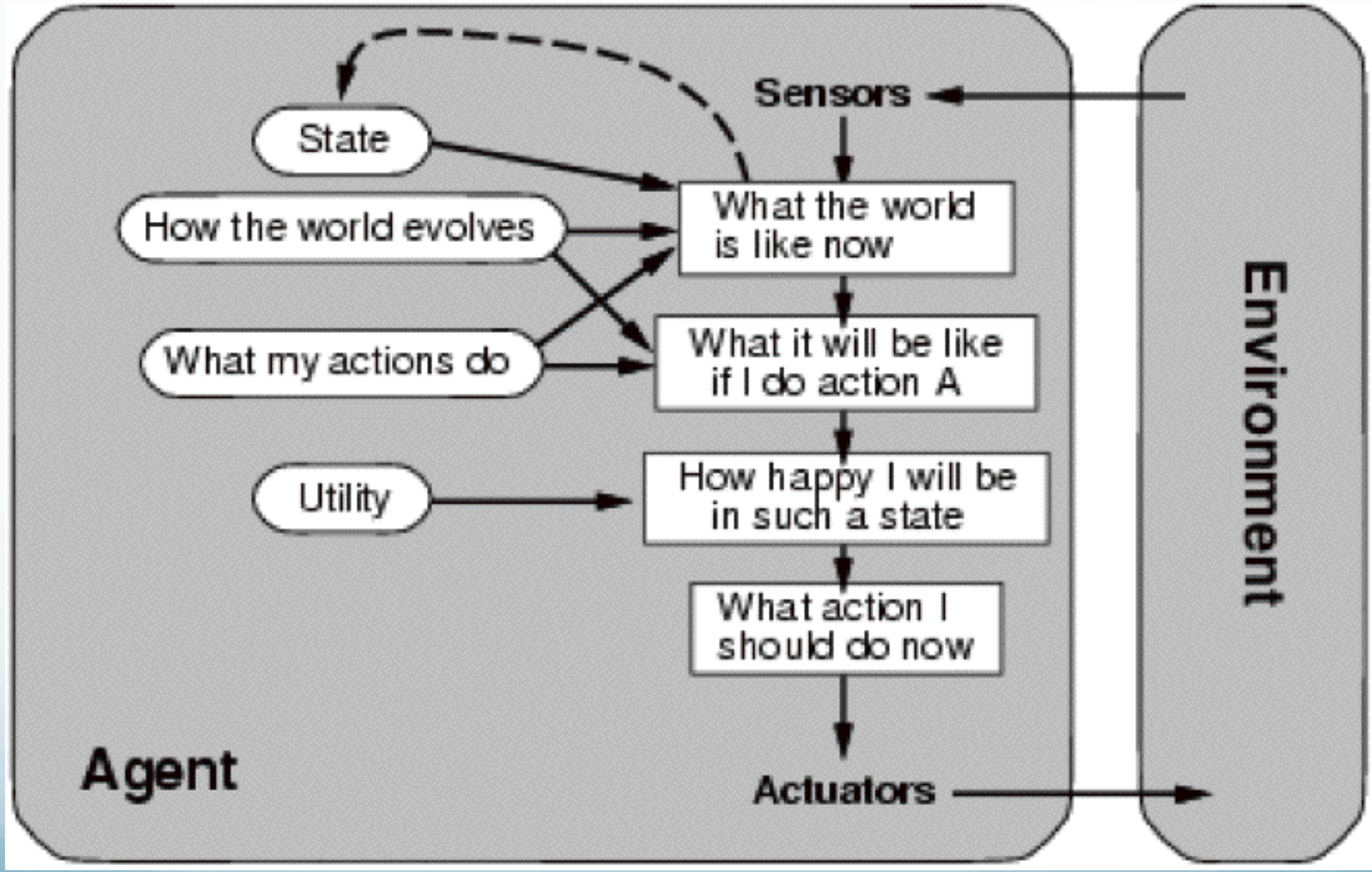
- A utility function
  - maps a state or state sequence
  - onto a real number
  - describing the goodness of that state
  - I.e. the resulting “happiness” of the agent
- Principle of Maximum Expected Utility:
  - A rational agent should choose an action that maximizes the agent’s expected utility

Maximum Expected Utility

- Principle of Maximum Expected Utility:
  - A rational agent should choose an action that maximizes the agent's expected utility
- Action A has possible outcome states $Result_i(A)$
- E: agent's evidence about current state of world
- Before doing A, agent estimates prob of each outcome
  - $P(Result_i(A)|Do(A),E)$
- Thus can compute expected utility:

$$EU(A|E) = \sum_i P(Result_i(A)|Do(A),E)U(Result_i(A))$$

Utility (Russell and Norvig)



Markov Decision Processes

- Or MDP
- Characterized by:
  - a set of states S an agent can be in
  - a set of actions A the agent can take
  - A reward r(a,s) that the agent receives for taking an action in a state

A brief tutorial example

- Levin et al (2000)
- A Day-and-Month dialogue system
- Goal: fill in a two-slot frame:
  - Month: November
  - Day: 12th
- Via the shortest possible interaction with user

What is a state?

- In principle, MDP state could include any possible information about dialogue
  - Complete dialogue history so far

What is a state?

- In principle, MDP state could include any possible information about dialogue
  - Complete dialogue history so far
- Usually use a much more limited set
  - Values of slots in current frame
  - Most recent question asked to user
  - Users most recent answer
  - ASR confidence
  - etc

State in the Day-and-Month example

- Values of the two slots day and month.
- Total:
  - 2 special initial states s_i and s_f.
  - 365 states with a day and month
  - 1 state for leap year
  - 12 states with a month but no day
  - 31 states with a day but no month
  - 411 total states

Actions in MDP models of dialogue

Actions in MDP models of dialogue

- Speech acts!
  - Ask a question
  - Explicit confirmation
  - Rejection
  - Give the user some database information
  - Tell the user their choices
- Do a database query

Actions in the Day-and-Month example

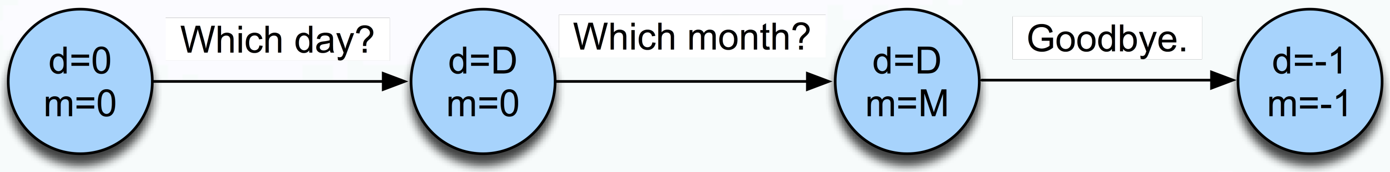
- a_d: a question asking for the day
- a_m: a question asking for the month
- a_{dm}: a question asking for the day+month
- a_f: a final action submitting the form and terminating the dialogue

A simple reward function

- For this example, let's use a cost function
- A cost function for entire dialogue
- Let
  - N_i=number of interactions (duration of dialogue)
  - N_e=number of errors in the obtained values (0-2)
  - N_f=expected distance from goal
    - (0 for complete date, 1 if either data or month are missing, 2 if both missing)
- Then (weighted) cost is:
- $C = w_i \times N_i + w_e \times N_e + w_f \times N_f$

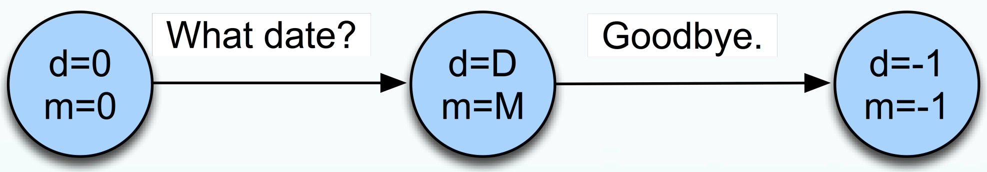
2 possible policies

Policy 1 (directive)



$c_1 = -3w_i + 2p_d w_e$

Policy 2 (open)



$c_2 = -2w_i + 2p_o w_e$

P_d=probability of error in directive prompt

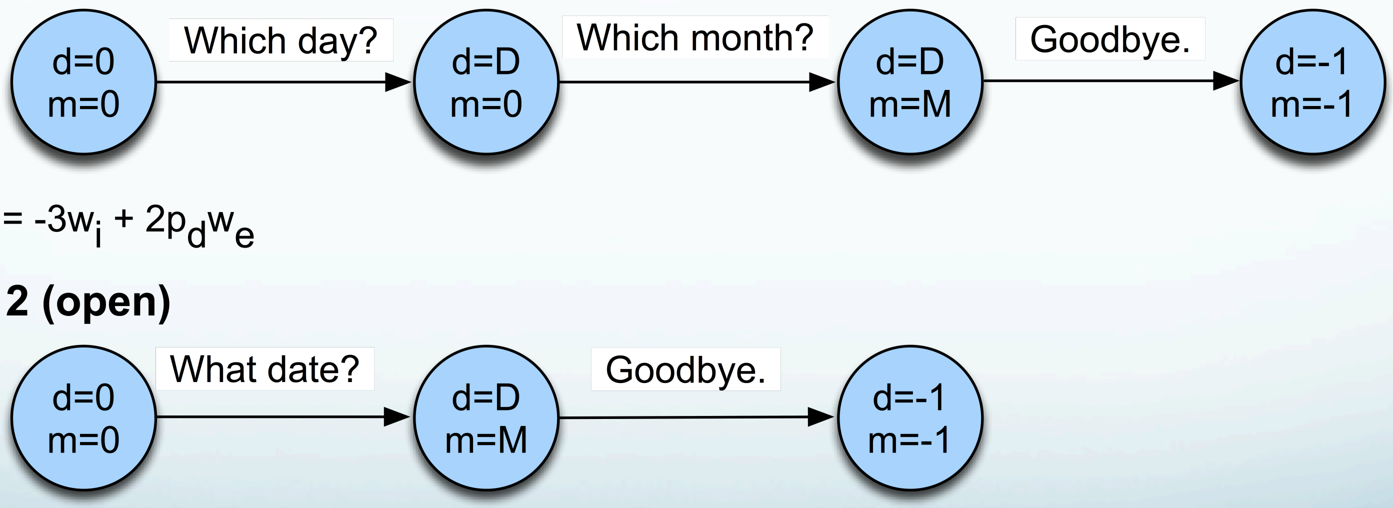
P_o=probability of error in open prompt

2 possible policies

Strategy 1 is better than strategy 2 when
improved error rate justifies longer interaction:

$$p_o - p_d > \frac{w_i}{2w_e}$$

Policy 1 (directive)



$c_1 = -3w_i + 2p_d w_e$

Policy 2 (open)

$c_2 = -2w_i + 2p_o w_e$

That was an easy optimization

- Only two actions, only tiny # of policies
- In general, number of actions, states, policies is quite large
- So finding optimal policy π^* is harder
- We need reinforcement learning
- Back to MDPs:

MDP

- We can think of a dialogue as a trajectory in state space

$$s_1 \rightarrow_{a1,r1} s_2 \rightarrow_{a2,r2} s_3 \rightarrow_{a3,r3} \cdots$$

- The best policy π* is the one with the greatest expected reward over all trajectories
- How to compute a reward for a state sequence?

Reward for a state sequence

- One common approach: discounted rewards
- Cumulative reward Q of a sequence is discounted sum of utilities of individual states

$$Q([s_0, a_0, s_1, a_1, s_2, a_2 \cdots]) = R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots,$$

- Makes agent care more about current than future rewards; the more future a reward, the more discounted its value

The Markov assumption

- MDP assumes that state transitions are Markovian

$$P(s_{t+1} \mid s_t, s_{t-1}, \ldots, s_o, a_t, a_{t-1}, \ldots, a_o) = P_T(s_{t+1} \mid s_t, a_t)$$

Expected reward for an action

- Expected cumulative reward Q(s,a) for taking a particular action from a particular state can be computed by Bellman equation:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

- Expected cumulative reward for a given state/action pair is:
  - immediate reward for current state
  - + expected discounted utility of all possible next states s'
  - Weighted by probability of moving to that state s'
  - And assuming once there we take optimal action a'

What we need for Bellman equation

- A model of p(s' |s,a)
- Estimate of R(s,a)
- How to get these?

What we need for Bellman equation

- A model of p(s' |s,a)
- Estimate of R(s,a)
- How to get these?
- If we had labeled training data
  - P(s' |s,a) = C(s,s' ,a)/C(s,a)

What we need for Bellman equation

- A model of p(s’|s,a)
- Estimate of R(s,a)
- How to get these?
- If we had labeled training data
  - P(s’|s,a) = C(s,s’,a)/C(s,a)
- If we knew the final reward for whole dialogue R(s1,a1,s2,a2,…,sn)
- Given these parameters, can use value iteration algorithm to learn Q values (pushing back reward values over state sequences) and hence best policy

Final reward

- What is the final reward for whole dialogue R(s1,a1,s2,a2,…,sn)?
- This is what our automatic evaluation metric PARADISE computes!
- The general goodness of a whole dialogue!!!!!

How to estimate p(s'|s,a) without labeled data

How to estimate p(s' |s,a) without labeled data

- Have random conversations with real people
  - Carefully hand-tune small number of states and policies
  - Then can build a dialogue system which explores state space by generating a few hundred random conversations with real humans
  - Set probabilities from this corpus

How to estimate p(s' |s,a) without labeled data

- Have random conversations with real people
  - Carefully hand-tune small number of states and policies
  - Then can build a dialogue system which explores state space by generating a few hundred random conversations with real humans
  - Set probabilities from this corpus
- Have random conversations with simulated people
  - Now you can have millions of conversations with simulated people
  - So you can have a slightly larger state space

An example

- Singh, S., D. Litman, M. Kearns, and M. Walker. 2002. Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System. Journal of AI Research.

- NJFun system, people asked questions about recreational activities in New Jersey
- Idea of paper: use reinforcement learning to make a small set of optimal policy decisions

Very small # of states and acts

- **States**: specified by values of 8 features
  - Which slot in frame is being worked on (1-4)
  - ASR confidence value (0-5)
  - How many times a current slot question had been asked
  - Restrictive vs. non-restrictive grammar
  - Result: 62 states
- **Actions**: each state only 2 possible actions
  - Asking questions: System versus user initiative
  - Receiving answers: explicit versus no confirmation.

Ran system with real users

- 311 conversations
- Simple binary reward function
  - 1 if competed task (finding museums, theater, winetasting in NJ area)
  - 0 if not
- System learned good dialogue strategy: Roughly
  - Start with user initiative
  - Backoff to mixed or system initiative when re-asking for an attribute
  - Confirm only a lower confidence values

State of the art

- Only a few such systems
  - From (former) ATT Laboratories researchers, now dispersed
  - And Cambridge UK lab
- Hot topics:
  - Partially observable MDPs (POMDPs)
  - We don't REALLY know the user's state (we only know what we THOUGHT the user said)
  - So need to take actions based on our BELIEF , I.e. a probability distribution over states rather than the "true state"

Summary

- Utility-based conversational agents
  - Policy/strategy for:
    - Confirmation
    - Rejection
    - Open/directive prompts
    - Initiative
    - +?????
  - MDP
  - POMDP

Learning Probabilistic Slot Filling

- Goal: Use machine learning to map from recognizer strings to semantic slots and fillers

Learning Probabilistic Slot Filling

- Goal: Use machine learning to map from recognizer strings to semantic slots and fillers

- Motivation:
  - Improve robustness – fail-soft
  - Improve ambiguity handling – probabilities
  - Improve adaptation – train for new domains, apps

Learning Probabilistic Slot Filling

- Goal: Use machine learning to map from recognizer strings to semantic slots and fillers
- Motivation:
  - Improve robustness – fail-soft
  - Improve ambiguity handling – probabilities
  - Improve adaptation – train for new domains, apps
- Many alternative classifier models
  - HMM-based, MaxEnt-based

HMM-Based Slot Filling

- Find best concept sequence C given words W

HMM-Based Slot Filling

- Find best concept sequence C given words W
- C^{*}= argmax P(C|W)
-

HMM-Based Slot Filling

- Find best concept sequence C given words W
- $C^* = \text{argmax } P(C|W)$
- $= \text{argmax } P(W|C)P(C)/P(W)$
-

HMM-Based Slot Filling

- Find best concept sequence C given words W
- $C^{*} = \text{argmax } P(C|W)$
- $= \text{argmax } P(W|C)P(C)/P(W)$
- $= \text{argmax } P(W|C)P(C)$

HMM-Based Slot Filling

- Find best concept sequence C given words W
- C^{*}= argmax P(C|W)
- = argmax P(W|C)P(C)/P(W)
- = argmax P(W|C)P(C)
- Assume limited M-concept history, N-gram words
  - = $$\prod_{i=2}^{N} P(w_i \mid w_{i-1} \ldots w_{i-N+1}, c_i) \prod_{i=2}^{N} P(c_i \mid c_{i-1} \ldots c_{i-M+1})$$

Probabilistic Slot Filling

- Example HMM