UiO : University of Oslo

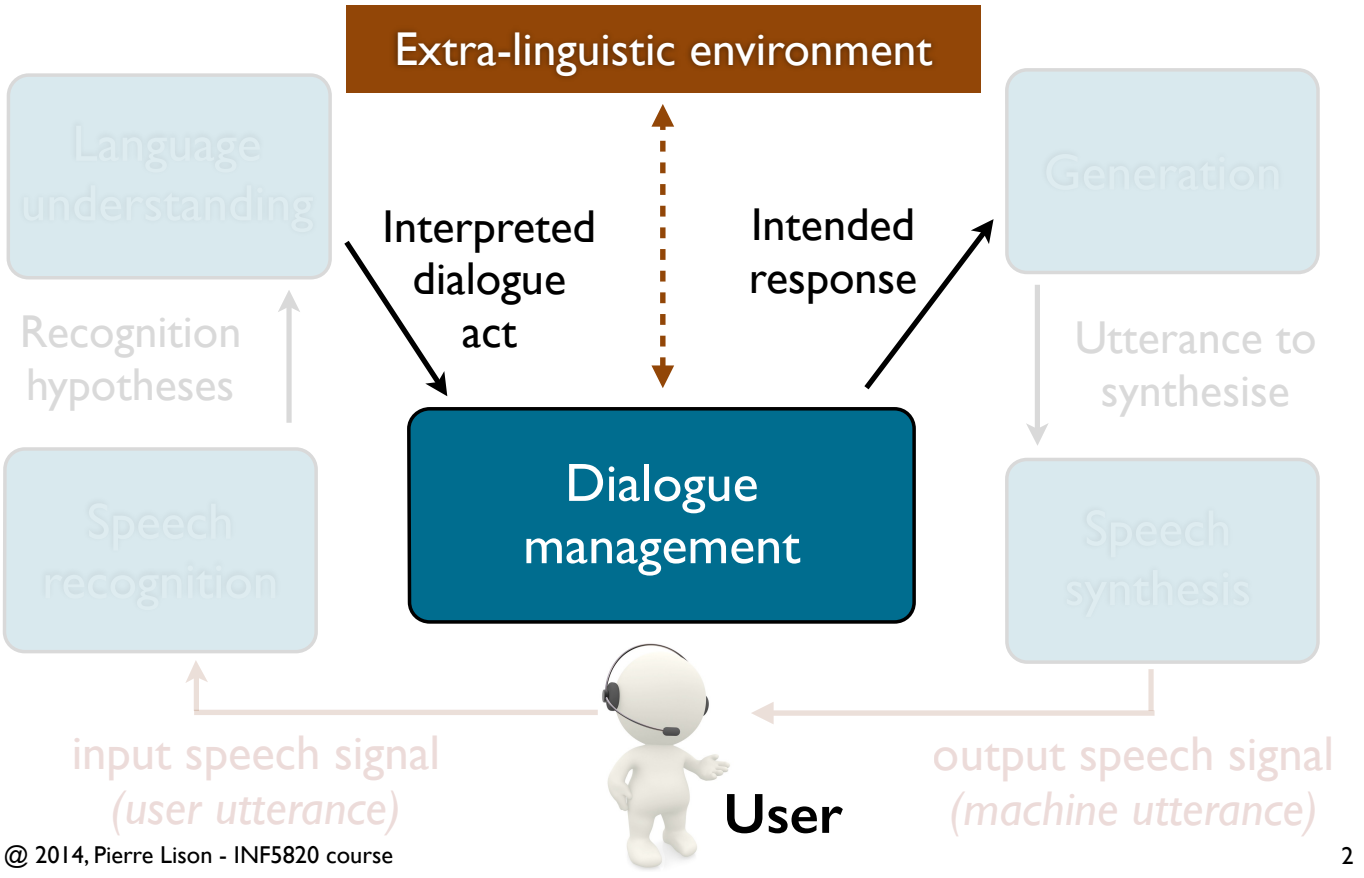
INF5820: Dialogue management

Pierre Lison,
Language Technology Group (LTG)
Department of Informatics

Fall 2014

UiO : University of Oslo

Dialogue management



Outline

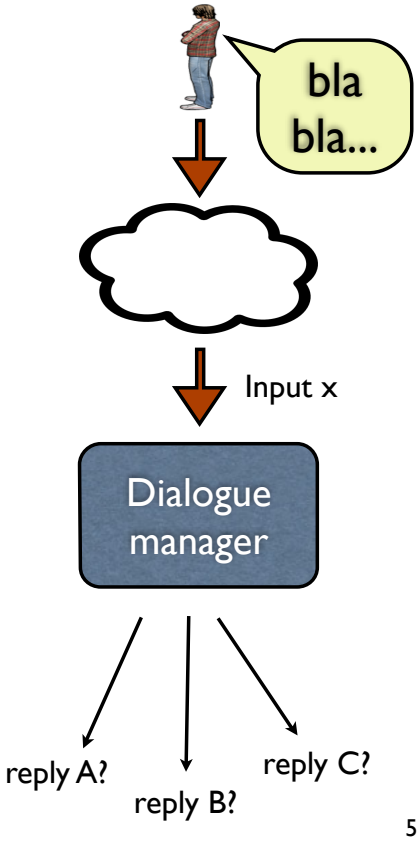
- What is dialogue management?
- Approaches
- Evaluation metrics
- Summary

Outline

- **What is dialogue management?**
- Approaches
- Evaluation metrics
- Summary

Dialogue management

- Dialogue management is about **decision-making**
  - i.e. what should the system decide to *say* or *do* at a given point
  - decision-making *under uncertainty*, since the communication channel is noisy
  - The action set can include both linguistic and non-linguistic actions
  - The same holds for the observation set (e.g. multimodal interaction)



Two core challenges

Spoken dialogue is …

Complex

- The "right" thing to do/say depends on a multitude of contextual factors
- Both linguistic *and* extra-linguistic factors

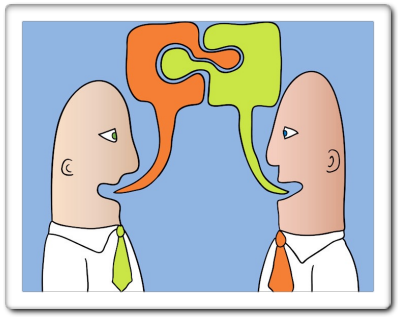
Uncertain

- Pervasiveness of noise, errors and ambiguity (at all processing levels)
- Numerous sources of variability: each dialogue is different!

Dialogue management as a problem of *sequential planning under uncertainty*

Dialogue management

- The dialogue manager is controlling the *flow* of the interaction
- Conversational skills to emulate:
  - Interpret utterances *contextually*;
  - Manage *turn-taking*;
  - Fulfill conversational *obligations* & social *conventions*;
  - *Plan* multi-utterance responses;
  - Manage the system *uncertainty*

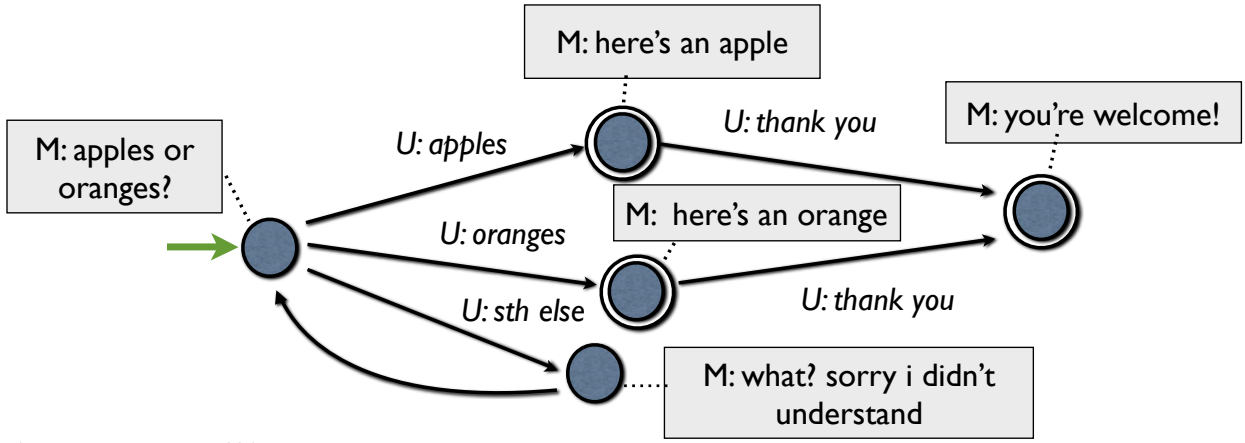


Outline

- What is dialogue management?
- **Approaches**
  - **Finite-state automata**
  - **Frame-based and logical systems**
  - **Statistical techniques**
- Evaluation metrics
- Summary

Finite-state automata

- The simplest approach is to encode dialogue strategies as **finite-state automata**
  - the nodes represent *machine actions*
  - and the edges possible (mutually exclusive) *user responses*

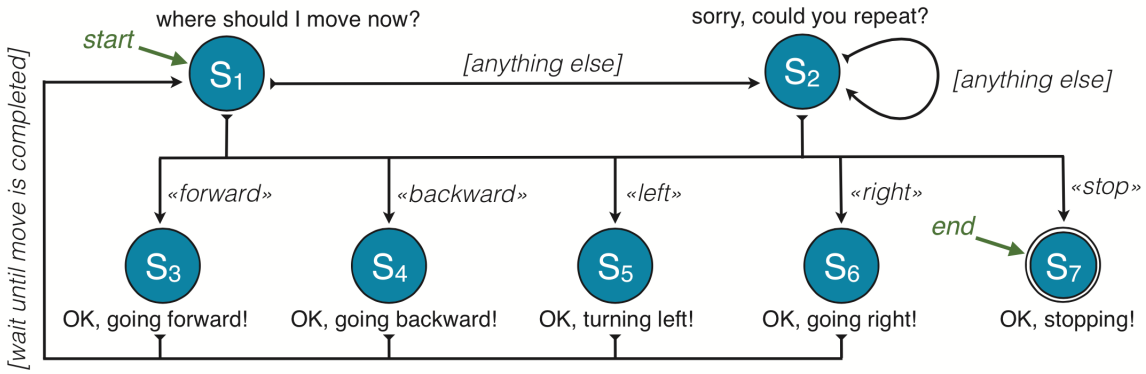


Finite-state automata

- Formalisation of a finite state automata:
  - Finite, non-empty set S of (atomic) states, each associated with a specific machine action.
  - A finite, non-empty set Σ of possible user inputs accepted by the automaton
  - A (partial) function $\delta : S \times \Sigma \rightarrow S$ defining the transition between states
  - An initial state $s_0 \in S$
  - A set of final states $F \subset S$

Finite-state automata

- The transitions can relate to other signals than user inputs (for instance, external events)
- And can also express complex conditions (confidence thresholds, pattern matching, etc.)



Finite-state automata

Advantages	Limitations
• Easy to design • Fast, efficient • *Predictable* system behaviour (both for the user and for the system designer)	• Only allows for *scripted* interactions - not true conversation • No principled account of uncertainties • Difficult to scale to complex domains with many variables and alternative inputs

Frame-based managers

- The interaction flow can be made slightly more flexible in *frame-based systems*
- The state is represented as a frame with slots, which have to be filled by the user's answers

Slot	Question
ORIGIN CITY	«From what city are you leaving?»
DESTINATION CITY	«Where are you going?»
DEPARTURE TIME	«When would you like to leave?»
ARRIVAL TIME	«When do you want to arrive?»

Frame-based managers

- The user can answer the system's question... but can also give some other information
  - «I want to leave from Oslo before 9:00 AM»
- The system recognizes the extra information from the user and fills the appropriate slots
- **VoiceXML**: Voice-extensible Markup Language
  - Standard markup language for basic slot-filling systems
  - Allows mixed initiative

VoiceXML

```
<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>
```

(see Martin & Jurafsky, section 24.3 for more complex examples)

Logic-based reasoning

- Difficult to capture complex interaction styles with finite-state automata or frames
  - Crude notion of a *dialogue state*
  - Crude notion of a *dialogue state transition*: only a few, «hard» transitions are possible for each node
- Possible solution: use richer (more expressive) representations of the state
  - These representations will then allows us to perform more sophisticated forms of reasoning

Logic-based reasoning

- «*Information-state update*» (ISU) is an example of approach based on a rich state representation
  - Encodes the mental states, beliefs and intentions of the speakers, the common ground, dialogue context
- This state is read/written by two types of rules:
  - *Update rules* modify the current state upon the observation of new user dialogue move
  - *Action selection rules* then select the system action based on the information present in this updated state

[S. Larsson and D. R. Traum (2000), «Information state and dialogue management in the TRINDI dialogue move engine toolkit» in *Natural Language Engineering*]

Logic-based reasoning

- One can also use classical planning:
  - Dialogue management viewed as *planning* problem: the system seeks to achieve a long-term goal via a sequence of basic actions
  - Dialogue understanding viewed as *plan recognition* (or *intention recognition*): the system must try to infer the hidden intention behind the conversational behaviour of the user
  - Typically rely on rich models of the task domain, which must be spelled out by the system designer

[R. Thomason & M. Stone (2006), «Enlightened update: A computational architecture for presupposition and other pragmatic phenomena»]

Logic-based reasoning

Advantages	Limitations
• Rich representation of the dialogue state that can capture the *relational structure* of the domain • More powerful reasoning tools for interpretation and decision • Can perform long-term planning	• No account of uncertainty • Requires detailed descriptions of the dialogue domain • More difficult to design (logical abstractions)

Interaction style

- Rigid, repetitive structure of the interaction
- Irritating confirmations & acknowledgements
- No user or context adaptivity

"Saturday night live" sketch comedy, 2005

Statistical techniques

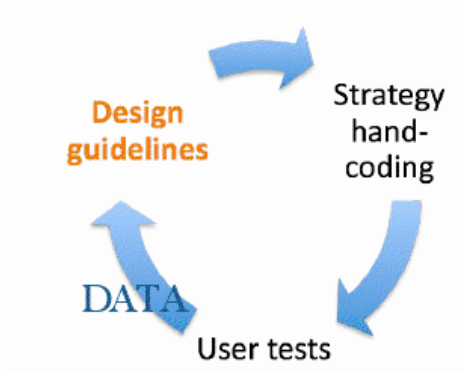
- The approaches presented so far suffer from a number of limitations:
  - Difficult to predict the user behaviour in advance
  - They essentially ignore all the *uncertainties* appearing through the dialogue (ASR errors, ambiguities, etc.)
  - Unable to *learn* or adapt to the users or the environment (leading to rigid/repetitive behaviour)
  - Limited to one single goal... but real interactions are trade-offs between *various competing objectives*

Statistical techniques

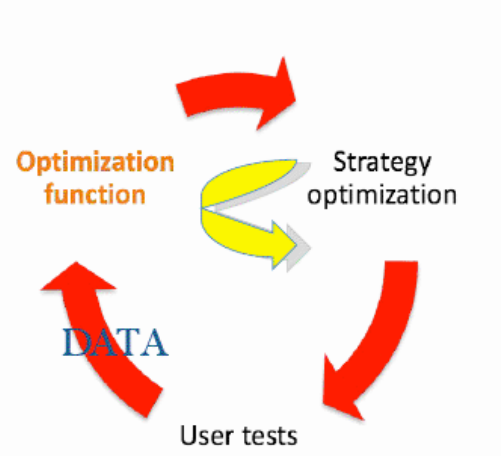
- *Solution*: perform automatic optimisation of the «dialogue policies» from experience:
  - Often based on *reinforcement learning* techniques
  - Experience: interactions with real or simulated users
- General procedure:
  - The dialogue manager starts with a «dumb» dialogue policy
  - It interacts with users, and receives a (positive/negative) feedback
  - It can then corrects his policy based on this feedback
  - This process is repeated until the policy is fully optimised

Statistical techniques

Conventional software life cycle



Automatic strategy optimisation



Design by "Best practices"

(Paek 2007)

Automatic design by optimization function

(= “programming by reward”)

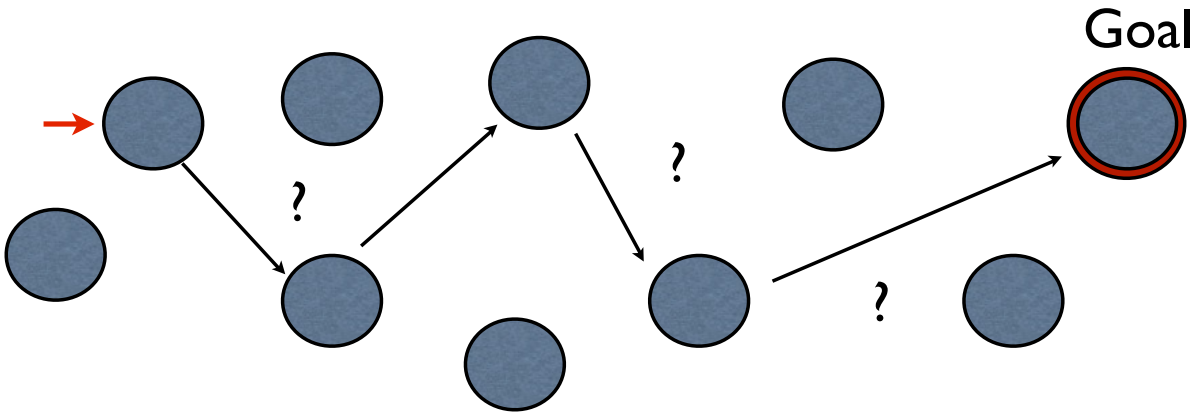
[slide borrowed from O. Lemon]

Statistical techniques

- Dialogue management is again viewed as a **planning/ control** problem:
  - Agent must control its *actions*
  - To reach a long-term *goal*
  - In an uncertain *environment*
  - Where there are many possible *paths* to the goal
  - ... and complex *trade-offs* need to be determined
- But this time, the planning problem will include *multiple goals*, will be performed *under uncertainty*, and will be automatically *learned* from the agent experience

Statistical techniques

- Planning problems are generally defined with three components:
  - A **state space** (the set of all possible states)
  - An **action space** (the set of all possible actions)
  - The **goals** for the task (encoded via e.g. rewards)



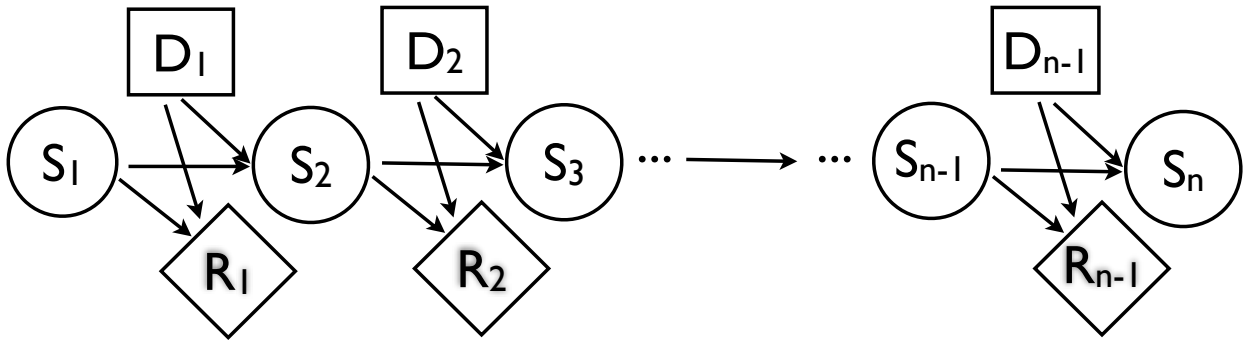
Statistical techniques

- Most tasks have to encode trade-offs between various, competing objectives
  - E.g. a flight booking system must ensure it is booking the right ticket
  - But it must do so with the fewest number of requests
- This is typically encoded via **rewards** (utilities) associated to particular state/action pairs

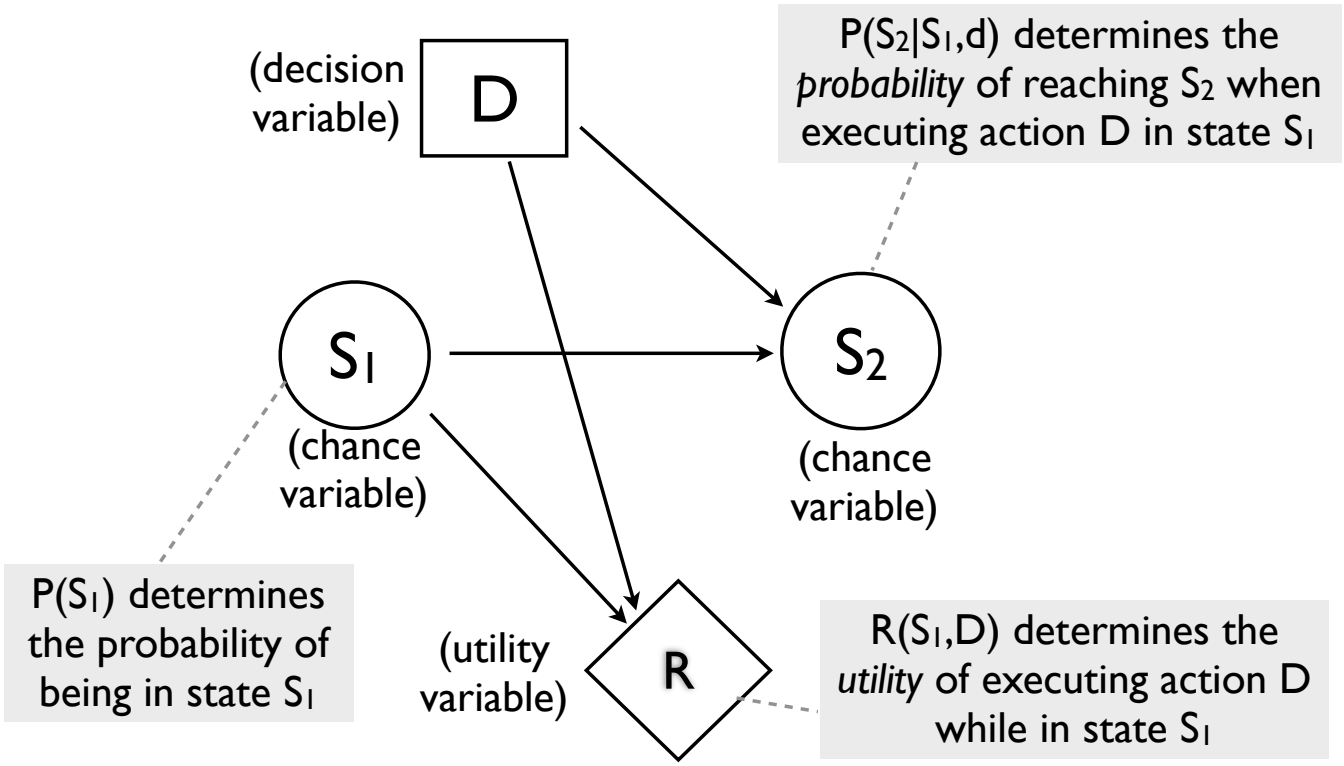
State	Action	Reward
User wants to book ticket *x*	Booking x	+10
User wants to book ticket *x*	Booking y ≠ x	−30
User wants to book ticket *x*	Clarification request	−1

Markov Decision Processes (MDPs)

- We can define these ideas more precisely using a formalism called **Markov Decision Processes** (MDPs)
- Markov Decision Processes are an extension of Markov Chains where the agent *selects an action* at each state
  - This action will then modify the state space
  - And will yield a particular reward for the agent



Graphical notation



Markov Decision Processes (MDPs)

- A MDP is defined as a tuple **<S,A,T,R>**, where:
  - **S** is the *state space* (space of possible states in the domain)
  - **A** is the *action space* (possible actions for the agent)
  - **T** is the *transition function*, defined as $T(s, a, s') = P(s'|s, a)$. It is the probability of arriving to state **s'** after executing action **a** in state **s**.
  - **R** is the *reward function*, defined as $R : S \times A \rightarrow R$. It is a real number encoding the utility for the agent to perform action **a** while in state **s**.

Expected cumulative reward

- In an MDP, the agent seeks to maximise its *expected cumulative reward* **Q(s,a)**

The agent must try to predict future inputs/rewards

The rewards accumulate over time

- How much worth is a reward expected at time (t+i) compared to one received right now?
  - We usually include a *discount factor* γ capturing this balance
  - Related to the problem of *delayed gratification* in psychology

Expected cumulative reward

- For a given state-action sequence s_0, a_0, s_1, a_1... s_n, a_n, the expected cumulative reward Q is:

$$Q([s_0, a_0, s_1, a_1, ...s_0, a_n])$$
$$= R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + ...\gamma^n R(s_n, a_n)$$
$$= \sum_{t=0}^{n} \gamma^t R(s_t, a_t)$$

Immediate reward of executing a_t in state s_t

Discount factor ($0 \leq \gamma \leq 1$)

Bellman equation

- The *Bellman equation* tells us that we can write the expected cumulative reward Q in a recursive fashion:

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

Notice that we are estimating the Q-values based on... our estimation of the Q-values (can be used to iteratively refine these estimates until convergence)

[R. Bellman (1957): «*Dynamic Programming*»]

MDP policy

- Given a MDP, we want to find a dialogue policy telling us which action to execute in a given state
- A dialogue policy is a *mapping* $\pi: S \rightarrow A$ from states to actions
- An *optimal* dialogue policy π^* is a policy that always outputs the action yielding the maximum expected cumulative reward:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$$

Reinforcement learning

- Reinforcement learning algorithms can help us determine these Q values
- They work by iteratively refining their estimate of the Q values
  - The agent is acting in the environment, and observe both the state sequence and the rewards it gets
  - This operation is repeated until convergence

[R. Sutton & A. Barto (1998): «*Reinforcement Learning: An Introduction*»]

Partially observable MDPs

- In an MDP, we assume the current state is fully observable
  - E.g. no uncertainty about the real state of the interaction
  - Often not a reasonable assumption in spoken dialogue!
- We can extend the MDP formalism to handle *partial observability*
  - Leads to a *Partially Observable Markov Decision Process (POMDPs)*
  - In this setting, the current state is uncertain, and we only have a probability distribution P(s) over possible current states

Partially observable MDPs

- POMDPs are defined as a tuple <S,A,O,T,Ω,R>, where:
  - S,A,T and R are defined similarly to MDPs
  - O is the observation space, i.e. the set of input observations which can be captured by the agent
  - Ω is the observation function, defined as $\Omega(z,a,s') = P(z|a,s')$. It defines the probability of observing **z** after executing action **a** when the true (hidden) state of the world is **s'**

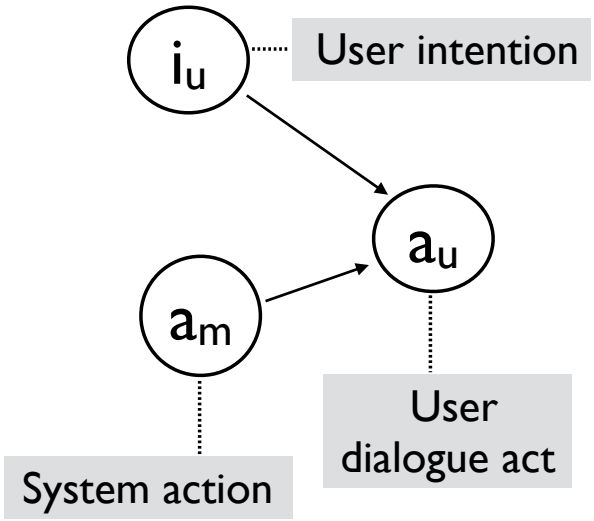
POMDPs for dialogue

- *State space*: set of possible states of the interaction;
- *Action space*: set of possible dialogue moves;
- *Observation space:* set of possible interpretations of linguistic utterances, together with their confidence score;
- *Transition function*: definition of the dialogue "dynamics"
- *Observation function*: "sensor model" between utterance interpretations and their actual (hidden) intentions
- *Reward function*: big positive reward for long-term goals (e.g. the retrieval of important information), and small negative rewards for various "inconveniences" (e.g. prompting the user to repeat).

POMDP belief state

- Key idea of POMDPs: the "true" dialogue state is not directly observable, but can only be inferred from observations.
- This is expressed by the **belief state**, which represents the information known to the agent
- This belief state is a *probability distribution* over possible states
- Often encoded as a *Bayesian network*, where each node expresses a state variable

Example of simple Bayesian network with three variables:



POMDP policies

- Given a POMDP, we want to find a policy mapping any belief b to an action to execute
- The policy is therefore a mapping $\pi: B \rightarrow A$, where B is the space of possible beliefs
- Extracting a dialogue policy for a POMDP is much trickier than for a MDP, since the space B is high-dimensional and continuous
  - Approximation is needed!

[S. Young et al. (2010). «The Hidden Information State Model: a practical framework for POMDP-based spoken dialogue management», in *Computer Speech and Language*]

Outline

- What is dialogue management?
- Approaches
- **Evaluation metrics**
- Summary

Evaluation

- Some dialogue processing tasks are relatively straightforward to evaluate:
  - ASR: *Word Error Rate*
  - NLU: *[precision, recall, F-score]* for parsing, reference resolution, and dialogue act recognition
  - TTS (next lecture): evaluation by human listeners on sound intelligibility and quality
- But evaluating the global conversational behaviour of the system is much harder!

Evaluation

- A good (but subjective) way to evaluate a dialogue system is to get **user satisfaction ratings**
- This can be done via surveys that users are asked to fill after interacting with the system, for instance:

TTS Performance	Was the system easy to understand ?
ASR Performance	Did the system understand what you said?
Task Ease	Was it easy to find the message/flight/train you wanted?
Interaction Pace	Was the pace of interaction with the system appropriate?
User Expertise	Did you know what you could say at each point?
System Response	How often was the system sluggish and slow to reply to you?
Expected Behavior	Did the system work the way you expected it to?
Future Use	Do you think you'd use the system in the future?

[M. Walker et al. (2001), «Quantitative and Qualitative Evaluation of Darpa Communicator Spoken Dialogue Systems», *Proceedings of ACL*]

Evaluation

- User evaluation surveys are useful, but they are expensive and time-consuming
  - Not feasible to do user evaluations after each system change!
- Need a way to automate the evaluation process
- Possible solution: rely on evaluation metrics that can be directly extracted from the dialogue, and are known to *correlate with user satisfaction*
  - Improving these observable metrics should therefore increase user satisfaction

Evaluation

Criteria	Description	Possible metrics
Task completion success	How often was the system able to complete its task successfully?	- κ agreement for slot-filling applications - completion ratio
Efficiency costs	How efficient was the system in executing its task?	- number of turns (from user, system, or both) - total elapsed time
Quality costs	How good was the system interaction?	- number of ASR rejection prompts - number of user barge-ins - number of error messages

NB: this list of metrics is of course not exhaustive!

Evaluation

- Assume we found a set of metrics for our domain... how do we weight their relative importance?
- PARADISE framework:
  1. We start by doing a user satisfaction survey for a set of dialogues (via questionnaires such as the one we have seen)
  2. We also measure their performance according to our metrics (task success, number of turns, number of errors, etc.)
  3. Finally, we apply *multiple regression* to train the weight of each metric (this way, we can ensure our metrics correlate with the user satisfaction)

[M. Walker et al. (1997), "PARADISE: A general framework for evaluating spoken dialogue agents", Proceedings of ACL]

Evaluation

PARADISE model of performance:

$$\text{Performance} = \sum_{i=1}^{n} w_i \, m_i$$

Weight of metric *i* (to learn via regression based on the user satisfaction)

Measure for metric *i* (for instance, task success = 0.91)

NB: the weights can be negative!

Evaluation

- The PARADISE model works quite well for slot-filling applications
- Appropriate metrics might be harder to define or apply to other domains
  - For instance, a «companion»-type of agent has no clear task, and shouldn't necessarily minimise the number of turns!
  - For these domains, the notion of «appropriateness», i.e. the system's ability to maintain a natural, fluid conversation over time might be more important

[D. Traum et al. (2004), «Evaluation of multi-party virtual reality dialogue interaction», in *Proceedings of LREC*]

Outline

- What is dialogue management?
- Approaches
- Evaluation metrics
- **Summary**

Summary

- The dialogue manager **decides** what to do or say in a given state of the conversation, based on:
  - Long-term goals (and trade-offs between them)
  - (Partial) current knowledge of the situation
- Various approaches to dialogue management:
  - Finite-state automata are simple to design, but are very rigid
  - Frame-based and logical systems are more scalable, but require detailed specifications of the dialogue domain
  - Finally, statistical techniques can *learn* optimal policies from data
- Objective and subjective metrics for evaluation

Next Friday

- Next Friday, we'll talk about:
  - *Natural language generation*: how to convert a high-level communicative goal into an utterance
  - *Speech synthesis*: how to convert an utterance into an actual waveform
  - *Wrap-up* of the "dialogue systems" part