UiO : University of Oslo

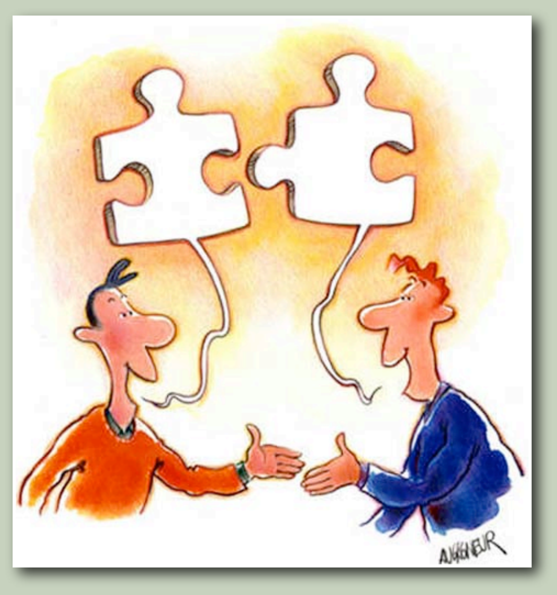
Multi-policy dialogue management

Pierre Lison

Logic & natural language group, Department of Informatics
University of Oslo, Norway.

Introduction

- Many dialogue domains are naturally *open-ended*, and exhibit both *partial observability* and *large state spaces*
- How can we efficiently design or optimise dialogue management (DM) policies of high quality for such complex domains?
- Most approaches to DM seek to capture the full complexity of the interaction in a *single* dialogue model and control policy
- We present an alternative approach where the dialogue manager operates directly with a *collection* of small, interconnected policies



- Viewing dialogue management as a process operating over **multiple policies** yields several benefits:
  - Easier for the application developer to model several small, local/partial interactions than a single monolithic one. Each local model can be independently modified or extended
  - Different frameworks can be combined: the developer is free to decide which approach is most appropriate to solve a specific problem, without having to commit to a unique framework
  - It also becomes possible to integrate both handcrafted and learned/optimised policies in the same control algorithm

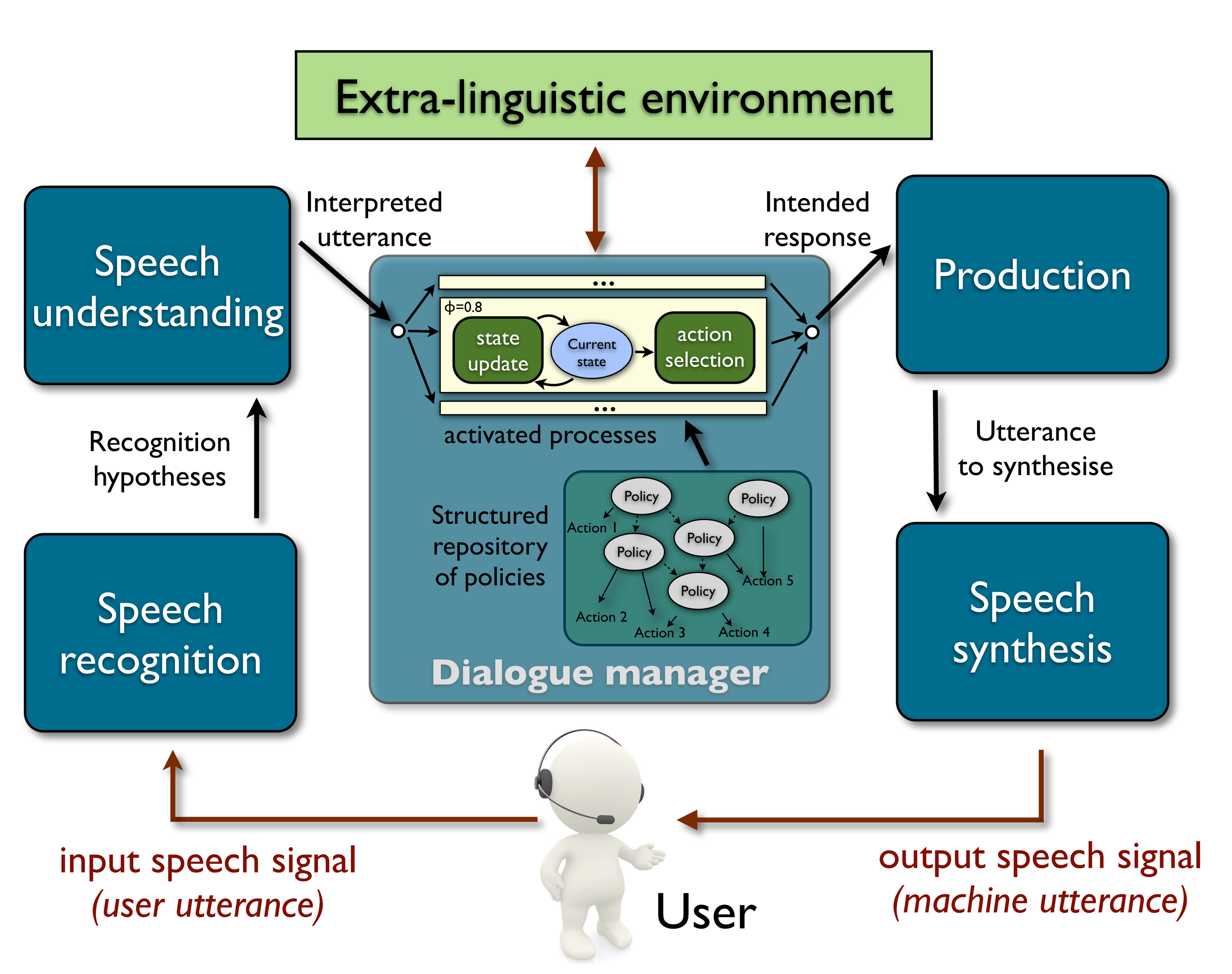
We allow policies to be connected with each other in two ways:

- *hierarchically*: one policy triggers one another via an *abstract action*
- *concurrently*: several policies are active and running in parallel

- **Challenge**: At each turn, the dialogue manager must know which policy is currently active and is responsible for deciding the next action to perform (= *meta-control* of the policies)
- In the general case, the "activation status" of a given policy is not directly observable and must be indirectly estimated
- We thus need a "soft" control mechanism able to explicitly account for the uncertainty about the completion status of each policy

- **Key idea:** provide a *probabilistic* treatment of this meta-control problem by introducing the concept of "*activation value*"
- The activation value of policy i is defined as the probability $P(\varphi_i)$, where φ_i is a random variable denoting the event of policy i being currently in focus. The estimate of the activation value given all available information at time t is denoted $b_t(\varphi_i) = P(\varphi_i | \ldots)$
- The values for all policies are grouped in an *activation vector* $\mathbf{b}_\Phi = \langle b(\varphi_1) \ldots b(\varphi_n) \rangle$ which is updated before and after each turn.
- We describe below how this activation vector is estimated, and how it is exploited to control the dialogue management execution

Approach



- We first decompose dialogue policies in 3 distinct functions:
  - **Observation update** ($\text{OBS-UPDATE} : \mathcal{S} \times \mathcal{O} \rightarrow \mathcal{S}$): given the current state s and observation o, update to state s'
  - **Action selection** ($\pi : \mathcal{S} \rightarrow \mathcal{A}$): takes the updated state s' as input, outputs the optimal action a^* to execute (if any)
  - **Action update** ($\text{ACT-UPDATE} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$): re-updates the current state s' given the execution of the action a^*

- We additionally define **two new functions** for each policy:
  - $\text{LIKELIHOOD}_i(s, o) : \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$ returns the likelihood of the observation o if policy i is active and in state s
  - $\text{ACTIVATION}_i(s) : \mathcal{S} \rightarrow [0, 1]$ computes the probability of policy i being active given its current local state, i.e. $P(\varphi_i | s_i)$
- These functions are implemented via heuristics which depend on the policy encoding (FSC, POMDP, etc.)

- We start with a set of processes $\mathcal{P}$. Each process has a specific policy i, a state s_i and activation value $b(\varphi_i)$
- Hierarchical and concurrent constraints between policies are specified in a network of constraints $\mathcal{C}$
- Algorithm 1 illustrates the execution process: selection of most active process (1-5), extraction of optimal action (6), and update of activation vector (7-11)
- Algorithm 2 extracts the optimal action given a process. If an abstract action is found, a new process is forked.

Algorithm 1 : MAIN-EXECUTION $(\mathcal{P}, o)$

Require: $\mathcal{P}$: the current set of processes
Require: $\mathcal{C}$: network of constraints on b_Φ
Require: o: a new observation

1: **for all** $i \in \mathcal{P}$ **do**
2: $P(o|\phi_i, s_i) \leftarrow \text{LIKELIHOOD}_i(s_i, o)$
3: $b'(\phi_i) \leftarrow \eta \cdot P(o|\phi_i, s_i) \cdot b(\phi_i)$
4: **end for**
5: Select process $p \leftarrow \arg\max_i b'(\phi_i)$
6: $a^* \leftarrow \text{GET-OPTIMAL-ACTION}(p, o)$
7: **for all** $i \in \mathcal{P}$ **do**
8: $P(\phi_i|s_i) \leftarrow \text{ACTIVATION}_i(s_i)$
9: Prune i from $\mathcal{P}$ if inactive
10: Compute $b(\phi_i)$ given $P(\phi_i|s_i)$ and $\mathcal{C}$
11: **end for**
12: **return** $a*$

Algorithm 2 : GET-OPTIMAL-ACTION (p, o)

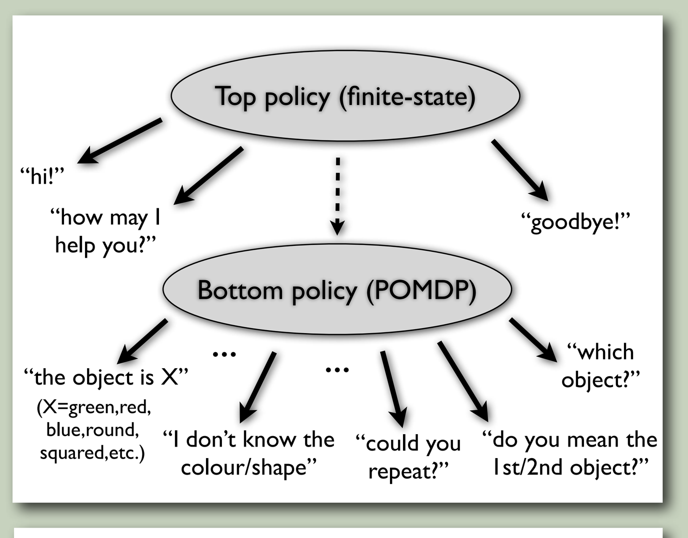
Require: p: process with current state s_p
Require: o: a new observation
Require: children(p): list of current processes directly or indirectly forked from p

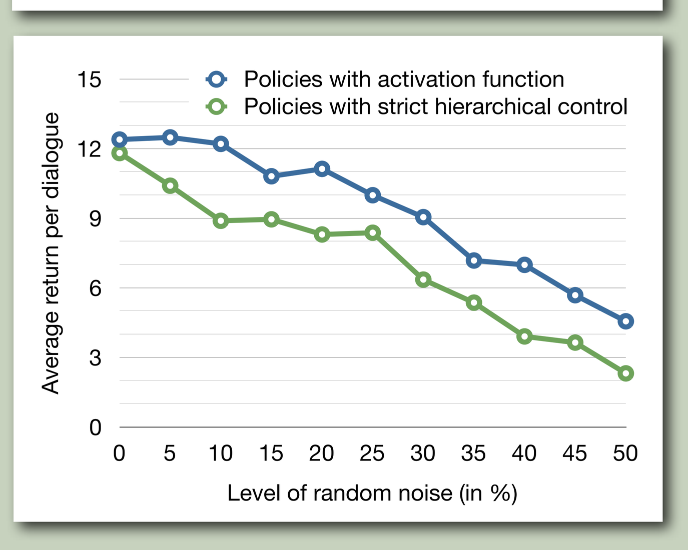
$s_p \leftarrow \text{OBS-UPDATE}_p(s_p, o)$
$a^* \leftarrow \pi_p(s_p)$
$s_p \leftarrow \text{ACT-UPDATE}_p(s_p, a^*)$
if a^* is an abstract action **then**
Fork new process q with policy from a^*
Add q to set of current processes $\mathcal{P}$
$a^* \leftarrow \text{GET-OPTIMAL-ACTION}(q, o)$
children$(p) \leftarrow \langle q \rangle +$ children(q)
else
children$(p) \leftarrow \langle\rangle$
end if
Add to $\mathcal{C}$ the constraint $b(\phi_p) = (1 - \sum_{i \in \text{children}(p)} b(\phi_i)) \cdot P(\phi_p|s_p)$
return $a*$

Evaluation

- Experiment with simple, simulated dialogue domain: visual learning task between a human and a robot in a scene including objects with various properties (color,shape)
  - The human asks questions about the objects, and replies to the robot's answer
  - Uncertainty in the verbal inputs and in the visual perception
- Dialogue domain is modelled with *two interconnected policies*:
  - Top policy (finite-state controller) handles the general interaction
  - Bottom policy (POMDP) answers the object-related queries

- *Goal of experiment*: compare the performance of Algorithm 1 with a hierarchical control mechanism (top policy blocked until bottom releases its turn), using a handcrafted user simulator and various levels of noise
- The results (average return) demonstrate that activation values are beneficial for multi-policy dialogue management, especially in the presence of noise.
- This is due to the *soft control* behaviour enabled by the use of the activation vector





Conclusion

- We introduced a new approach to dialogue management based on **multiple, interconnected policies** weighted by *activation values*
- Activation values are updated after each turn to reflect which part of the interaction is in focus
- Future work will focus on:
  - enabling the use of *shared state variables* accessible to distinct policies
  - applying *reinforcement learning* to learn model parameters on multiple policies

Want to know more? Check out my papers at:
http://folk.uio.no/plison
Or email me at: plison@ifi.uio.no