Multi-Policy Dialogue Management

Pierre Lison
Logic and Natural Language Group
Department of Informatics
University of Oslo, Norway

Abstract

We present a new approach to dialogue management based on the use of multiple, interconnected policies. Instead of capturing the complexity of the interaction in a single large policy, the dialogue manager operates with a collection of small local policies combined concurrently and hierarchically. The meta-control of these policies relies on an activation vector updated before and after each turn.

1 Introduction

Many dialogue domains are naturally open-ended. This is especially the case in situated dialogue, where the conversational agent must operate in continuously changing environments where there is often no single, pre-specified goal to achieve. Depending on the situation and the (perceived) user requests, many distinct tasks may be performed. For instance, a service robot for the elderly might be used for cleaning, monitoring health status, and delivering information. Each of these tasks features a specific set of observations, goals, constraints, internal dynamics, and associated actions.

This diversity of tasks and models poses significant challenges for dialogue systems, and particularly for dialogue management. Open-ended interactions are indeed usually much more difficult to model than classical slot-filling applications, where the application domain can provide strong constraints on the possible dialogue transitions. Using machine learning techniques to learn the model parameters can help alleviate this issue, but only if the task can be efficiently factored and if a sufficient amount of data is available. Once a model of the interaction and its associated environment is available, a *control policy* then needs to be learned or designed for the resulting state space. The extraction of good control policies can be computationally challenging, especially for interactions which simultaneously combine *partial observability* (to deal with noisy and incomplete observations) and *large state spaces* (if the optimal behaviour depends on a wide range of user- and context-specific factors) – which is the case for many open-ended domains.

In this paper, we present ongoing work on a new approach to dialogue management which seeks to address these issues by leveraging prior knowledge about the interaction structure to break up the full domain into a set of smaller, more predictable subdomains. Moving away from the idea of capturing the full interaction complexity into a unique, monolithic policy, we extend the execution algorithm of the dialogue manager to directly operate with a *collection* of small, interconnected local policies.

Viewing dialogue management as a decision process over multiple policies has several benefits. First, it is usually easier for the application developer to model several small, local interactions than a single large one. Each local model can also be independently modified, extended or replaced without interfering with the rest of the system, which is crucial for system maintenance. Finally, different theoretical frameworks can be used for different policies, which means that the developer is free to decide which approach is most appropriate to solve a specific problem, without having to commit to a unique theoretical framework for the whole application. For instance, one policy might be expressed as a solution to a Partially Observable Markov Decision Process (POMDP) while another policy is encoded as a

Proceedings of the SIGDIAL 2011: the 12th Annual Meeting of the Special Interest Group on Discourse and Dialogue, pages 294–300,
Portland, Oregon, June 17-18, 2011.

hand-crafted finite-state controller, and the two can be integrated in the same control algorithm.

One of the challenges when operating with multiple policies is the "meta-control" of these policies. At each turn, the system must know which policy is currently in focus and is responsible for deciding the next action to perform. Since dialogue management operates under significant uncertainty, the system can never be sure whether a given policy is terminated or not. We thus need a "soft" *control mechanism* which is able to explicitly account for the uncertainty about the completion status of each policy. This is precisely what we present in this paper.

The rest of the paper is as follows. We first provide general definitions of dialogue policies, and present an algorithm for dialogue management operating on multiple policies. We then present an implementation of the algorithm together with an empirical evaluation of its performance, and conclude the paper by comparing our approach to related work.

2 Background

We start by providing a generic definition of a policy which can hold independently of any particular encoding. Dialogue policies can indeed generally be decomposed in three basic functions, which are called consecutively upon each turn: (1) *observation update*, (2) *action selection* and (3) *action update*.

2.1 Observation update

The role of *observation update* is to modify the policy's current state[1] upon receiving a new observation, which can be linguistic or extra-linguistic.

Observation update is formally defined as a function OBS-UPDATE : $\mathcal{S} \times \mathcal{O} \rightarrow \mathcal{S}$ which takes as input the current state s and a new observation o, and outputs the updated state s'. For instance, a finite-state controller is expressed by a set of nodes $\mathcal{N}$ and edges $\mathcal{E}$, where the state is expressed by the current node, and the update mechanism is defined as:

$$\text{OBS-UPDATE}(s, o) = \begin{cases} s' & \text{if } \exists \text{ an edge } s \xrightarrow{o} s' \\ s & \text{otherwise} \end{cases}$$

In information-state approaches (Larsson and Traum, 2000), the update is encoded in a collection of *update rules* which can be applied to infer the new state. In POMDP-based dialogue managers (Young et al., 2010), the observation update corresponds to the *belief monitoring/filtering* function.

2.2 Action selection

The second mechanism is *action selection*, whose role is to select the optimal (communicative) action to perform based on the new estimated state. The action selection is a function $\pi : \mathcal{S} \rightarrow \mathcal{A}$ which takes the updated state as input, and outputs the optimal action to execute (which might be void).

Different encodings are possible for the action selection mechanism. Finite-state controllers use a straightforward mechanism for π, since each state node in the graph is directly associated with a unique action. Information-state approaches provide a mapping between particular sets of states and actions by way of selection rules. Decision-theoretic approaches such as MDPs and POMDPs rely on an estimated *action-value function* which is to be maximised: $\pi(s) = \arg\max_a Q(s, a)$. The utility function $Q(s, a)$ can be either learned from experience or provided by the system designer.

2.3 Action update

Once the next action is selected and sent for execution, the final step is to re-update the dialogue state given the action. Contrary to the two previous functions which can be found in all approaches, this third mechanism is optional and is only implemented in some approaches to dialogue management.

Action update is formally defined as a function ACT-UPDATE : $\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. Finite-state and information-state approaches typically have no explicit account of action update. In (PO)MDPs approaches, the action update function is computed with the *transition function* of the model.

3 Approach

3.1 Activation vector

To enable the dialogue manager to operate with multiple policies, we introduce the notion of *activation value*. The activation value of a policy i is the probability $P(\phi_i)$ that this policy is in focus for the interaction, where the random variable ϕ_i denote the activation of policy i. In the rest of this paper, we

[1] We adopt here a broad definition of the term "state" to express any description of the agent's current knowledge. In a POMDP, the state thus corresponds to the *belief state*.

shall use $b_t(\phi_i)$ to denote the activation value of policy ϕ at time t, given all available information. The $b_t(\phi_i)$ value is dependent on both the completion status of the policy itself and the activations of the other policies: $b_t(\phi_i) = P(\phi_i|s_i, b_t(\phi_1), ...b_t(\phi_n))$. We group these values in an *activation vector* $b_\Phi = \langle b(\phi_1)...b(\phi_n)\rangle$ which is updated after each turn.

3.2 Activation functions

To compute the activation values, we define the two following functions associated with each policy:

1. $\text{LIKELIHOOD}_i(s, o) : \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$ computes the *likelihood* of the observation o if the policy i is active and currently in state s. It is therefore an estimate of the probability $P(o|\phi_i, s)$.

2. $\text{ACTIVATION}_i(s) : \mathcal{S} \rightarrow [0, 1]$ is used to determine the probability of policy i being active at a given state s. In other words, it provides an estimate for the probability $P(\phi_i|s)$.

These functions are implemented using heuristics which depend on the encoding of the policy. For a finite-state controller, we realise the function $\text{LIKELIHOOD}(s, o)$ by checking whether the observation matches one of the outward edges of the current state node – the likelihood returns a high probability if such a match exists, and a low probability otherwise. Similarly, the ACTIVATION function can be defined using the graph structure of the controller:

$$\text{ACTIVATION}(s) = \begin{cases} 1 & \text{if } s \text{ non-final} \\ \delta & \text{if } s \text{ final with outgoing edges} \\ 0 & \text{if } s \text{ final w/o outgoing edges} \end{cases}$$

where δ is a constant between 0 and 1.

3.3 Constraints between policies

In addition to these activation functions, various *constraints* can hold between the activation of related policies. Policies can be related with each other either *hierarchically* or *concurrently*.

In a *hierarchical mode*, a policy A triggers another policy B, which is then executed and returns the control to policy A once it is finished. As in hierarchical planning (Erol, 1996; Pineau, 2004), we implement such hierarchy by distinguishing between *primitive actions* and *abstract actions*. An abstract action is an action which corresponds to the execution of another policy instead of leading directly to a primitive action. With such abstract actions, the system designer can define a hierarchical structure of policies as illustrated in Figure 1. When a policy A executes an abstract action pointing to policy B, the activation value of policy B is increased and the one of policy A proportionally decreased. This remains so until policy B terminates, at which point the activation is then transferred back to policy A.

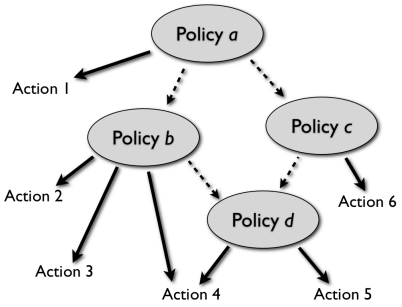

Figure 1: Graphical illustration of a hierarchical policy structure. Dotted lines denote abstract actions.

In a *concurrent mode*, policies stand on an equal footing. When a given policy takes the turn after an observation, the activations of all other concurrent policies are decreased to reflect the fact that this part of the interaction is now in focus. This redistribution of the activation mass allows us to run several policies in parallel while at the same time expressing a "preference" for the policy currently in focus. The "focus of attention" is indeed crucial in verbal interactions, and in linguistic discourse in general (Grosz and Sidner, 1986) – humans do not arbitrarily switch from one topic to another and back, but rather concentrate on the most salient elements.

The set of constraints holding between the activation values of hierarchical and concurrent policies is encoded in a simplified Bayesian network.

3.4 Execution algorithm

Algorithm 1 illustrates how the activation values are exploited to select the optimal action for multiple policies. The algorithm relies on a set of processes $\mathcal{P}$, where a process i is associated with a specific policy, a current state s_i for the policy, and a current activation value $b(\phi_i) \in b_\Phi$. As we have seen, each policy is fully described with five functions: $\text{LIKELIHOOD}(s, o)$, $\text{OBS-UPDATE}(s, o)$, $\pi(s)$, $\text{ACT-UPDATE}(s, a)$, and $\text{ACTIVATION}(s)$. A network

of conditional constraints $\mathcal{C}$ on the activation vector is also given as input to the algorithm.

Algorithm 1 operates as follows. Upon receiving a new observation, the procedure loops over all processes in $\mathcal{P}$ and updates the activation values $b'(\phi_i)$ for each given the likelihood of the observation (with η as a normalisation factor). Once this update is completed, the process p with the highest activation is selected, and the function GET-OPTIMAL-ACTION(p, o) is triggered.

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: $P(o|\phi_i, s_i) \leftarrow$ 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: $a^* \leftarrow$ GET-OPTIMAL-ACTION(p, o)
7: **for all** $i \in \mathcal{P}$ **do**
8: $P(\phi_i|s_i) \leftarrow$ 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** $a*$

Within GET-OPTIMAL-ACTION, the state of the process is updated given the observation, the next action a^* is selected using $\pi(s)$ and the state is updated again given this selection. If the action is abstract, the above-mentioned procedure is repeated until a primitive action is reached. The resulting hierarchical structure is recorded in $\mathsf{children}(p)$ which details, for each process $p \in \mathcal{P}$, the list of its children processes. To ensure consistency among the activation values in this hierarchy, a constraint is added to $\mathcal{C}$ for each process visited during execution.

Once the action a^* is found, the activation values $b(\phi_i)$ are recomputed according to the local activation function combined with the constraints $\mathcal{C}$. Processes which have become inactive (i.e. which have transferred control to one parent process) are also pruned from $\mathcal{P}$. Finally, the action a^* is returned.

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

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

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

4 Evaluation

The described algorithm has been implemented and tested with different types of policies. We present here a preliminary experiment performed with a small dialogue domain. The domain consists of a (simulated) visual learning task between a human and a robot in a shared scene including a small number of objects, described by various properties such as color or shape. The human asks questions related to these object properties, and subsequently confirms or corrects the robot's answers – as the case may be. We account for the uncertainty both in the linguistic inputs and in the visual perception.

We model this domain with two connected policies, one top policy handling the general interaction (including engagement and closing acts), and one bottom policy dedicated to answering each user question. The top policy is encoded as a finite-state controller and the bottom policy as a POMDP solved using the SARSOP algorithm, available in the APPL toolkit[2] (Kurniawati et al., 2008). A sample run is provided in Appendix **A**.

The experiment was designed to empirically compare the performance of the presented algorithm

[2] http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/

with a simpler hierarchical control algorithm which does not use any activation vector, but where the top policy is blocked until the sub-policy releases its turn. The policies themselves remain identical in both scenarios. We implemented a handcrafted user simulator for the domain, and tested the policies with various levels of artificial noise.

The average return for the two scenarios are provided in Figure 2. The results show that activation values are beneficial for multi-policy dialogue management, especially in the presence of noise.. This is due to the soft control behaviour provided by the activation vector, which is more robust than hierarchical control. Activation values provide a more fine-grained mechanism for expressing the completion status of a policy, and therefore avoid fully "blocking" the control at a given level.

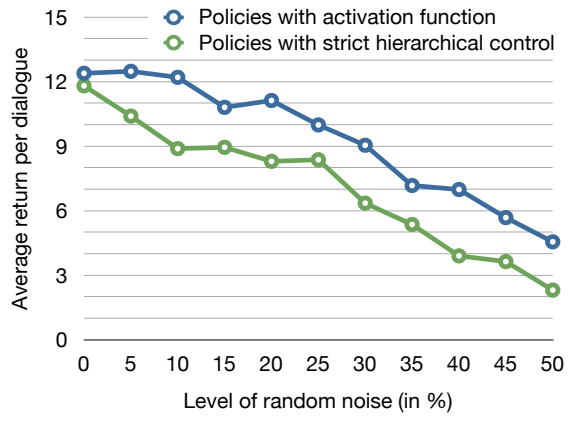

Figure 2: Average return (as generated by the handcrafted user simulator) for the two connected policies, using either the present algorithm or strict hierarchical control. 400 runs are used for each level of noise.

5 Related work

The exploitation of prior structural knowledge in control has a long history in the planning community (Erol, 1996; Hauskrecht et al., 1998), and has also been put forward in some approaches to dialogue modelling and dialogue management – see e.g. (Grosz and Sidner, 1990; Allen et al., 2000; Steedman and Petrick, 2007; Bohus and Rudnicky, 2009). These approaches typically rely on a task decomposition in goals and sub-goals, and assume that the completion of each of these goals can be fully observed. The novel aspect of our approach is precisely that we seek to relax this assumption of perfect knowledge of task completion. Instead, we treat the activation/termination status of a given policy as a *hidden variable* which is only indirectly observed and whose value at each turn is determined via probabilistic reasoning operations.

The idea of combining different dialogue management frameworks in a single execution process has also been explored in previous work such as (Williams, 2008), but only as a filtering mechanism – one policy constraining the results of another. Related to the idea of concurrent policies, (Turunen et al., 2005) describes a software framework for distributed dialogue management, mostly focussing on architectural aspects. In the same vein, (Lemon et al., 2002; Nakano et al., 2008) describe techniques for dialogue management respectively based on multi-threading and multi-expert models. (Cuayáhuitl et al., 2010) describe an reinforcement learning approach for the optimisation of hierarchical MDP policies, but is not extended to other types of policies. Closest to our approach is the PolCA+ algorithm for hierarchical POMDPs presented in (Pineau, 2004), but unlike our approach, her method does not support temporally extended actions, as the top-down trace is repeated after each time step.

6 Conclusion

We introduced a new approach to dialogue management based on multiple, interconnected policies controlled by activation values. The values are updated at the beginning and the end of each turn to reflect the part of the interaction currently in focus.

It is worth noting that the only modification required in the policy specifications to let them run in a multi-policy setting is the introduction of the two functions LIKELIHOOD(s, o) and ACTIVATION(s). The rest remains untouched and can be defined independently. The presented algorithm is therefore well suited for the integration of dialogue policies encoded in different theoretical frameworks.

Future work will focus on various extensions of the approach and the use of more extensive evaluation metrics. We are also investigating how to apply reinforcement learning techniques to learn the model parameters in such multi-policy paradigms.

Acknowledgements

This work was supported by the EU FP7 IP project "*'ALIZ-E: Adaptive Strategies for Sustainable Long-Term Social Interaction*" (FP7-ICT-248116) and by a PhD research grant from the University of Oslo. The author would like to thank Stephan Oepen, Erik Velldal and Alex Rudnicky for their comments and suggestions on earlier drafts of this paper.

References

J. Allen, D. Byron, M. Dzikovska, G. Ferguson, L: Galescu, and A. Stent. 2000. An architecture for a generic dialogue shell. *Natural Language Engineering*, 6:213–228, September.

D. Bohus and A. I. Rudnicky. 2009. The RavenClaw dialog management framework: Architecture and systems. *Computer Speech & Language*, 23:332–361, July.

H. Cuayáhuitl, S. Renals, O. Lemon, and H. Shimodaira. 2010. Evaluation of a hierarchical reinforcement learning spoken dialogue system. *Computer Speech & Language*, 24:395–429, April.

K. Erol. 1996. *Hierarchical task network planning: formalization, analysis, and implementation*. Ph.D. thesis, College Park, MD, USA.

B. J. Grosz and C. L. Sidner. 1986. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12:175–204, July.

B. J. Grosz and C. L. Sidner. 1990. Plans for discourse. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 417–444. MIT Press, Cambridge, MA.

M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier. 1998. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 220–229.

H. Kurniawati, D. Hsu, and W.S. Lee. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. Robotics: Science and Systems*.

S. Larsson and D. R. Traum. 2000. Information state and dialogue management in the trindi dialogue move engine toolkit. *Natural Language Engineering*, 6:323–340, September.

O. Lemon, A. Gruenstein, A. Battle, and S. Peters. 2002. Multi-tasking and collaborative activities in dialogue systems. In *Proceedings of the 3rd SIGDIAL workshop on Discourse and Dialogue*, pages 113–124, Stroudsburg, PA, USA.

M. Nakano, K. Funakoshi, Y. Hasegawa, and H. Tsujino. 2008. A framework for building conversational agents based on a multi-expert model. In *Proceedings of the 9th SIGDIAL Workshop on Discourse and Dialogue*, pages 88–91, Stroudsburg, PA, USA.

J. Pineau. 2004. *Tractable Planning Under Uncertainty: Exploiting Structure*. Ph.D. thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

M. Steedman and R. P. A. Petrick. 2007. Planning dialog actions. In *Proceedings of the 8th SIGDIAL Workshop on Discourse and Dialogue (SIGdial 2007)*, pages 265–272, Antwerp, Belgium, September.

M. Turunen, J. Hakulinen, K.-J. Räihä, E.-P. Salonen, A. Kainulainen, and P. Prusi. 2005. An architecture and applications for speech-based accessibility systems. *IBM Syst. J.*, 44:485–504, August.

J. D. Williams. 2008. The best of both worlds: Unifying conventional dialog systems and POMDPs. In *International Conference on Speech and Language Processing (ICSLP 2008)*, Brisbane, Australia.

S. Young, M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu. 2010. The hidden information state model: A practical framework for pomdp-based spoken dialogue management. *Computer Speech & Language*, 24:150–174, April.

A Example of execution with two policies

We provide here an example of execution of Algorithm 1 with the two policies described in the evaluation section. Figure 3 illustrates the policy hierarchy, which consists of two policies connected with an abstract action. The finite-state graph of the top policy is shown in Figure 4.

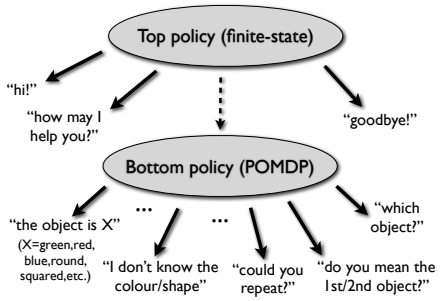

Figure 3: Hierarchical structure of the two policies.

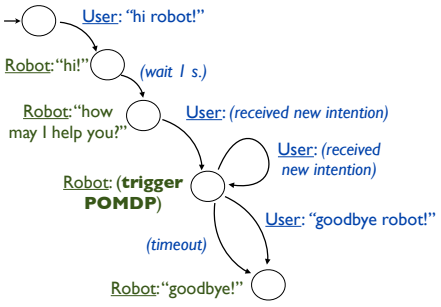

Figure 4: Finite-state controller for the top policy.

A.1 Example of interaction

Transcript A.1 provides a full example of interaction as generated by the user simulator. Human utterances can contain variable noise levels, which makes it necessary to use clarification requests when the user intention is unclear (lines 5 and 10).

Contextual knowledge can also influence how the dialogue unfolds – here, the robot knows that the first object is blue but is uncertain about the second object. Such contextual knowledge is expressed in terms of hidden variables in the POMDP model.

The execution of Algorithm 1 for the above example proceeds as follows. From line 1 to line 4, the set $\mathcal{P}$ contains one single process, based on the top

Transcript A.1 Example of interaction

1: HUMAN: Hi robot!
2: ROBOT: Hi! [*wait 1 s.*]
3: ROBOT: How may I help you?
4: HUMAN: Could you tell me the colour of the first object? [*trigger POMDP*]
5: ROBOT: Sorry, could you repeat?
6: HUMAN: What colour is the first object?
7: ROBOT: It is blue
8: HUMAN: That's right
9: HUMAN: Now, could you tell me the colour of the second object? [*trigger POMDP*]
10: ROBOT: Do you mean the second object?
11: HUMAN: Yes
12: ROBOT: I don't know
13: HUMAN: It is green
14: ROBOT: Thanks
15: HUMAN: Ok I'm done. Goodbye!
16: ROBOT: Goodbye! [*close interaction*]

policy. This process is fully activated. At line 5, the new user intention is mapped to the abstract action, and thus to the creation of a second process based the POMDP policy. This second process becomes highly activated due its function ACTIVATION(s). Given the constraint holding between the activation values of two processs, the high activation of the bottom process leads to a proportional reduction of the activation value of the top process.

The utterance at line 8 is interpreted as a signal that the user intention has been met, and the activation value of the bottom process is subsequently decreased. The top process is then partially reactivated, and interprets the utterance at line 9 as a new user intention, leading to the creation of a new POMDP process. This process handles the interaction until the line 14, where the control is transferred back to the finite-state controller.

The better performance of Algorithm 1 compared to strict hierarchical control is due to the lines $8-9$. In the presence of noise, the transition to the second question might not be detected (if the confidence scores of the utterance is below a fixed threshold). In such case, the dialogue manager might stay "stuck" in the first POMDP process instead of interpreting the utterance as a new question.