Decision-Theoretic Planning with Linguistic Terms in GOLOG

Stefan Schiffer
Knowledge-Based Systems Group
RWTH Aachen University
Aachen, Germany
schiffer@cs.rwth-aachen.de

Alexander Ferrein
Aachen University of Applied Sciences
Mobile Autonomous Systems &
Cognitive Robotics Institute
Aachen, Germany
ferrein@fh-aachen.de

Abstract

In this paper we propose an extension of the action language GOLOG that integrates linguistic terms in non-deterministic argument choices and the reward function for decision-theoretic planning. It is often cumbersome to specify the set of values to pick from in the non-deterministic-choice-of-argument statement. Also, specifying a reward function is not always easy, even for domain experts. Instead of providing a finite domain for values in the non-deterministic-choice-of-argument statement in GOLOG, we now allow for stating the argument domain by simply providing a formula over linguistic terms and fuzzy fluents. In GOLOG's forward-search DT planning algorithm, these formulas are evaluated in order to find the agent's optimal policy. We illustrate this in the Diner Domain where the agent needs to calculate the optimal serving order.

1 Introduction

The action language GOLOG [Levesque *et al.*, 1997] has proven useful for encoding the high-level behaviors of a robot or agent (e.g. [Ferrein and Lakemeyer, 2008; Schiffer *et al.*, 2012]). With its foundations in the Situation Calculus [McCarthy, 1963; Reiter, 2001], complex behaviors are described in terms of actions with preconditions and effects. The world evolves from an initial situation due to actions. So-called fluents (predicates with a situation term as the last argument) keep track of changes of the properties of the world. Many extensions to the original GOLOG dialect were proposed, for instance, to deal with continuous change, allow for probabilistic projections, or decision-theoretic planning (e.g. [Grosskreutz, 2000; Grosskreutz and Lakemeyer, 2001; Boutilier *et al.*, 2000]). We build on a variant of GOLOG called READYLOG [Ferrein and Lakemeyer, 2008] which integrates many of the different dialects into an online interpreter that allows to encode high-level behaviors of an agent for dynamic real-time domains. READYLOG has shown its usefulness in applications ranging from robotic soccer to domestic service robots [Schiffer *et al.*, 2006; 2012].

One of the features that we found particularly useful to define the behavior of an agent or robot in a flexible way was to use decision-theoretic planning. The programmer states the different action alternatives and a reward function in order to select preferred world situations; the optimal policy is then calculated and executed. Besides this non-deterministic choice of actions, GOLOG offers a non-deterministic-choice-of-argument statement. For a finite domain of arguments, an optimal policy is computed. This is in particular helpful when the agent faces incomplete knowledge and particular information has to be sensed at run-time. However, to specify the argument domain is often cumbersome. In this paper, we propose an extension to GOLOG that integrates linguistic terms in non-deterministic argument choices and the reward function for decision-theoretic planning. We demonstrate the extension in the Diner Domain, where a waitron agent has to serve coffee and dishes as hot as possible. The agent has to decide on which order to deliver first as the coffee and meals cool down over time. We show how linguistic terms as defined in the Fuzzy Logic extension of GOLOG [Ferrein *et al.*, 2008; Schiffer *et al.*, 2011] are formally integrated into the forward-search value iteration algorithm used in READYLOG.

The remainder of the paper is organized as follows. In the next section, we briefly introduce the Diner Domain. Then, in Section 3, we review the background of this work, namely READYLOG and our Fuzzy Logic extensions to the Situation Calculus. In Section 4, we bring together the linguistic terms and decision-theoretic planning and define the respective language constructs formally. We conclude with Section 5.

2 The Diner Domain

The example we use in this paper will be from the Diner Domain. In the Diner Domain, a waitron agent has to decide which of its assigned tables it should serve first in order to serve coffee and meals as hot as possible. Of course, the longer the distance for coffee and meals to be served, the cooler the dishes will be when served to the customer. In our example, the waitron was assigned to serve tables T_1 and T_9. The coffee is regarded as cold if its temperature lies between 0–50 centigrades, it is perceived as luke warm between 45 and 65 degrees, hot between 60 and 80 degrees; above 75 degrees we regard the coffee as veryhot. Despite a negative exponential cooling rate in reality, we assume a linear rate for the sake of simplicity in this example. For every 10 seconds we assume that coffee and meals cool down 1 degree. Traversing a square in the Diner Domain takes the waitron

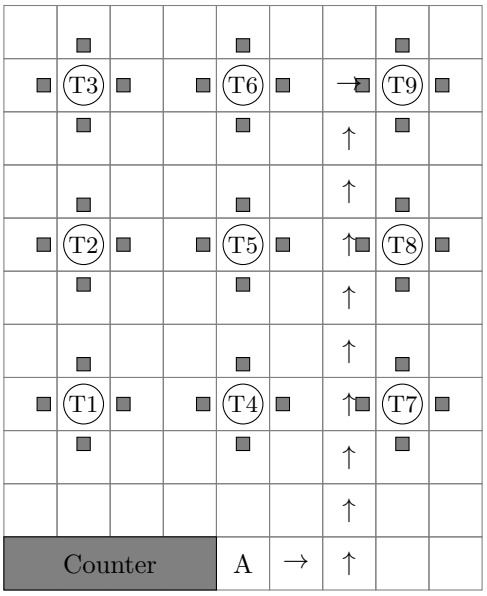

Figure 1: The Diner Domain

nil	empty program
α	primitive action
$\varphi?$	wait/test action
$\mathbf{waitFor}(\tau)$	event-interrupt
$[\sigma_1; \sigma_2]$	sequence
$\mathbf{if}\ \varphi\ \mathbf{then}\ \sigma_1\ \mathbf{else}\ \sigma_2\ \mathbf{endif}$	conditional
$\mathbf{while}\ \varphi\ \mathbf{do}\ \sigma\ \mathbf{endwhile}$	loop
$\mathbf{withCtrl}\ \varphi\ \mathbf{do}\ \sigma\ \mathbf{endwithCtrl}$	guarded execution
$\sigma_1 \,\|\|\, \sigma_2$	prioritized execution
$\mathbf{withPol}(\sigma_1, \sigma_2)$	prioritized exec. of σ_2
$\mathbf{prob}(p, \sigma_1, \sigma_2)$	probabilistic exec. of σ_1, σ_2
$\mathbf{pproj}(c, \sigma)$	prob. projection of prog's
$\{\mathbf{proc}\ P_1(\vec{\vartheta}_1)\sigma_1\ \mathbf{endproc}; \cdots\}$	procedures
$\mathbf{solve}(h, f, \sigma)$	initiate decision-theoretic optimization over σ up to a fixed horizon h
$\sigma_1 \,\vert\, \sigma_2$	non-deterministic decision-theoretic choice of prog's
$\mathbf{pickBest}(h, \vec{x}, \sigma)$	non-deterministic decision-theoretic choice of arg's

Figure 2: Some of READYLOG's constructs

agent 5 seconds. Fig. 1 shows an example. The agent (A) needs 12 actions (r, r, u, u, u, u, u, u, u, u, u, r) to get to table T_9. It takes the agent 60 seconds to reach that table. That means that a hot coffee at 65 centigrades will be lukewarm when served at table T_9. The deliverCoffee action finally delivers the coffee to the customer, once the right table has been reached by the agent.

3 Background

In this section we briefly introduce the Situation Calculus and READYLOG, showing the forward-search value iteration algorithm in greater detail. Then, we outline how Fuzzy sets can be formalized in the Situation Calculus.

3.1 Situation Calculus and READYLOG

The Situation Calculus is a second order language with equality which allows for reasoning about actions and their effects. The world evolves from an initial situation due to primitive actions. Possible world histories are represented by sequences of actions. The Situation Calculus distinguishes three different sorts: *actions*, *situations*, and domain *objects*. A special binary function symbol $do : action \times situation \rightarrow situation$ exists, with $do(a, s)$ denoting the situation which arises after performing action a in the situation s. The constant S_0 denotes the initial situation, i.e. the situation where no actions have occurred yet. The state the world is in is characterized by functions and relations with a situation as their last argument. They are called *functional* and *relational fluents*, respectively. Actions in the Situation Calculus are characterized by unique names. For each action one has to specify a *precondition axiom* stating under which conditions it is possible to perform the respective action and an *effect axiom* formulating how the action changes the world in terms of the specified fluents. Finally, we need a so-called basic action theory, which consists of the successor state (a special form of effect axioms) and precondition axioms and states what is true in the initial situation. Some foundational axioms are also required. For details we refer to [Reiter, 2001].

READYLOG [Ferrein and Lakemeyer, 2008; Ferrein, 2010], our variant of GOLOG [Levesque *et al.*, 1997], borrows ideas from [Levesque *et al.*, 1997; De Giacomo *et al.*, 2000; Grosskreutz, 2000; Grosskreutz and Lakemeyer, 2001; Boutilier *et al.*, 2000] and features the constructs given in Fig. 2. Besides standard constructs, READYLOG also features non-standard constructs such as **pproj**, where a program is probabilistically projected into the future, or the non-deterministic decision-theoretic choices of programs or arguments ("|" and **pickBest**, respectively). These constructs are used inside a **solve** statement and leave choices open that are filled by the decision-theoretic forward search algorithm deployed in READYLOG which we will explain next. The other constructs of READYLOG are shown in Fig. 2.

As we are aiming at extending decision-theoretic planning (DTP) in GOLOG in this paper, we have a closer look at the forward-search DTP algorithm that was proposed by [Boutilier *et al.*, 2000]. The search tree is expanded in a forward direction induced by the basic action theory. Fig. 3 shows the principle. The nodes in the search tree are expanded and the values are propagated back to the root. The path with the highest value represents the optimal policy. As the tree is constructed based on the basic action theory, it is particularly easy to restrict the search. Consider our waitron agent from the Diner Domain. To compute a path from the counter to table T_9, it could simply use the program shown in Algorithm 1, assuming the robot has the following action set $A = \{go\_right, go\_left, go\_up, go\_down\}$. Each of these actions takes the robot to the intended direction with a high probability, with a low probability it will end up in an adjacent location. The goal state, table T_9, has a positive reward while each other field has a negative reward. READYLOG now computes the optimal policy (shortest path) from the *Counter* to T_9 employing forward-search DTP. The forward-search DTP algorithm is implemented in terms of a number of $BestDo$

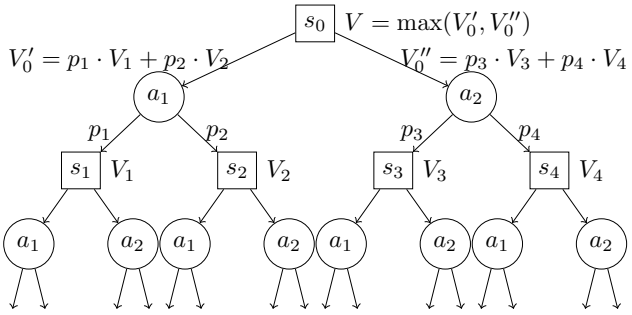

Figure 3: Decision tree search in READYLOG representing a non-deterministic choice of actions. In each situation s_i the agent may choose between a_1 or a_2. The resulting optimal policy is represented by the branch with maximal value V.

predicates. For non-deterministic choices of actions, its formal definition following [Boutilier *et al.*, 2000] is:

$$\begin{aligned} &BestDo((p_1 \mid p_2); p, s, h, \pi, v, pr) \stackrel{def}{=} \\ &\quad \exists \pi_1, v_1, pr_1. BestDo(p_1; p, s, h, \pi_1, v_1, pr_1) \wedge \\ &\quad \exists \pi_2, v_2, pr_2. BestDo(p_2; p, s, h, \pi_2, v_2, pr_2) \wedge \\ &\quad ((v_1, p_1) \geq (v_2, p_2) \wedge \pi = \pi_1 \wedge pr = pr_1 \wedge v = v_1) \vee \\ &\quad (v_1, p_1) < (v_2, p_2) \wedge \pi = \pi_2 \wedge pr = pr_2 \wedge v = v_2) \end{aligned}$$

The non-deterministic choice of action arguments is defined as:

$$\begin{aligned} &BestDo(\mathbf{pickBest}(x, \tau, p; p'), s, h, \pi, v, pr) \stackrel{def}{=} \\ &\quad BestDo(p|^x_{c_1} \mid \cdots \mid p|^x_{c_n}); p', s, h, \pi, v, pr) \end{aligned} \tag{1}$$

Free variables x in the program p are bound to a finite domain τ; for each "variable assignment" (denoted by $p|^x_{c_i}$) a new non-deterministic branch in the forward-search DTP is added. The policy is hence optimized for all possible variable assignments leading to the assignment which maximizes the reward function.

3.2 Fuzzy GOLOG

In [Ferrein *et al.*, 2008; Schiffer *et al.*, 2011] we introduced the notion of fuzzy fluents in GOLOG. Fuzzy fluents extend "ordinary" functional fluents in that they have a membership relation that defines, for a number of linguistic fuzzy terms the degree of membership for a particular function value.

In our Diner Domain, we want to serve hot coffee to our customers. The coffee, however, cools down quickly, depending on how long it takes to deliver the coffee. This, in turn, depends on the distance between the counter and the table where the coffee should be served. As an example for a fuzzy set defining the linguistic terms, we look at a distance relation. Distance in our diner example is understood as the Manhattan distance between two positions in the diner. We define distances between 0 and 3 blocks as close, between 3 and 6 as medium, and above 6 as far. Formally,

Algorithm 1: Decision-theoretic path planning in READYLOG

```
1 proc navigate
2     solve(h, reward, while loc ≠ goal do
3         (go_right | go_left | go_up | go_down)
4     endwhile)
5 endproc
```

$$\begin{aligned} &\mathfrak{F}(distance, u, \mu_u) \equiv \\ &\quad (distance = \mathsf{close} \supset (0, 1.0) \vee (1, 1.0) \vee \\ &\qquad (2, 0.75) \vee (3, 0.25) \vee (13/12, 0.5)) \wedge \\ &\quad (distance = \mathsf{medium} \supset (3, 0.25) \vee (4, 0.75) \vee \\ &\qquad (5, 0.75) \vee (6, 0.25) \vee (9/2, 0.5)) \wedge \\ &\quad (distance = \mathsf{far} \supset (6, 0.25) \vee (7, 0.75) \vee \\ &\qquad (8, 1.0) \vee (9, 1.0) \vee (95/12, 0.5)), \end{aligned}$$

where we use (u_i, μ_{u_i}) as an abbreviation for $u = u_i \wedge \mu_u = \mu_{u_i}$. The fuzzy set for the coffee temperature is shown in Fig. 4. Note that fuzzy categories can overlap. For instance, the coffee temperature 62°C belongs to the category luke as well as to the category hot. To query whether a value belongs to a certain category, one has to check if in the respective fuzzy set the value has a positive membership degree in that particular categorize. This is done with the predicate $is \subseteq real \times linguistic$. It is defined as

$$\mathrm{is}(\mathfrak{f}(\vec{t}, \sigma), \gamma) \stackrel{def}{=} \exists u, \mu_u. \mathfrak{f}(\vec{t}, \sigma) = u \wedge \mathfrak{F}(\gamma, u, \mu_u) \wedge \mu_u > 0,$$

where $\mathfrak{f}$ having the numeric value u is the fuzzy fluent to be queried, $\mathfrak{F}$ is the respective fuzzy set and μ_u is the degree of membership of value u in the fuzzy set $\mathfrak{F}$. In our distance example above, for instance, we have $\mathfrak{F}(\mathsf{medium}, 5, 0.75)$ to say that the numerical value 5 has a membership degree of 0.75 for the category medium. The predicate holds if the degree of membership is greater zero. For complex queries (logical formulas with fuzzy fluents), we have to define similar predicates $\mathrm{is}_{\complement}$ for the complement, $\mathrm{is}_{\star}$ for the conjunction, and $\mathrm{is}_{\oplus}$ for the disjunction of fuzzy fluents. See [Ferrein *et al.*, 2008; Schiffer *et al.*, 2011] for the formal definitions. Further, we need to define a function to defuzzify a linguistic term to a numeric value. As a defuzzifying function, we use the center of gravity (cog) which we formally define in [Ferrein *et al.*, 2008; Schiffer *et al.*, 2011]. In our distance example, the center of gravity for the category close is $13/12$. Note that we have to manually add the center of gravity for this respective category in our Situation Calculus fuzzy set formalization.

In the Diner Domain, we want to refer to positions in a room in a qualitative manner. This is why we introduce linguistic categories for the position in X and Y by the following membership functions:

$$\begin{aligned} &\mathfrak{F}(posX, u, \mu_u) \equiv \\ &\quad (posX = \mathsf{left} \supset (1, 1.0) \vee (2, 1.0) \vee (3, 1.0) \wedge \\ &\quad (posX = \mathsf{center} \supset (4, 1.0) \vee (5, 1.0) \vee (6, 1.0) \wedge \\ &\quad (posX = \mathsf{right} \supset (7, 1.0) \vee (8, 1.0) \vee (9, 1.0)). \end{aligned}$$

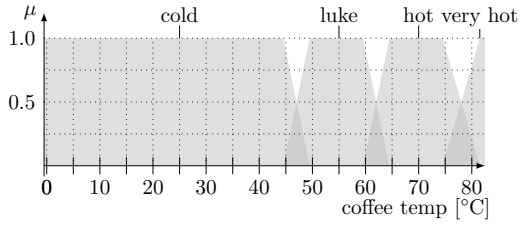

Figure 4: Coffee temperature membership function

For the y-coordinate we introduce a fuzzy fluent $posY$ and define the categories front, middle, back, referring to the tables whose ordinate have a distance of close, medium, and far from the *Counter*. In the next section, we propose an extension to DTP integrating those linguistic notions.

4 Extending DT-Planning in GOLOG with Linguistic Terms

One of the convenient features when specifying intelligent agents in GOLOG is that the agent designer can leave choices open that the agent then resolves on its own using an optimization theory. As already mentioned, the choices are the non-deterministic choice of action and the non-deterministic choice of argument. The latter is realized by means of the **pickBest** statement. It allows for specifying a set of possible values for a specific fluent for the program in the body of the statement. That program is evaluated with any of the values from the set.

4.1 Picking from Fuzzy Sets

We now propose to use, instead of a finite set of values, a fuzzy expression to specify the set of possible values for a fuzzy fluent. We introduce a new predicate **pickBestF** which takes a fuzzy expression instead of the regular set of the classical **pickBest**.

The idea is that instead of giving a finite set of variable or fluent values in the **pickBest** statement, the programmer now can state a formula specifying linguistic categories for a fuzzy fluent. For instance, if we want to optimize the coffee serving temperature, we could simply state to choose a coffee whose temperature is hot. What the **pickBestF** statement does is to translate this into a set of temperatures with positive membership values for the category hot. In our case shown in Fig. 4, this would be translated into the temperatures 60–80 centigrades. For each of the temperatures, the forward-search algorithm would try and optimize the respective program attached, say, $serveCoffee(T_9)$ (serve a coffee at table 9) with the **pickBestF** statement.

For a single linguistic category we define **pickBestF** as

$$\begin{aligned}
&BestDo(\mathbf{pickBestF}(\mathfrak{f} : \gamma, p); p', s, h, \pi, v, pr) \stackrel{def}{=} \\
&\quad \exists u_1, \ldots, u_k. \bigvee_{u \in \{u_1, \ldots, u_k\}} [\mathfrak{f}[s] = u \wedge \mathrm{is}(\mathfrak{f}[s] = u, \gamma)] \wedge \\
&\quad \neg \exists u_j. (\mathfrak{f}[s] = u_j \wedge \mathrm{is}(\mathfrak{f}[s], \gamma) \wedge \\
&\qquad u_j \neq u_1 \wedge \ldots \wedge u_j \neq u_k \wedge \\
&\quad BestDo(p|^{\mathfrak{f}}_{u_1} \mid \cdots \mid p|^{\mathfrak{f}}_{u_k}); p', s, h, \pi, v, pr)
\end{aligned}$$

The intuitive meaning of the above definition is to collect all possible numerical values of a linguistic category as follows: First, we assume that the is predicate holds for k numerical values u_i of the category γ. Then, we "check" if these k values are all values for which $\mathrm{is}(\cdot, \gamma)$ holds. Lastly, we call $BestDo$, replacing the fluent $\mathfrak{f}$ in the program p with any such value (denoted by $p|^{\mathfrak{f}}_{u_i}$).[1] This is analogous to the definition of $BestDo$ for **pickBest** (Eq. 1), where the fluent was replaced with any element of the set τ (cf. Section 3.1). We give the remaining definitions for the complement of a linguistic category and for conjunction and disjunction of several linguistic categories below.

If the expression is the *complement* of a linguistic category we have

$$\begin{aligned}
&BestDo(\mathbf{pickBestF}(\mathfrak{f} : \neg\gamma, p); p', s, h, \pi, v, pr) \stackrel{def}{=} \\
&\quad \exists u_1, \ldots, u_k. \bigvee_{u \in \{u_1, \ldots, u_k\}} [\mathfrak{f}[s] = u \wedge \mathrm{is}_{\complement}(\mathfrak{f}[s] = u, \gamma)] \wedge \\
&\quad \neg \exists u_j. (\mathfrak{f}[s] = u_j \wedge \mathrm{is}_{\complement}(\mathfrak{f}[s], \gamma) \wedge \\
&\qquad u_j \neq u_1 \wedge \ldots \wedge u_j \neq u_k) \wedge \\
&\quad BestDo(p|^{\mathfrak{f}}_{u_1} \mid \cdots \mid p|^{\mathfrak{f}}_{u_k}); p', s, h, \pi, v, pr)
\end{aligned}$$

For a *conjunction* of n linguistic categories we have

$$\begin{aligned}
&BestDo(\mathbf{pickBestF}(\mathfrak{f} : \Gamma_\star, p); p', s, h, \pi, v, pr) \stackrel{def}{=} \\
&\quad \exists u_1, \ldots, u_k. \bigvee_{u \in \{u_1, \ldots, u_k\}} [\mathfrak{f}[s] = u \wedge \mathrm{is}_\star(\mathfrak{f}[s] = u, \Gamma_\star)] \wedge \\
&\quad \neg \exists u_j. (\mathfrak{f}[s] = u_j) \wedge \mathrm{is}_\star(\mathfrak{f}[s], \Gamma_\star) \wedge \\
&\qquad u_j \neq u_1 \wedge \ldots \wedge u_j \neq u_k \wedge \\
&\quad BestDo(p|^{\mathfrak{f}}_{u_1} \mid \cdots \mid p|^{\mathfrak{f}}_{u_k}); p', s, h, \pi, v, pr)
\end{aligned}$$

where $\Gamma_\star$ is an abbreviation for $\Gamma_\star \stackrel{def}{=} \gamma_1 \wedge \cdots \wedge \gamma_n$. For the *disjunction* of n linguistic categories we have

$$\begin{aligned}
&BestDo(\mathbf{pickBestF}(\mathfrak{f} : \Gamma_\oplus, p); p', s, h, \pi, v, pr) \stackrel{def}{=} \\
&\quad \exists u_1, \ldots, u_k. \bigvee_{u \in \{u_1, \ldots, u_k\}} [\mathfrak{f}[s] = u \wedge \mathrm{is}_\oplus(\mathfrak{f}[s] = u, \Gamma_\oplus)] \wedge \\
&\quad \neg \exists u_j. (\mathfrak{f}[s] = u_j \wedge \mathrm{is}_\oplus(\mathfrak{f}[s], \Gamma_\oplus) \wedge \\
&\qquad u_j \neq u_1 \wedge \ldots \wedge u_j \neq u_k) \wedge \\
&\quad BestDo(p|^{\mathfrak{f}}_{u_1} \mid \cdots \mid p|^{\mathfrak{f}}_{u_k}); p', s, h, \pi, v, pr)
\end{aligned}$$

$\Gamma_\oplus$ is an abbreviation for $\Gamma_\oplus \stackrel{def}{=} \gamma_1 \vee \cdots \vee \gamma_n$. By the above definitions we provide our new **pickBestF** allowing to specify the argument choice in terms of a fuzzy expressions for a single fuzzy fluent based on the existing $BestDo$ statements for standard sets. The idea is to branch over all fluent values for which the fuzzy expression holds in the resulting non-deterministic choice of action statement having all occurrences of $\mathfrak{f}$ replaced by the respective value from the respective fuzzy set.

4.2 Fuzzy Expressions in the Reward Function

As a second way to increase the naturalness of specifying programs for decision-theoretic planning we introduce a means

[1] Note that for readability we only use $\mathfrak{f}$ to refer to a fuzzy fluent. $\mathfrak{f}[s]$ denotes the fluent $\mathfrak{f}$ with its situation argument being restored which we need to determine its value in a particular situation.

to use fuzzy expression in the reward function. We propose a statement **fcase** that modifies the reward according to a fuzzy expression, i.e., a single fuzzy category, the complement of a single fuzzy category, the conjunction of several fuzzy categories and the disjunction of multiple categories (all for the same fuzzy fluent $\mathfrak{f}$).

The **fcase** statement distinguishes the four above cases and handles them according to the following definitions. For a single linguistic category

$$\begin{aligned}\mathbf{fcase}(\mathfrak{f},\gamma,r) = reward \stackrel{def}{=}\\ \mathrm{is}(\mathfrak{f}[s],\gamma) \wedge (reward = r) \vee\\ \neg\mathrm{is}(\mathfrak{f}[s],\gamma) \wedge (reward = 0)\end{aligned}$$

For the complement of a single linguistic category

$$\begin{aligned}\mathbf{fcase}(\mathfrak{f},\neg\gamma,r) = reward \stackrel{def}{=}\\ \mathrm{is}_{\complement}(\mathfrak{f}[s],\gamma) \wedge (reward = r) \vee\\ \neg\mathrm{is}_{\complement}(\mathfrak{f}[s],\gamma) \wedge (reward = 0)\end{aligned}$$

For the conjunction of n linguistic categories

$$\begin{aligned}\mathbf{fcase}(\mathfrak{f},\gamma_1 \wedge \ldots \gamma_n,r) = reward \stackrel{def}{=}\\ \mathrm{is}_{\star}(\mathfrak{f}[s],\gamma_1,\ldots,\gamma_n) \wedge (reward = r) \vee\\ \neg\mathrm{is}_{\star}(\mathfrak{f}[s],\gamma_1,\ldots,\gamma_n) \wedge (reward = 0)\end{aligned}$$

For the disjunction of n linguistic categories

$$\begin{aligned}\mathbf{fcase}(\mathfrak{f},\gamma_1 \vee \ldots \gamma_n,r) = reward \stackrel{def}{=}\\ \mathrm{is}_{\oplus}(\mathfrak{f}[s],\gamma_1,\ldots,\gamma_n) \wedge (reward = r) \vee\\ \neg\mathrm{is}_{\oplus}(\mathfrak{f}[s],\gamma_1,\ldots,\gamma_n) \wedge (reward = 0)\end{aligned}$$

4.3 Walking through an Example

To illustrate the extensions proposed above, consider the following example in our Diner Domain: The restaurant has several waitron robots and we need to specify the control program for one of them. Assume the robot is responsible for the tables located in the corners of the room. Let the position of the tables be composed of their x and y cell-coordinates. In terms of a linguistic description, we might then say that the robot needs to serve tables that are in the left or the right part of the room and that are in the front or the back part of the room. The robot can take orders for coffee or meals from any of the tables it needs to serve. Assume the robot has a (finite) list of orders in its world model, each with a number, the table it came from and the temperature the meal was served with. The individual properties of those orders can be retrieved via respective functions, where order_i is used to refer to the order number i. The serving temperature is zero for as long as an order has not been served.

Writing a program for such an agent includes letting the robot choose which table to serve in which order. Using decision-theoretic planning, we can specify an optimization theory by means of a reward function. With our newly introduced **pickBestF** statement we can write a control program in a very straight-forward manner as given in Alg. 2.

Let us assume that the reward is computed by giving a negative amount for any open order (i.e. any order that has not been served yet) and by giving a positive amount for food being served with a high temperature. We can use the newly introduced **fcase** statement to specify such a reward function with linguistic terms as follows. For simplicity, we limit ourselves to a list of only two orders.

Algorithm 2: Decision-theoretic planning in READYLOG for serving a room with fuzzy argument choice

```
1  proc serve_room
2      navigate(counter);
3      while haveOpenOrder(room) do
4          pickBestF(posX, left ∨ right ) {
5              pickBestF(posY, front ∨ back ) {
6                  tableWithOpenOrder(table, posX, posY);
7                  pickBestF(mealTemp, luke ∨ hot ) {
8                      mealWithTempReady(meal, mealTemp);
9                      load_meal(meal,tray);
10                     bring_meal(tray,table);
11                     serve_meal(tray,table); } } }
12         navigate(counter);
13     endwhile
14 endproc
```

$$\begin{aligned}reward(s) = r \stackrel{def}{=}\\ r = \mathrm{numOpenOrders(s)} \cdot (-100) +\\ \mathbf{fcase}(serveTemp(\mathrm{order}_1,s),\mathsf{hot},100) +\\ \mathbf{fcase}(serveTemp(\mathrm{order}_2,s),\mathsf{hot},100) +\\ \mathbf{fcase}(serveTemp(\mathrm{order}_1,s),\mathsf{luke},10) +\\ \mathbf{fcase}(serveTemp(\mathrm{order}_2,s),\mathsf{luke},10)\end{aligned}$$

The fuzzy fluent $serveTemp$ returns the temperature at which a meal was served.

Let us assume, the robot has orders from tables T_9 and T_1. For simplicity we assume there is no coffee and only one meal to order hence both tables may be served with the same meal. The robot finds two meals M_1 and M_2 ready to serve on the counter with temperatures of 54 and 74 centigrades respectively. If the robot uses the above program it yields an execution trace as follows.

The first **pickBestF** statement has a disjunction as its fuzzy expression. Hence, we apply the corresponding $BestDo$ definition. That is, by means of the existential quantifiers we collect those u_i (and only those!) for which the predicate $\mathrm{is}_{\oplus}(u_i,\mathsf{left},\mathsf{right})$ is true. Using the $\mathfrak{F}$ definition for the posX fuzzy fluent we find six values, namely $1,2,3,7,8,9$. Similarly for the second **pickBestF**-statement we collect possible y-coordinates $1,2,3,7,8,9$. Using the $BestDo$ definition, we replace in the body of the **pickBestF** statement the variables posX and posY by any of the available values. For each combination we check whether there is a table with an open order at that position with the predicate $\mathrm{tableWithOpenOrder}$. The only positions for which this is true are T_1 and T_9. For those two tables we continue with the program, i.e. we do another **pickBestF**, now for the fluent $\mathrm{mealTemp}$. Again, us-

Table 1: Table of possible courses of action with their corresponding reward.

serving order	serve-temp1	serve-temp2	total reward
(M_1,T_1),(M_2,T_9)	51	62	120
(M_1,T_9),(M_2,T_1)	48	59	20
(M_2,T_1),(M_1,T_9)	71	42	110
(M_2,T_9),(M_1,T_1)	68	39	110

ing the *BestDo* definition for a disjunctive fuzzy expression (hot $\vee$ luke) we collect a set of values to replace the fluent mealTemp in the remaining program. In our example this is the set $\{45,\ldots,80\}$ as per our specification of the fuzzy sets $\mathfrak{F}$ for luke and hot (cf. Sect. 2). Hence, we consider to execute the sequence inside the innermost **pickBestF** for any combination of existing table positions with open orders in the areas that our robot has to serve, each with any of the meals available with temperatures from the set $\{45,\ldots,80\}$ that we have from our *BestDo* definition. First we check whether a meal with a given temperature is ready on the counter by the predicate mealWithTempReady. Only if this is the case, we attempt to load the meal, bring it to a table and serve it. Starting from the initial situation as given above, from the sets constructed by our *BestDo* definitions for **pickBestF** by means of the two predicates tableWithOpenOrder and mealWithTempReady what remains for the innermost program part are for the position $(8,8)$ and $(2,2)$, each in combination with a meal of either 54 or 74 centigrades temperature.

Starting at the counter, we need 12 steps to reach table T_9 and 6 steps to reach table T_1. This means, a meal cools down by $12 \cdot 5/10 = 6$ centigrades when it is being delivered to T_9 and $6 \cdot 5/10 = 3$ when it is being delivered to T_1. With the two orders to serve and two meals to pick from for each we are left with four courses of action, shown with their reward in Tab. 1. The reward for the course of actions essentially depends on the temperature that each meal is being served at. For meals being served with luke temperature the agent receives a reward of 10, for those being served hot it is rewarded with 100. The most rewarding situation is reached with first serving table T_1 with meal M_1, and then delivering M_2 to T_9. This yields a total reward of 120. The policy returned for the agent to execute then is navigate(counter), tableWithOpenOrder(T_9, 8, 8), mealWithTempReady(M_1, 74), load_meal(M_1, tray), bring_meal(tray, T_9), serve_meal(tray, T_9), navigate(counter), tableWithOpenOrder(T_1, 2, 2), mealWithTempReady(M_2, 54), load_meal(M_2, tray), bring_meal(tray, T_1), serve_meal(tray, T_1), navigate(counter).

Our newly introduced constructs allow for a seamless integration of linguistic notions in decision-theoretic planning in agent programs. The agent designer can use fuzzy expressions both, to specify the set of values to pick from for the non-deterministic choice of argument and to specify portions of the reward function that is used as the underlying optimization theory in decision-theoretic planning.

5 Discussion

In this paper we proposed an extension to READYLOG which combines fuzzy fluents and decision-theoretic planning. Fuzzy fluents are fluents that have a membership function attached. With this function, it can be checked whether or not a fluent value belongs to the linguistic category in question. In our previous work, we defined a predicate "is" to test this. With this predicate, we can handle negation, conjunction and, disjunction, respectively, of linguistic terms. The decision-theoretic extension of GOLOG implements a forward-search value iteration algorithm and allows to optimize non-deterministic choices of action or arguments w.r.t. a given reward function. The search for an optimal policy can be guided by a GOLOG program to restrict the search space. Our practical experiences with programming robots with READYLOG shows that, in particular, non-deterministic choices of actions and arguments are very useful when specifying the behavior of a robot or agent in a flexible way. For the non-deterministic choice of arguments, the programmer has to give a finite domain from which the program arguments for computing the optimal policy are evaluated. This can be a cumbersome process.

In this work, we extend the non-deterministic argument choice such that it can handle simple fuzzy fluent formulas. This facilitates the specification of the argument set in **pickBest** statements. To this end, we introduced and defined a statement **pickBestF** that translates the values for which the given fuzzy fluent formula holds as an argument set for the ordinary **pickBest** statement. Further, we introduced a statement **fcase** which allows to use simple fuzzy fluent formulas in the reward function of the forward-search value iteration algorithm. The programmer can make use of linguistic terms and fuzzy categories when assigning rewards to preferred world situations. We showed the use of the new constructs by an example from the Diner Domain, where a waitron agent has to find an optimal schedule to serve coffee or dishes to its customers.

Our further steps are as follows. So far, we did not allow arbitrary formulas over fuzzy fluents yet. Enabling such formulas is not as easy because, for example, in fuzzy logic the excluded middle does not always hold. We will look into possible realizations of more complex formulas. Furthermore, broadening the application of linguistic terms in planning to planning with preferences is on our agenda. Here we want to investigate how our work can be married with the work of [Fritz and McIlraith, 2006; Bienvenu *et al.*, 2011] who compile modal logic preference formulas into GOLOG programs Similarly, we will have a look into [Finzi and Pirri, 2004; Cesta *et al.*, 2011] who use temporal interval planning to solve scheduling problems.

References

[Bienvenu *et al.*, 2011] Meghyn Bienvenu, Christian Fritz, and Sheila A McIlraith. Specifying and computing preferred plans. *Artificial Intelligence*, 175(7):1308–1345, 2011.

[Boutilier *et al.*, 2000] Craig Boutilier, Ray Reiter, Mikhail Soutchanski, and Sebastian Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Proc. AAAI-00*, pages 355–362. AAAI Press, 2000.

[Cesta *et al.*, 2011] Amedeo Cesta, Simone Fratini, Andrea Orlandini, Alberto Finzi, and Enrico Tronci. Flexible plan verification: Feasibility results. *Fundamenta Informaticae*, 107(2):111–137, 2011.

[De Giacomo *et al.*, 2000] G. De Giacomo, Y. Lésperance, and H. Levesque. ConGolog, A concurrent programming language based on situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

[Ferrein and Lakemeyer, 2008] Alexander Ferrein and Gerhard Lakemeyer. Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems, Special Issue on Semantic Knowledge in Robotics*, 56(11):980–991, 2008.

[Ferrein *et al.*, 2008] A. Ferrein, S. Schiffer, and G. Lakemeyer. A fuzzy set semantics for qualitative fluents in the situation calculus. In Caihua Xiong, Honghai Liu, Yongan Huang, and Youlun Xiong, editors, *Proceedings of the International Conference on Intelligent Robotics and Applications*, volume 5314 of *Lecture Notes in Computer Science*, pages 498–509. Springer, 2008.

[Ferrein, 2010] Alexander Ferrein. Robot controllers for highly dynamic environments with real-time constraints. *Künstliche Intelligenz*, 24(2):175–178, 2010.

[Finzi and Pirri, 2004] Alberto Finzi and Fiora Pirri. Flexible interval planning in concurrent temporal golog. In *Working notes of the 4th international cognitive robotics workshop*, 2004.

[Fritz and McIlraith, 2006] Christian Fritz and Sheila A McIlraith. Decision-theoretic golog with qualitative preferences. In *KR*, pages 153–163, 2006.

[Grosskreutz and Lakemeyer, 2001] Henrik Grosskreutz and Gerhard Lakemeyer. On-line execution of cc-Golog plans. In Bernhard Nebel, editor, *Proc. IJCAI-01*. Morgan Kaufmann, 2001.

[Grosskreutz, 2000] H. Grosskreutz. Probabilistic projection and belief update in the pgolog framework. In *CogRob-00*, pages 34–41. ECAI-00, 2000.

[Levesque *et al.*, 1997] H. Levesque, R. Reiter, Y. Lésperance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.

[McCarthy, 1963] J. McCarthy. Situations, actions and causal laws. Technical report, Stanford University, 1963.

[Reiter, 2001] R. Reiter. *Knowledge in Action*. MIT Press, 2001.

[Schiffer *et al.*, 2006] Stefan Schiffer, Alexander Ferrein, and Gerhard Lakemeyer. Football is coming home. In *Proceedings of the 2006 International Symposium on Practical Cognitive Agents and Robots (PCAR'06)*, pages 39–50, New York, NY, USA, November 27-28 2006. ACM.

[Schiffer *et al.*, 2011] Stefan Schiffer, Alexander Ferrein, and Gerhard Lakemeyer. Fuzzy representations and control for domestic service robots in golog. In Sabina Jeschke, Honghai Liu, and Daniel Schilberg, editors, *Proceedings of the 4th International Conference on Intelligent Robotics and Applications (ICIRA 2011)*, volume 7102 of *Lecture Notes in Computer Science*, pages 241–250. Springer, 2011.

[Schiffer *et al.*, 2012] Stefan Schiffer, Alexander Ferrein, and Gerhard Lakemeyer. CAESAR: An Intelligent Domestic Service Robot. *Journal of Intelligent Service Robotics*, 5(Special Issue on Artificial Intelligence in Robotics: Sensing, Representation and Action):259–273, 2012.