Fuzzy Representations and Control for Domestic Service Robots in Golog

Stefan Schiffer, Alexander Ferrein, and Gerhard Lakemeyer

Knowledge Based Systems Group
RWTH Aachen University, Aachen, Germany
{schiffer,ferrein,gerhard}@cs.rwth-aachen.de

Abstract. In the RoboCup@Home domestic service robot competition, complex tasks such as "get the cup from the kitchen and bring it to the living room" or "find me this and that object in the apartment" have to be accomplished. At these competitions the robots may only be instructed by natural language. As humans use qualitative concepts such as "near" or "far", the robot needs to cope with them, too. For our domestic robot, we use the robot programming and plan language Readylog, our variant of Golog. In previous work we extended the action language Golog, which was developed for the high-level control of agents and robots, with fuzzy concepts and showed how to embed fuzzy controllers in Golog. In this paper, we demonstrate how these notions can be fruitfully applied to two domestic service robotic scenarios. In the first application, we demonstrate how qualitative fluents based on a fuzzy set semantics can be deployed. In the second program, we show an example of a fuzzy controller for a follow-a-person task.

1 Introduction

Classical applications for approaches to cognitive robotics and reasoning about actions are delivery tasks, where the robot should deliver a letter or fetch a cup of coffee. In these domains, it becomes obvious that solving such tasks deploying reasoning and knowledge representation is superior to, say, reactive approaches in terms of flexibility and expressiveness. An even more advanced application domain is RoboCup@Home [13, 14]. As a distinguished league under the roof of the RoboCup federation the robots have to fulfil complex tasks such as "*Lost&Found*", "*Fetch&Carry*", or "*WhoIsWho*" in a domestic environment. In the first tasks the robot has to remember and to detect objects, which are hidden in an apartment, or has to fetch a cup of coffee from, say, the kitchen and bring it to the sitting room, while in the latter the robot needs to find persons and recognise their faces. The outstanding feature of these applications is that they require integrated solutions for a number of sub-tasks such as safe navigation, localisation, object recognition, and high-level control (e.g. reasoning). A particular complication is that the robot may only be instructed by means of natural interaction, e.g. speech or gestures. Human-robot interaction is hence

largely based on natural language. For example, in the *Fetch&Carry* task it is allowed to help the robot with hints like "The teddy is near the TV set".

Humans make frequent use of qualitative concepts like *near* or *far*, as the example shows. It would be desirable that the robot could interpret these concepts and cope with them. When reasoning techniques are deployed to come up with a problem solution for these domestic tasks, also these mechanisms need to be able to cope with those qualitative concepts. But even as logic-based reasoning approaches make inherently use of qualitative concepts, the rest of the complex robot architecture does not. Hence, one needs to bridge the gap between the qualitative high-level control and the quantitative robot control system.

In this paper, we show how this gap can be bridged for domestic robot applications. We extended the logic-based high-level robot programming and plan language Readylog [4] with so-called *qualitative fluents* describing properties of the world based on fuzzy set theory [5] and integrated fuzzy control techniques into the robot control language [6]. This enables us (1) to map qualitative predicates to quantitative values based on a well-defined semantics, and (2) to combine fuzzy control and logic-based high-level control. In the sequel, we show how these concepts can be used beneficially to formulate compact solutions for tasks such as *Fetch&Carry*. While we only give a preliminary specification here, for our future work we aim at deploying these programs to our domestic robot platform, which participated successfully at RoboCup@Home competitions in the past.

The rest of this paper is organised as follows. In Section 2, we give a brief introduction to the robot programming and planning language Readylog and the situation calculus, which Readylog is based on. We recapitulate previous work on integrating fuzzy sets and fuzzy control structures into Golog in Section 3, before we show our qualitative domain description in Section 4. In particular, we define necessary qualitative predicates for the domestic service robotics domain and define fuzzy control structures to enable the robot to cope with qualitative predicates. We conclude with Section 5.

2 The Situation Calculus and Golog

The Situation Calculus [10] 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 dependent *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 situation s. The constant S_0 denotes the initial situation, i.e. the situation where no actions have yet occurred. We abbreviate the expression $do(a_n, \ldots do(a_1, S_0) \ldots)$ with $do([a_1, \ldots, a_n], S_0)$.

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. The third sort of the situation calculus is the sort *action*. For each action one has to specify a *precondition axiom* stating under which conditions it

is possible to perform the respective action and *effect axioms* formulating how the action changes the world in terms of the specified fluents. An action precondition axiom has the form $Poss(a(\boldsymbol{x}), s) \equiv \Phi(\boldsymbol{x}, s)$ where the binary predicate $Poss \subseteq action \times situation$ denotes when an action can be executed, and $\boldsymbol{x}$ stands for the arguments of action a. After having specified when it is physically possible to perform an action, it remains to state how the respective action changes the world. This is done by so-called successor state axioms [11].

ReadyLog [4] is our variant of Golog [9] and also makes use of Reiter's BATs as described above. The aim of designing the language ReadyLog was to create a Golog dialect which supports the programming of the high-level control of agents or robots in dynamic real-time domains such as domestic environments or robotic soccer. ReadyLog borrows ideas from [1, 3, 7–9] and features the following constructs: (1) sequence $(a; b)$, (2) non-deterministic choice between actions $(a|b)$, (3) solve a Markov Decision Process (MDP) ($solve(p, h)$, p is a Golog program, h is the MDP's solution horizon), (4) test actions ($?(c)$), (5) event-interrupt ($waitFor(c)$), (6) conditionals ($if(c, a_1, a_2)$), (7) loops ($while(c, a_1)$), (8) condition-bounded execution ($withCtrl(c, a_1)$), (9) concurrent execution of programs ($pconc(p_1, p_2)$), (10) probabilistic actions ($prob(val_{prob}, a_1, a_2)$), (11) probabilistic (offline) projection ($pproj(c, a_1)$), and (12) procedures ($proc(name(parameters), body)$). The idea of Golog to combine planning with programming was accounted for in ReadyLog by integrating decision-theoretic planning; only partially specified programs which leave certain decisions open, which then are taken by the controller based on an optimization theory, are needed.

A nice feature of Golog and ReadyLog is that its semantics is based on the situation calculus. That means that both languages have a formal semantics and properties of programs can be proved formally. We refer the interested reader to [4] for the complete formal definition of the language. Golog languages come with run-time interpreters usually programmed in Prolog. Also, a ReadyLog implementation is available in Prolog.

3 Qualitative Fluents and Fuzzy Controllers in Golog

In this section, we briefly go over our previous work on integrating fuzzy fluents and fuzzy controllers into Golog. For technical details we refer to [5, 6].

3.1 Fuzzy Fluents

The essence of qualitative representations is to find appropriate equivalence classes for a number of quantitative values and to group them together in these qualitative classes. Fuzzy set theory seems appealing as it avoids sharp boundaries of the classes: a quantitative value can be, for instance, in two classes at the same time, the transition between two neighbouring classes can be designed as being smooth. This characteristic can avoid problems every roboticist already

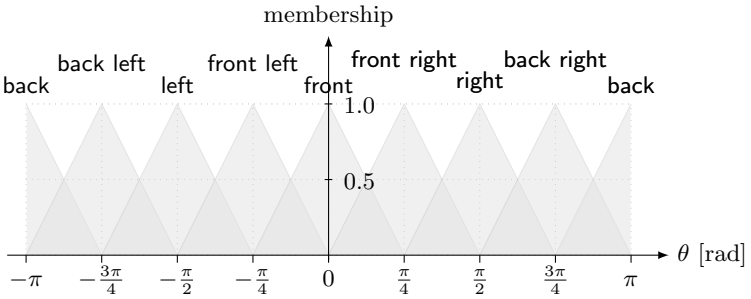


Fig. 1. Membership function for qualitative orientation at level 3

has experienced: sensor values oscillate between two categories resulting in awkward behaviour of the robot.

Our formalisation of fuzzy fluents is based on the idea to extend ordinary functional fluents with a degree of membership to a certain qualitative category. To use these fluents, one simply defines the different categories and membership values in the domain specification. An example for the orientation fluent is given in Fig. 1. What is further needed in order to do reasoning with these kinds of fluents, is a routine that restores a quantitative value from a qualitative category, that is, to *defuzzify* a category. In [5], we formalise a *centre-of-gravity defuzzifier* in the situation calculus. However, other defuzzifiers known from fuzzy set theory can easily be used as well.

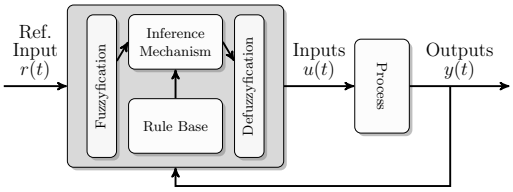
For illustrating reasoning with qualitative positional information consider the following simple example. A robot is situated in a one dimensional room with a length of ten metric units. To keep things simple, we restrict ourselves to integer values for positions in the following. We have one single action called $\text{gorel}(d)$ denoting the relative movement of d units of the robot in its world. This action is always possible, i.e. $Poss(\text{gorel}(d), s) \equiv \top$. The action has impact on the fluent *pos* which denotes the absolute position of the robot in the world. The successor state axiom of *pos* is defined as

$$pos(do(a, s)) = y \equiv a = \text{gorel}(d) \wedge y = pos(s) + d \vee a \neq \text{gorel}(d) \wedge y = pos(s).$$

There is a table in the robot's world, its position is defined by the macro $pos_{table} = p \doteq p = 9$. In the initial situation, the robot is located at position 0, i.e. $pos(S_0) = 0$. We want to evaluate the robot's position and its distance to the table. Therefore we define a functional fluent *dist* which returns the distance between the robot and the table:

$$dist(do(a, s)) = d \equiv \exists p_1.pos_{table} = p_1 \wedge \exists p_2.pos(do(a, s)) = p_2 \wedge d = p_1 - p_2.$$

We partition the distance in categories close, medium, and far, and introduce qualitative categories for the position of the robot as back, middle, and front. We give the (fuzzy) definition of those categories below, where we use (u_i, μ_i) as an abbreviation for $u = u_i \wedge \mu = \mu_i$. For instance, the fuzzy categories for the



(a) Generic fuzzy controller

fuzzy_controller(
if ϕ **then** $assign(\mathfrak{f}, c_k); \cdots$;
if ψ **then** $assign(\mathfrak{g}, c_l)$;
default$(assign(\mathfrak{f}, c_n); assign(\mathfrak{g}, c_m)))$

(b) ReadyLog fuzzy controller

Fig. 2. Generic architecture and ReadyLog statement for a fuzzy controller

position of the robot in the world can be defined as

$$\begin{aligned}&\mathfrak{F}(position, u, \mu_u) \equiv\\ &\quad (position = \mathsf{back} \supset (0, 0.25) \vee (1, 0.75) \vee (2, 0.75) \vee (3, 0.25)) \vee\\ &\quad (position = \mathsf{middle} \supset (3, 0.25) \vee (4, 0.75) \vee (5, 0.75) \vee (6, 0.25)) \vee\\ &\quad (position = \mathsf{front} \supset (6, 0.25) \vee (7, 0.75) \vee (8, 0.75) \vee (9, 0.5)),\end{aligned}$$

Similary, the orientation relation can be defined. Note that $\mathfrak{F}(c, u, \mu)$ is our first-order definition of a fuzzy set for a linguistic category c with μ_u being the membership value of the quantitative value u denoting to which degree u belongs to c (see [5] for the complete axiomatization).

The robot can move around in integer steps. Restricting to integers presupposes that we need to use an altered version $cog'(c)$ of the centre-of-gravity defuzzyfier formula: $cog'(c) \doteq \lfloor cog(c) \rfloor$.

Suppose now that the robot's control program contains the action gorel(far) mentioning the qualitative term far. At which position will the robot end up in situation $s = do(\text{gorel}(\mathsf{far}), S_0)$? The qualitative category has to be handled in the successor state axiom. We need to apply the function $cog'(c)$ to the qualitative term which yields always a quantitative representative. The extended definition of the successor state axiom then looks as follows:

$$\begin{aligned}&pos(do(a, s)) = y \equiv\\ &\quad a = \text{gorel}(d) \wedge ((\exists d', c, u, \mu_u.\mathfrak{F}(c, u, \mu_u) \wedge c = d \wedge d' = cog'(d)) \vee\\ &\quad (\neg\exists c, u, \mu_u.\mathfrak{F}(c, u, \mu_u) \wedge d' = d)) \wedge \;\; y = pos(s)+d' \vee a \neq \text{gorel}(d) \wedge y = pos(s).\end{aligned}$$

Note that we rely on the completeness of the specification of the membership function here, so that if d is a linguistic term there always is an entry in the membership function for that d. Otherwise we could end up computing y as the sum of a real and category. Our formalization yields $\text{is}(pos(do(\text{gorel}(\mathsf{far}), S_0)), \mathsf{front})$, i.e. the robot ends up in the front part of its world after executing gorel(far). Note that "is" denotes a predicate in our framework to query fuzzy fluent values. It will be also used in Algorithms 1 and 2.

3.2 Fuzzy Controller in Readylog

Fig. 2 shows a schematic fuzzy controller. The quantitative sensor values $y(t)$, together with some reference input $r(t)$, which describes the vital state of the

system, need to be fuzzified, i.e. the membership to a certain class needs to be determined. The *Inference Mechanism* uses these fuzzified input values together with a rule base of fuzzy rules to select the appropriate control output. The output as such uses fuzzy categories and thus must be defuzzified to serve as an input $u(t)$ for the real world (the control output). The output of the real world process serves as the sensor input for the next control step.

To map this into Readylog, we introduce a statement **fuzzy_controller** which takes a rule base as input and returns the control output (cf. also [6]). We give the general form of this statement in Fig. 2(b).
A fuzzy rule base in Readylog is interpreted as follows. Each matching fuzzy rule will be replaced by its consequence, i.e. a special assignment statement, while non-matching ones contribute *nil*. The assignment statement $assign(\mathfrak{f}, c)$ used in the controller is a Readylog action which assigns the qualitative category c to the fuzzy fluent $\mathfrak{f}$. As defuzzifier, we use the centre-of-gravity (*cog*). Depending on the assigned output category, control actions can be sent to the actuators. The condition of a rule can be a complex formula over fuzzy fluents stating for example: *is the object close and very close?* Sometimes, it may happen that no given rule in a controller block matches at all, nevertheless some output would be required. We therefore define an additional statement **default**$(assign(\mathfrak{f}, c); \ldots)$, which is interpreted in case the control output was the *nil* action after evaluating the rule base. This gives the basic idea how a rule base is encoded in Readylog. We left out the formal definition of the construct. It can be found in [6].

4 Applications in a Domestic Service Robotics Domain

In this section we give two examples for using fuzzy fluents and fuzzy controllers in the domestic robot domain. We start with a brief description of the tasks. Before we show the example Readylog programs, we define the required distance and orientation relations.

4.1 A Domestic Service Robotics Domain (RoboCup@Home)

In the RoboCup@Home competition *service and assistive robot technology* that is highly relevant for future personal domestic applications should be demonstrated [12]. In the competition, the robots have to fulfil tasks such as:

- *FollowMe!*: the robot has to follow a human through the apartment;
- *Fetch&Carry*: a human names known objects and the robot needs to fetch them. The human may give hints such as: "The teddy is near the TV";
- *Walk'n'Talk*: in a guidance phase, a human instructor leads the robot around in an apartment and tells it certain landmarks such as "kitchen table", "TV set", or "fridge". In a second phase the robot is instructed to navigate to some of these just learnt places.

The rules of the RoboCup@Home competition state that a robot—to be successful in the competition—is to be endowed with a certain set of basic abilities,

like navigation, person and object recognition, and manipulation. Furthermore, fast and easy calibration and setup is essential, as the ultimate goal is to have a robot up and running out of the box. Also, human-robot interaction has to be achieved in a natural way, i.e. interacting with the robot is allowed only using natural language (that is by speech) and gesture commands. As mentioned in the introduction, humans tend to make use of qualitative concepts such as *near* or *far*. With introducing suitable qualitative concepts, we bridge the gap between human and robot representations of domestic environments.

But not all parts of the solution of a domestic task require deliberation. For some decisions simple reactive controllers are sufficient. However, these reactive mechanisms also need to understand qualitative concepts. Here, we can make use of our embedding of fuzzy controllers in Readylog. In the next sections, we show some specification examples.

4.2 Qualitative Representations for Domestic Environments

One very important form of interaction between a human and a robot in the RoboCup@Home domain is to give the robot some hints where objects might be located. Based on Clementini, Felici, and Hernandez [2], we deploy qualitative representations for positional and directional information that can be used to instruct the robot. The position of a *primary object* is represented by a pair of distance and orientation relations with respect to a *reference object*. Both relations depend on a so-called *frame of reference* which accounts for several factors like the size of objects and different points of view.

In the domestic settings we can define different distance relations according to: (1) external references such as the maximal size of the apartment: *"The plant is at the far end of the corridor"*; (2) intrinsic references used in relating objects to each other such as room or table: *"The cup is on the table close to the plate"* vs. *"The teddy is close to the TV"*; and (3) an appropriate distance system. In our domestic environment we suggest to make finer distinctions in the neighbourhood of the reference object than in the periphery. Hence, we can distinguish the scales $\textit{dist-scale} \in \{\text{apartment}, \text{room}, \text{object}(o)\}$, where object o refers to objects such as *table*, or *bookshelf*.

Hence, we must provide a procedure *analyseHint*, which takes a hint given by the human instructor and distills the position of the object, the frame of reference as well as the scale from that hint. For instance: (a) *"The plant is far on the left side of the corridor"*; the primary object is the plant, the point of view is the view point of the robot, the distance scale is set to the size of the corridor. (b) *"The cup is on the table close to the plate"*; the primary object is the cup, the reference object is the plate, the distance scale is set to the size of the table. No orientation relation is given. (c) *"The teddy is close to the TV"*; the primary object is the teddy, the reference object is the TV, the distance scale should be set to the size of the room where the TV is located. Again, no orientation relation is given.

With this procedure at hand, we can adopt our fuzzy fluents for the qualitative distance and orientation. The membership function for the orientation

fluent was given in Fig. 1. We can define the membership function for distance in a similar way. In the next section, we give an idea of how these fluents can be used for programming the robot.

4.3 Qualitative Notions in High-level Programs

Now that we have proposed an initial modelling of qualitative representations of positional information in a domestic setting we show how we can make use of these representations within our existing high-level control mechanism. Algorithm 2 shows a slightly abstracted version of a Readylog control program for the *Fetch&Carry* task.

The procedure *fetch_and_carry* takes the object that should be fetched and a user hint as input. At first, the action *analyseHint* is executed. This is a complex action which involves natural language processing. From the user phrase, the frame of reference for orientation and distance as well as the distance scale is extracted (as pointed out in the previous section). The action's effect axioms are changing fluent values for the fluents describing the orientation's frame of reference, the distance system, the distance scale, the distance's frame of reference as well as the qualitative position of the reference object. The next statement in the program is a so-called "*pick*" statement (π) which is used to instantiate the free variables in the logical formula in the next test action (denoted by the "?"). The whole construct can be seen as an existential quantifier, and the effect is that the variables *pos*, for_θ, for_{dist} are bound. The next step is to call the search routine with these parameters. The search involves the activation of decision-theoretic planning (*solve*) at a position where the object is meant to be according to the user's hint. The position is defuzzified, taking the frame of reference information into account. That is, the position based on the distance scales and the quantitative orientations given the points of view etc. can now be calculated. The action *lookForObject* again is a complex action which actually tries to seek the object.

4.4 Domestic Golog Fuzzy Controllers

As detailed in Sect. 3.2 we integrated fuzzy controllers in Golog in [6]. If (a part of) a task does not require high-level decision making (decision-theoretic planning as used in the previous section), but can instead be solved with a reactive mechanism it may still be convenient to make use of the qualitative representations. One example in the domestic setting is the "*FollowMe!*" test. The control of the follow behaviour can be modelled quite straight-forwardly.

In the following we show a simple rule base that can be used to solve the *FollowMe!* task. The rule base for this test could look like Alg. 1. As we stated in Sect. 3, a rule base consists of a number of if-then rules where both, the antecedent and the consequence, mention fuzzy fluents. So, the first rule reads as follows: "*if the distance to the user is close and its speed is slow, then set the robot speed to slow*", the second rule reads "*if the distance to the user is far and its speed is medium, then set the robot speed to fast*", where user is the person

proc follow_me_rulebase
  fuzzy_controller(...;
    if $\text{is}_\star(\text{dist}_{\text{user}}, \text{close}, \text{speed}_{\text{user}}, \text{slow})$ **then** $assign(\text{speed}_{\text{robot}}, \text{slow})$;
    if $\text{is}_\star(\text{dist}_{\text{user}}, \text{far}, \text{speed}_{\text{user}}, \text{medium})$ **then** $assign(\text{speed}_{\text{robot}}, \text{fast})$;
    ...; **default**$(\text{speed}_{\text{robot}}, \text{medium})$)) ;`/* end fuzzy_controller */`
  applySpeed()
endproc

Algorithm 1: A fuzzy controller for the "*FollowMe!*" test

proc fetch_and_carry($object, hint$)
  analyseHint($hint$);
  $\pi(pos, for_\theta, for_{dist}).[ori\_type(for_\theta) \wedge dist\_system(for_{dist}) \wedge$
    $dist\_scale(for_{dist}) \wedge dist\_type(for_{dist}) \wedge object\_pos(pos)]?$;
  search($object$, pos, for_θ, for_{dist})
endproc
proc search($object$, pos, for_θ, for_{dist})
  solve(**while** ¬objectFound **do**
      pickBest($search\_pos$ = defuzzify(pos, for_θ, for_{dist}));
      lookForObjectAt($object$, $search\_pos$);
    endwhile, H) /* *end solve with horizon H* */
  pickup_and_return($object$);
endproc

Algorithm 2: A Readylog program for the "*Fetch&Carry*" test.

to be followed. The $is_\star$ predicate is defined in [6] and denotes the conjunction of the fuzzy fluents $dist_{user}$ and $speed_{user}$. If neither condition applies, the default speed selection is set to medium. Finally, the $speed_{robot}$ fuzzy fluent has to be defuzzified, that is, a quantitative value is calculated for the qualitative class. Then, we can apply the quantitative speed to the robot motors.

By using a fuzzy controller with its simple concept of a set of rules we alleviate the specification of the control. We can use linguistic terms to describe the intended behaviour and leave the details on what values to send to the mid- and low-level modules to our automatic machinery.

5 Conclusions

In this paper, we presented an approach on how high-level robot controllers could deal with qualitative representations for domestic environments. For robot competitions such as RoboCup@Home this is useful, as the robot needs to be instructed by a human operator by natural language. Having qualitative representations in place allows for more human-like instructions as humans tend to use qualitative (spatial) representations such as *far* or *left-of*. In our previous work, we defined qualitative fluents in the situation calculus based on fuzzy sets. This allows us to define qualitative fluents in a well-founded way. Particularly, it gives a semantics to derive quantitative values from qualitative categories and vice versa. Further, we proposed a semantics for fuzzy controller

in Golog. Both, the definition of fuzzy fluents and fuzzy controllers, allows us to write programs mentioning qualitative values in a straight-forward way. For the RoboCup@Home tasks *Fetch&Carry* and *FollowMe!* we showed example implementations, how qualitative representations and fuzzy controllers could be beneficially deployed. While these programs only reflect first ideas of deploying fuzzy fluents and fuzzy controllers in domestic robot applications, we aim at implementing different controllers and programs making use of the fuzzy notions for our future work on our domestic robot platform.

References

1. Boutilier, C., Reiter, R., Soutchanski, M., Thrun, S.: Decision-theoretic, high-level agent programming in the situation calculus. In: Proc. 17th Nat'l Conf. on Artificial Intelligence (AAAI-00). pp. 355–362 (2000)
2. Clementini, E., Felice, P.D., Hernandez, D.: Qualitative representation of positional information. Artificial Intelligence 95(2), 317–356 (1997)
3. De Giacomo, G., Lésperance, Y., Levesque, H.J.: ConGolog, A concurrent programming language based on situation calculus. Artificial Intelligence 121(1–2), 109–169 (2000)
4. Ferrein, A., Lakemeyer, G.: Logic-based robot control in highly dynamic domains. Robotics and Autonomous Systems, Special Issue on Semantic Knowledge in Robotics 56(11), 980–991 (2008)
5. Ferrein, A., Schiffer, S., Lakemeyer, G.: A fuzzy set semantics for qualitative fluents in the situation calculus. In: Proc. Int'l Conf. on Intelligent Robotics and Applications (ICIRA'08), vol. 5314, pp. 498–509. Springer (2008)
6. Ferrein, A., Schiffer, S., Lakemeyer, G.: Embedding fuzzy controllers into golog. In: Proc. IEEE Int'l Conf. on Fuzzy Systems (FUZZ-IEEE-09). pp. 498–509 (2009)
7. Grosskreutz, H.: Probabilistic projection and belief update in the pGOLOG framework. In: Proceedings of the 2nd Cognitive Robotics Workshop (CogRob'00) at the 14th European Conference on Artificial Intelligence (ECAI'2000), pp. 34–41 (2000)
8. Grosskreutz, H., Lakemeyer, G.: cc-Golog – An Action Language with Continuous Change. Logic Journal of the IGPL 11(2), 179–221 (2003)
9. Levesque, H.J., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.B.: Golog: A logic programming language for dynamic domains. J. Logic Program. 31(1-3), 59–84 (1997)
10. McCarthy, J.: Situations, actions and causal laws. TR, Stanford University (1963)
11. Reiter, R.: Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
12. Wisspeintner, T., van der Zant, T., Iocchi, L., Schiffer, S.: Robocup@home: Scientific Competition and Benchmarking for Domestic Service Robots. Interaction Studies. Special Issue on Robots in the Wild 10(3), 392–426 (2009)
13. van der Zant, T., Wisspeintner, T.: Robocup x: A proposal for a new league where robocup goes real world. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup. LNCS, vol. 4020, pp. 166–172. Springer (2005)
14. van der Zant, T., Wisspeintner, T.: Robotic Soccer, chap. RoboCup@Home: Creating and Benchmarking Tomorrows Service Robot Applications, pp. 521–528. I-Tech Education and Publishing (2007)