Problem Decomposition for Behavioural Cloning

Dorian Šuc and Ivan Bratko

Faculty of Computer and Information Sciences, University of Ljubljana, Slovenia
Tržaška 25, 1000 Ljubljana
{dorian.suc, ivan.bratko}@fri.uni-lj.si

Abstract. In behavioural cloning of the human operator's skill, a controller is usually induced directly as a classifier from system's states into actions. Experience shows that this often results in brittle controllers. In this paper we explore a decomposition of the cloning problem into two learning problems: the learning of operator's control trajectories and the learning of the system's dynamics separately. We analyse advantages of such *indirect controllers*. We give characterization of the learner's error that is plausible explanation of why this decomposition approach has empirically proved to be usually superior to *direct* cloning.

1 Introduction

Controllers for dynamic systems can be designed by machine learning using different kinds of information available to the learning system. The idea of *behavioural cloning* (a term introduced by Donald Michie (1993)) is to make use of the operator's skill in the development of an automatic controller. Early work in behavioural cloning was done by Donaldson [8]) and Chambers and Michie [7]. A skilled operator's control traces are used as examples for machine learning to reconstruct the underlying control strategy that the operator executes subconsciously. The goal of behavioural cloning is not only to induce a successful controller, but also to achieve better understanding of the human operator's subconscious skill [13]. Behavioural cloning was successfully used in problem domains as pole balancing, production line scheduling, piloting [11] and operating cranes. These experiments are reviewed in [6].

The usual approach is to induce a control rule as a function $Action = Action(State)$ where $State$ is the state vector of the dynamic system, and $Action$ is the control action to be performed in $State$. The induced function is typically represented by a decision or regression tree. Although successful clones have been induced in the form of trees or rule sets, the following problems have generally been observed with this approach:

- Typically, induced clones are brittle with respect to small changes in the control task.
- The clone induction process typically has low yield: the proportion of successful controllers among all the induced clones is low, typically well below 50%.

R. López de Mántaras, E. Plaza (Eds.): ECML 2000, LNAI 1810, pp. 382–391, 2000.

An approach aiming at rectifying these defeciences, proposed in [14], exploits some results from control theory. This approach considers the system's dynamics and automatic identification of discrete subgoals. It improves both the clones' robustness with respect to changes in the control task, and the yield of the cloning process. However, this approach still has difficulties in domains with significant nonlinearities or where the operator's control plan is not simply expressed in terms of a few discrete subgoals.

In [17] we proposed another approach to behavioural cloning, suitable also for strongly nonlinear domains. The trajectory the operator is trying to follow is generalized separately from the system's dynamics and can be viewed as a *continuous subgoal*. In particular, we do not learn the trajectory in time, but rather the constraints among the state variables in the execution trace. These constraints determine the corresponding desired trajectory to the goal, also for system states other than those explicitly mentioned in the operator's execution trace. Actions that maintain the desired trajectory are computed using knowledge of the system's dynamics, learned by nonlinear function approximators. So the initial learning problem is decomposed into two learning problems: first, the generalisation of the operator's control *trajectory* to areas outside the example trace, and second, the learning of the system's dynamics.

Experiments performed in the crane domain in [17] and in the Acrobot domain [15] demonstrated that this decomposition approach enormously improves the yield of the cloning process and provides a good insight in the operator's subcognitive skill. Qualitative strategy, generalized from the operator's trajectory, is comprehensible and offers a possibility to optimize the operator's control strategy.

In this paper we investigate why the proposed decomposition of the controller induction task into two learning tasks (learn generalized trajectory T and system's dynamics D) works better than the original problem definition (learn $Action = Action(State)$ directly). In other words, why *indirect* controllers are more robust than *direct* controllers.

2 Problem Decomposition and Indirect Controllers

Let us now present the details of the problem decomposition for behavioural cloning investigated in this paper. The construction of a controller by induction from the operator's traces consists of three stages (see Fig. 1):

1. Learn constraints T on operator's trajectories, which can be formally stated as a mapping T:

$$T : States \times States \mapsto \{true, false\} \tag{1}$$

The trajectory constraints are usually represented as a function t, expressing a chosen state variable (*dependent* variable) as a function of the system's state. A suitable chosen dependent variable is one that is most directly affected by the available actions. For example, for the pole-cart sys-

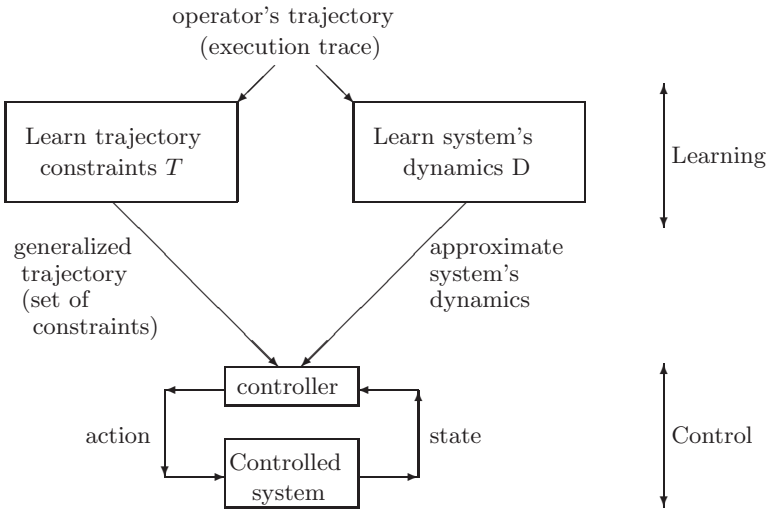

Fig. 1. Indirect controller can be induced from operator's control trace by generalizing the operator's trajectory and learning the system dynamic's model. Both together are used to control the system

tem: $\dot{x} = t(x, \theta, \dot{\theta})$. Constraints T define what will be called a *generalized operator's trajectory*.

2. Learn appropriate model D of the system's dynamics. D can for example be a function that maps system's states and actions into acceleration, eg.: $\ddot{x} = f(x, \dot{x}, Action)$.
3. Once a trajectory model T and dynamics model D have been induced, they are used to construct a controller which works as follows:

   (a) Given a current system state x_k at time point k, $\Delta = \text{diff}(x_k, T)$ denotes the deviation between the state and the generalized operator's trajectory. Usually the deviation is defined simply as the difference between the value of chosen distinguished variable defined by T and its current value.

   (b) Use system's dynamics model D to determine an action A which reduces the deviation Δ in time. For discrete time k, this can be written as:

$$A_k = \arg\min_{A \in \mathcal{A}} \text{diff}(x_{k+1}, T) \tag{2}$$

   where $x_{k+1} = D(x_k, A_k)$ and $\mathcal{A}$ is the set of possible actions. We call this an *indirect controller* as opposed to a *direct controller* that computes action directly as $A_k = A(x_k)$ (see Fig. 2).

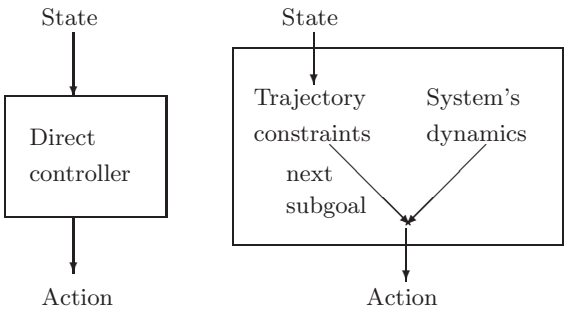

Fig. 2. Direct and indirect controller: on the left is an direct controller, and on the right is indirect controller. Direct controller uses a direct mapping from states to actions, whereas the indirect controller uses the trajectory constraints and the model of the system dynamics to take the action towards the next subgoal

3 Direct and Indirect Controllers

In the following analysis we assume deterministic system's dynamics f and discrete time, i.e. $x_{k+1} = f(x_k, u_k)$, where index k indicates successive sample times t_0, $t_0 + dt$, ..., $t_0 + k\,dt$, x denotes the state vector and u the control action vector.

Controller observes current state x and takes corresponding action u, i.e, it is a function mapping states to control actions: $x \mapsto u$. This controller function can take different forms. In the case of a linear controller it is a product of the error in state $(x - x_g)$ and a matrix K, while in the case of classical approach to behavioral cloning, it is a regression or decision tree. Classic control theory approach to controller design usually involves modeling of the system's dynamics, whereas the usual approach to behavioral cloning does not use the system's dynamics model and learns a direct mapping from states to actions.

To make distinction between the controllers that do and do not use a system's dynamics model and to compare the learnability of such controllers we define two kinds of controllers. *Direct controller* corresponds to a direct mapping from the observed state to the control action. *Indirect controller* uses the system's dynamics model and some intermediate concepts to achieve the goal-directness of the controller and therefore corresponds to an indirect mapping from states to actions. Approaches like BOXES [10], ASE/ACE, [3] and behavioral cloning (reviewed in [6]) learn direct controllers. A controller using the generalized trajectory [16,17] is an indirect controller. The classical linear controller ([5], for example) can also be viewed as an indirect controller. These controllers use the system's dynamics model and intermediate concepts in the form of goal x_g (linear controller) or generalized trajectory. One could argue that a tree is an indirect mapping too, because it maps state x to a generalized state represented by a leaf in the tree and then maps this generalized state to action u. If we are lucky, the

generalized state is an understandable intermediate concept, perhaps a region in a state space where the operator takes similar actions. But the point is that it does not consider the system's dynamics and is not goal (or subgoal) directed.

In the case of a generalized trajectory, the controller is a composite of two functions: the trajectory $f^T : x \mapsto x^T$ and the inverse dynamics $f^D : x \times x^T \mapsto u$. The trajectory function maps the current state x_k to desired state x^T_{k+1} (note that typically x^T_{k+1} is not completely quantified; some of the state components can be omitted or just qualitatively described). The inverse dynamics function maps the current state x_k and the desired next state x^T_{k+1} to action u_k which aims to achieve x^T_{k+1} from x_k.

4 Robustness of Direct and Indirect Controllers against the Learning Error

Here we study and compare the performance of direct and indirect controllers learned from an operator's execution trace $((x^{op}_k, u^{op}_k), k = 0 \dots N)$. Direct controller generalizes the actions, while indirect controller generalizes the trajectory $((x^{op}_k), k = 0 \dots N)$. We will show that a direct controller is prone to the departure from the trajectory and that this departure is likely to happen, due to the learning error of the generalized action.

The operator takes the action u^{op}_k in the current state x^{op}_k that achieves the desired next state x^{op}_{k+1}. The learned controller mimics the operator and takes action u_k that achieves the desired next state with some error ϵ_{k+1}: $x_{k+1} = f(x_k, u_k) = x^{op}_{k+1} + \epsilon_{k+1}$.

Intuitively a direct controller is not likely to be successful, if the operator's action is applied in a state far from the operator's trajectory. This is because a property of many nonlinear dynamics systems is that an action applied far from the operator's trajectory tends to cause the system to move in a different direction than when the same action is applied on the trajectory. Since the generalized action u^D_k is learned with some error, and these errors are often propagated to the next state, the trajectory is likely to diverge from the operator's trajectory.

As opposed to the direct controller, an indirect controller's action u^I_k uses the learned dynamics at the current state $x_k = x^T_k + \epsilon_k$ when aiming to approach the next state on the trajectory x^T_{k+1}. Even if the current state x_k is far from the trajectory, the controller explicitly aims at decreasing the error at the next step. Since the system's dynamics model can be updated online, indirect controller can approach the trajectory even from states not seen in the operator's trace.

4.1 Experiment

Here we compare the robustness of direct and indirect controllers with respect to the learning error on an example of a simple dynamic system control. We consider a deterministic, discrete time dynamic system with the following dynamics:

$$\begin{aligned} x_{k+1} &= x_k + \dot{x}_k \\ \dot{x}_{k+1} &= x_{k+1} + u_k x^2_{k+1} + u_k \end{aligned} \tag{3}$$

A similar system was used in [2] as an example of a nonlinear system that is easily controlled with the optimal linear controller (LQR) in the region near zero, but hard to control elsewhere in the state space.

The task is to control the system to reach and maintain the goal position $x_g = 0.5$ from the start state ($x_0 = \dot{x}_0 = 0$). The criterion of the controller's success is the controller's error c. Here we define c as the sum of absolute errors in x for the first $N = 30$ steps: $c = \sum_{k=0}^{k=N} |x_k - x_g|$. If $|x_k|$ exceeds 100 the error is set to its maximum $c = 100$. Since $x_0 = 0$ and $x_1 = 0$, $c \geq 1$.

We suppose that the operator uses a very simple control rule that does the task with $c = 1.004$:

$$u_k^{op} = \begin{cases} 0.5, \text{if } x_k \leq 0 \text{ and } \dot{x_k} = 0 \\ -0.4, \text{if } x_k > 0 \text{ and } |x_g - x_k| \leq 0.001 \\ -0.4 - 0.001\ sign(x_g - x_k), \text{otherwise} \end{cases} \tag{4}$$

making the system to move approximately along the trajectory $((x_0, \dot{x}_0) \ldots (x_{k+1}^{op}, \dot{x}_{k+1}^{op}) \ldots)$, where:

$$\begin{aligned} x_{k+1}^{op} &= x_k + \dot{x}_k \\ \dot{x}_{k+1}^{op} &= \begin{cases} (x_g - x_k) - \dot{x_k}, \text{if } x_k = 0 \\ 0.09(x_g - x_k), \text{otherwise} \end{cases} \end{aligned} \tag{5}$$

A generalized trajectory is specified by constraints between the current and the next state. Since the control u_k directly affects only $\dot{x}_{k+1}$ (and not x_{k+1}), an appropriate generalized trajectory can be learned as a function mapping from $(x_k, \dot{x}_k)$ to the desired $\dot{x}_{k+1}^T$ and the indirect controller action u_k^I can be computed from the learned system's dynamics. In our experiments with indirect controllers we used a very simple kind of local learning [1,2] to learn (during the control) a local model of the system's dynamics near the current state $(x_k, \dot{x}_k)$: 10 examples $((x_j, \dot{x}_j), u_j, (x_{j+1}, \dot{x}_{j+1}))$ with the smallest quadratic distance $(x_j - x_k)^2 + (\dot{x}_j - \dot{x}_k)^2$ are used for linear regression to compute the parameters a,b,c,d of the local linear model $\dot{x}_{k+1} = ax_k + b\dot{x}_k + cu_k + d$ used at the point $(x_k, \dot{x}_k)$. The points used for local learning are continuously updated during the control, i.e. starting with the set $\{ x_k^{op}, \dot{x_k}^{op}, u_k^{op} \}$ and adding new points during the control. The control u_k^I, aiming to achieve the desired next state on the generalized trajectory is then computed as $u_k^I = (\dot{x}_{k+1}^T - ax_k - b\dot{x}_k - d)/c$.

From the execution trace $((x_k^{op}, \dot{x_k}^{op}, u_k^{op}),\ k = 0, \ldots, N)$, where rule 4 was used, the generalized operator's action u_k^D and the generalized operator's trajectory $\dot{x}_k^T$ can be learned with the learning errors ϵ^D and ϵ^T, i.e. generalization errors due to the generalization of the learning system. So the predicted values of the induced predictors are $u_k^D = u_k^{op} + \epsilon_k^D$ and $\dot{x}_k^T = \dot{x}_k^{op} + \epsilon_k^T$.

When experimenting with direct controllers that generalize the operator's action (rule 4) with some learning error ϵ_k^D, we noticed that they are very brittle w.r.t. the learning error and result in much higher controller errors than these in Table 1. The reason is that the rule 4 is purely reactive. In the rest of the

Table 1. The robustness of direct and indirect controllers against the learning error: c^D and c^I denote the controller error for the direct and the indirect controllers with the learning errors modeled as Gaussian noise with std.dev. σ. c^D* and c^I* denote the controller's error where the predictors' error was modeled by *biased* Gaussian noise. The results are averages of 20 runs

learning	direct controller		indir. controller	
error σ	c^D	c^D*	c^I	c^I*
0.0	1.005	1.005	1.005	1.005
10^{-3}	1.08	1.22	1.08	1.17
10^{-2}	1.6	3.3	1.6	3.1
0.1	9.5	58.2	8.7	18.3
0.2	18.9	> 100	20.0	39.8
0.5	86.5	> 100	83.5	> 100

experiments we used better control rule for u_k^{op}. It uses the system dynamics (eq. 3) to achieve the desired $\dot{x}_{k+1}^{op}$ from rule 5. In this way direct controllers were given the advantage of exact system model, while indirect controllers used a very simple method to approximate the system dynamics.

To compare the robustness of direct and indirect controllers with respect to the learning error we performed a set of experiments, where we modeled different error distributions of the learning system and measured the performance of both controllers. We experimented with two prediction error models. First the error was randomly generated as zero mean Gaussian noise with various standard deviations σ. The second model is the same as Gaussian noise, but with all the errors (in the same control trace) in the same directions, that is errors in the same control trace are either positive or negative, with random absolute value. We call this *biased* Gaussian noise. Biased Gaussian noise is obtained from Gaussian noise just by adjusting the sign (making the sign uniform for the whole control trace). Note that biased Gaussian noise thus has the same σ, but of course no longer has zero mean.

The rationale for defining our experiment in this way is as follows. We wanted to make our experiment independent of the particular learning technique, so we modeled the "induced" predictors simply by taking the correct value and corrupting it with noise. Gaussian noise as error model seem to be an obvious choice. The second model, biased Gaussian noise is, however, not quite so obvious. We designed this error model to account for a frequent property of induced continuity predictors: namely, that small changes in the attribute values usually do not cause large changes in the predicted class values. Therefore the errors at neighbour points have the same sign. The results in Table 1 show that Gaussian noise in the respective predictors affect both direct and indirect controllers in terms of their control cost to a similar extent. Biased Gaussian noise is more damaging to both controllers than zero mean Gaussian noise with the same σ. However, the direct controller is affected much more than the indirect controller. This happens, surprisingly, in spite of the fact that in addition to the error in

generalized trajectory, the indirect controller also has to cope with the error in the induced dynamics model. This result provides a plausible explanation why indirect controllers have proved to be much more successful in the experiments with learning techniques in the crane [17,16] and acrobot [15] domains.

We believe that this superiority of the indirect controller follows from their goal-directness (generalized operator's trajectory can be seen as the continuous subgoal) and their use of the system's dynamics model, which is learned online and thus enable them to behave well also in the yet unseen regions of the state space.

5 Why Learning Indirect Controllers Is Better than Learning Direct Controllers?

Here we would like to comment on the plausible advantages of learning indirect controllers. We believe that most of these advantages are consequences of the decomposition of learning an indirect controller into two natural subproblems: learning what to do (the trajectory) and learning how to do it (the system's dynamics). The only difficult task is to learn the generalized trajectory. As confirm the experimental results, the system's dynamics can be approximated sufficiently well by instance based methods of learning from execution traces and can be also updated online during the control. Advantages of learning indirect controllers are:

1. An indirect controller is less prone to the departure from the desired trajectory. Due to the operator's unconsistency (mistakes, taking different actions in similar situations) and due to the learning errors the learned trajectory or the action is not perfect. When taking an incorrect action the system often departs from the desired trajectory and arrives to the yet unseen region of the state space. Indirect controller is goal directed and considers the system's dynamics to take an action back towards the trajectory. A direct controller has no clue how to get back to the known region of the state space and often just gets stuck in the unknown region.
2. An indirect controller is more robust against change in the system's dynamics and small changes in the task. The robustness against change in the system's dynamics is ensured by updating the system's dynamics model during the control. The robustness against small changes in the task (like changing the system start state) follows from the better robustness of the indirect controller against the departure from the desired trajectory. This of course assumes that the learned generalized trajectory is still appropriate for the changed system's dynamics or the changed task.
3. Generalizing the trajectory is often easier than generalizing the actions. In many systems different actions can result in similar effect on the system behavior. When selecting the action to take, an experienced operator keeps in mind only the desired effect. In similar situations he often uses different actions to achieve the same effect. These different actions in similar states are

noise for the learning system if it learns the mapping from the system's states to the actions, but are not noise when learning the mapping from the system's states to the desired effects, that is generalized trajectory. For example, when controlling a crane, the operator can use quite different actions to slow down the trolley: the sequence of oscillating actions around $Force_X$=2000 can result in similar behavior as a sequence of actions $Force_X$=2000.

4. Learning an indirect controller seems usually to be an easier task than learning a direct controller. When learning an indirect controller, the only difficult task is to learn the generalized trajectory, that is what the operator is doing. In the case of a direct controller the task is to learn what the operator is doing and how he does it. This seems to be a more difficult learning task.

In addition indirect controllers have other advantages:

1. The learned trajectory is easier to understand than the actions, which usually involve knowledge of the system's dynamics. The desired trajectory usually captures the knowledge of performing the given task in a more compact way. When driving the car on the parking space from one point to the other, it is easier to understand the (x,y) trajectory, than the sequence of actions like pushing the gas pedal, turning the wheel for 30 degrees and straightening it after, pushing the breaks, turning the wheel again,...
2. Knowledge learned by generalized symbolic trajectory can easily be adopted to new tasks. For example, in the crane domain, the trajectory for attaining the goal position at X=60 can easily be adopted to goal position at X=80.

6 Related Work

Another approach that deals with learning to control dynamic systems is reinforcement learning [9]. Unlike behavioural cloning, reinforcement learning learns control from scratch, through trial-and-error. The idea is to use dynamic programming to learn the value function or Q-function estimating how promising (in achieving the goal) each state or action is. In the most common approach they learn direct controllers. To reduce experimentation with the dynamic system, model-based reinforcement learning methods [12] also use learned model of system dynamics and therefore effectively learn indirect controllers. However, reinforcement learning does not use human operator's control skill and is not concerned with the comprehensibility of the learned strategy.

A similar idea of decomposing the cloning problem into two learning problems appears in [4]. They use abduction to learn effects of control actions and decision trees to learn subgoals. However, they do not report on benefits regarding the success of controllers constructed in this way.

Acknowledgments

The results in this paper are part of the first author's work on PhD thesis at University of Ljubljana, Faculty of Computer and Information Science. Both authors were supported by Slovenian Ministry of Science and Technology.

References

1. Aha, D., Kibler, D., Albert, M.: Instance-based learning algorithms. *Machine Learning* 6:37–66 (1991). 387
2. Atkeson, C., Moore, A., Schaal, S.: Locally weighted learning for control. *Artificial Intelligence Review* 11:75–113 (1997). 387
3. Barto, A., Sutton, R., Anderson, C.: Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics* SMC-13(5):834–846 (1983). 385
4. Bain, B., Sammut, C.: A framework for behavioural cloning. *Machine Intelligence 15* Oxford University Press. Forthcoming. 390
5. Bertsekas, D.: *Dynamic Programming: Deterministic and Stochastic Models*. Englewood Cliffs: Prentice Hall (1987). 385
6. Bratko, I., Urbančič, T., Sammut, C.: Behavioural cloning of control skill. In Michalski, R.; Bratko, I.; and Kubat, M., eds., *Machine Learning and Data Mining: Methods and Applications*, 335–351. John Wiley & Sons, Ltd (1998). 382, 385
7. Chambers, R., Michie, D.: Man-machine co-operation in a learning task. *Computer Graphics: Techniques and Applications* 179–186 (1969). 382
8. Donaldson, P.: Error decorrelation studies on a human operator performing a balancing task. *Med. Electron. Biol. Engng.* 2:393–410 (1964). 382
9. Kaelbling, L., Littman, M., Moore, A.: Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4:237–295 (1996). 390
10. Michie, D., Chambers, R.: Boxes: an experiment in adaptive control. In Dale, E., and Michie, D., eds., *Machine Intelligence 2*, 137–152. Edinburgh University Press (1968). 385
11. Sammut, C., Hurst, S., Kedzier, D., Michie, D.: Learning to fly. In *Proceedings of the 9th International Workshop on Machine Learning*, 385–393. Morgan Kaufmann (1992). 382
12. Sutton, R.: Integrated arhitectures for learning, plannining and reacting based on approximating dynamic programming. In *Proceedings of the 7th International Conference on Machine Learning*, 216–224. Morgan Kaufmann (1990). 390
13. Urbančič, T., Bratko, I.: Reconstructing human skill with machine learning. In Cohn, A., ed., *Proceedings of the 11th European Conference on Artificial Intelligence*, 498–502. John Wiley & Sons, Ltd (1994). 382
14. Šuc, D., Bratko, I.: Skill reconstruction as induction of LQ controllers with subgoals. In Mellish, C. S., ed., *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, volume 2, 914–920 (1997). 383
15. Šuc, D., Bratko, I.: Skill modelling through symbolic and qualitative reconstruction of operator's trajectories. Technical report, Artificial Intelligence Lab., Faculty of Computer and Information Sc., University of Ljubljana, Slovenia (1998). 383, 389
16. Šuc, D., Bratko, I.: Modelling of control skill by qualitative constraints. In *Proceedings of the 13th International Workshop on Qualitative Reasoning*, 212–220. University of Aberystwyth. Loch Awe, Scotland (1999). 385, 389
17. Šuc, D., Bratko, I.: Symbolic and qualitative reconstruction of control skill. *Electronic Transactions on Artificial Intelligence* 3. Forthcoming. 383, 385, 389