

Logic Programming with Simulation-based Temporal Projection for Everyday Robot Object Manipulation

Lars Kunze, Mihai Emanuel Dolha, and Michael Beetz
Intelligent Autonomous Systems, Technische Universität München
{kunzel, dolha, beetz}@cs.tum.edu

Abstract—In everyday object manipulation tasks, like making a pancake, autonomous robots are required to decide on the appropriate action parametrizations in order to achieve desired (and to avoid undesired) outcomes. For determining the right parameters for actions like pouring a pancake mix onto a pancake maker, robots need capabilities to predict the physical consequences of their own manipulation actions. In this work, we integrate a simulation-based approach for making temporal projections for robot manipulation actions into the logic programming language PROLOG. The realized system enables robots to determine action parameters that bring about certain effects by utilizing simulation-based temporal projections within PROLOG’s chronological backtracking mechanism. For a set of formal parameters and their respective ranges of values, the developed system translates the manipulation problems into physical simulations, monitors and logs the relevant data structures of the simulations, translates the logged data back into first-order time-interval-based representations, called timelines, and eventually evaluates the individual timelines with respect to specified performance criteria. Integrating the proposed approach into robot control programs allow robots to mentally simulate the consequences of different action parametrizations before committing to them and thereby to reduce the number of undesired outcomes.

I. INTRODUCTION

Robots that are to successfully perform everyday manipulation tasks must decide on their actions and their appropriate parametrizations based on the expected consequences of the respective parametrized actions. A number of researchers consider this form of prediction-based decision making as an essential component of cognition-enabled robot control.

Consider for example a robot making an egg sunny side up. In order to do so, the robot has to break an egg and let the content gently drop into the frying pan. A key problem in performing such everyday manipulation actions is that the actions will typically insufficiently specified. When humans instruct each others to break an egg they will typically not specify the force with which the egg is to be grasped, how hard it is to be hit, where, to which part of which other object, and so on. Thus, to perform such everyday manipulation actions successfully, the robot has to appropriately fill in the knowledge gaps by inferring decisions and parametrizations.

In order to make informed decisions and infer appropriate parametrizations, the robot has to predict the consequences of different action parametrizations — at least qualitatively. In a sense the robots have to perform lightweight naive physics reasoning to determine the expected action consequences.

As this kind of naive physics and commonsense reasoning is done in order to improve the robot’s manipulation capa-

bilities, they must be linked much more tightly to the robot’s perception and control routines and have to make inferences for specific (continuous) action parametrizations.

In this paper, we propose to equip robots with this kind of reasoning capabilities by coupling PROLOG, a logic programming language, with a physical and sensor-based simulation engine. The basic idea is that the robot control program asserts the available information as assertions to the PROLOG system and queries the PROLOG interpreter about the possible consequences of executing an underspecified action in the asserted state. The PROLOG system selects instantiations of the unbound variables that are necessary to predict the consequences of the actions and then backtracks over possible instantiations. For each, fully instantiated prediction problem a simulation task for the simulator is created and executed. The data structures generated and updated by the simulation process are logged and virtually transformed into a timeline, a first-order temporal logic based representation of what is simulated. This timeline is then used to answer queries about the consequences of the action.

In the remainder of this paper we proceed as follows. First, we recap a simulation-based approach for making temporal projections for robot manipulation tasks in Section II. Thereafter, we explain how this approach is integrated into the logic programming language PROLOG in Section III. Experimental results for three manipulation actions, namely grasping an egg, pouring a pancake mix onto a pancake maker, and turning a pancake are presented in Section IV. We briefly discuss features and limitations of our approach in Section V. Finally, we put our work into context in Section VI before we conclude in Section VII.

II. SIMULATION-BASED TEMPORAL PROJECTION

This paper builds on work on simulation-based temporal projections described in [1]. In this section, we shortly recapitulate this approach before we explain how it is integrated with the logic programming language PROLOG.

Figure 1 depicts the overall process of simulation-based temporal projection. A logical axiomatization of a manipulation scenario, stored in the knowledge base, is translated into a physics-based simulation, the simulation is executed, objects of interest are monitored, and relevant data structures are logged. Eventually, the logged simulations are translated back into time-interval-based first-order representations, called timelines, which are interpreted by using a logic programming language.

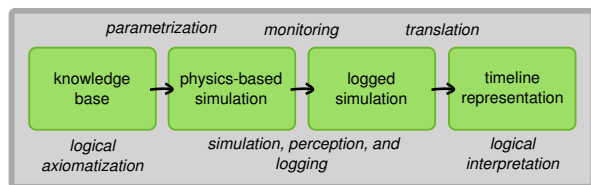


Fig. 1. Simulation-based temporal projection.

The knowledge base, which is formalized using Description Logics, in particular the Web Ontology Language (OWL), holds information about manipulation scenarios, i.e., working environment, robots, and task-related objects. Object descriptions, which include type, shape and property information are used to configure the physical parameters of the rigid-body simulator Gazebo¹. For simulating object behaviors like breaking that are not covered by the rigid-body physics, graph-based object models and controllers that perform specialized computations in each simulation step have been introduced. Monitoring routines are attached to objects of interest in order to log the data structures and states traversed in simulation. The resulting logs are translated into back time-interval-based first-order representations (timelines). Timelines are represented using predicates known from the *event calculus*, namely $Holds(f,t)$ and $Occurs(e,t)$, where f denotes a fluent, e denotes an event and t denotes the time. For reasoning on the timeline representations derived from logged simulations the logic programming language PROLOG is employed. Queries can be formulated by using temporal relations between asserted fluents and events. For example, the following query asks for all fluents F that hold for $egg1$ after it has been dropped onto the floor at time $T1$:

```
?- occurs(Evt,T1), typeOf(Evt,dropping), objOf(Evt,egg1),
   after(T2,T1), holds(F,T2), objOf(F,egg1).
```

Overall, the system allows to make temporal projections by evaluating queries about physical phenomena on timeline representations which are based on detailed physical simulations. However, the developed system cannot *automatically* generate and evaluate timelines for different action parametrizations and slight variations of manipulation problems. In the following section, we first explain the general framework of logic programming with simulation-based temporal projection, and second, we present the language elements that integrate the projection mechanism into PROLOG to automatically backtrack over different parametrizations.

III. LOGIC PROGRAMMING AND TEMPORAL PROJECTION

After giving an overview of the overall architecture of the logic programming framework we describe how the simulation-based temporal projection explained in the previous section is integrated into the framework.

A. Framework Overview

Figure 2 shows the individual components of the simulation-based logic programming framework. The framework enables a robot to automatically reason about the

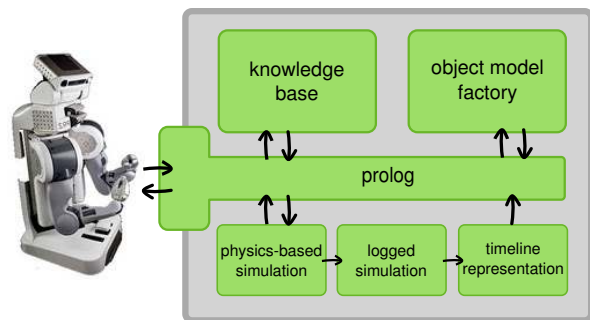


Fig. 2. Simulation-based logic programming framework.

consequences when executing a control program with both different action parametrizations and/or different variants of manipulation tasks, e.g., varying the positions of objects.

The main component of the framework is a PROLOG engine which serves as interface to the robot, and coordinates all other components of the framework. In order to answer a query from the robot, PROLOG retrieves the descriptions of the all task-related objects and the robot's environment from the knowledge base, and, when necessary, calls the object model factory to generate parametrized object models for the physics-based simulator. Then, PROLOG sets up the simulation using the world description from the knowledge base and the generated object models and launches a parametrized robot control program that is executed within the simulator. During the execution object states and associated data structures are monitored and logged. When PROLOG stops the simulation after a given time, the object logs are translated back into first-order representations and eventually they are evaluated in regard to specified performance criteria. If the evaluation is successful, PROLOG presents a solution, otherwise it backtracks over different parametrizations.

B. Temporal Projection Language

The basic idea of a logic programming language for making simulation-based temporal projections is as follows. First, a new scenario is created and task-relevant descriptions are added to it successively, for example, an environment description, a robot description, and a number of object descriptions. After initializing the simulator with the asserted scenario descriptions, a robot control program is executed whereby formal control parameters are selected from a specified range of possible values. States of the robot and objects that are traversed during simulation are monitored, logged, and translated into time-interval-based first-order representations, namely timelines. Eventually, the generated timelines are subject to further evaluations of specialized predicates. For example, a timeline is evaluated with respect to desired (or undesired) outcomes, qualitative spatial relations, or other performance criteria like the speed of execution.

The following PROLOG query shows exemplarily how the simulation-based temporal projection can be used where terms starting with an upper-case letter like *Scenario* denote variables, terms starting with a lower-case like *kitchen_env* denote concrete instances in the knowledge base, and the

¹<http://playerstage.sourceforge.net/gazebo/gazebo.html>

predicate *occurs* stands exemplarily for a specialized predicate that evaluates a given timeline:

```
?- create_scenario(Scenario),
   add_to_scenario(Scenario,kitchen_env),
   add_to_scenario(Scenario,pr2_robot),
   add_to_scenario(Scenario,obj1).
?- make_projection($Scenario,
   program,
   [(P1, param1, range1),
    (P2, param2, range2)],
   Timeline),
   occurs(event, Time, Timeline).
```

Values for the formal parameters, e.g. *param1*, are selected from their respective ranges and are bound to the parameter variables (e.g. *P1*) in order to make them accessible for further evaluations. How to generate and/or select the values of the parameters more effectively is another interesting problem. Intuitively, the parameters could be chosen depending on the qualitative outcomes of the simulation, however, this is beyond the scope of this paper.

The language elements for making temporal projections, i.e., the PROLOG programs (or predicates) that have been implemented in order to assert a scenario, to perform a simulation-based temporal projection, and to logically evaluate the resulting timelines, are explained in the following.

create_scenario(Scenario) Creates a new scenario and generates a unique identifier (*Scenario*) to access the scenario within other predicates.

add_to_scenario(Scenario, Entity) Adds an entity or set of entities to a given scenario. There are three ways of adding an entity: either by naming the entity, if it is already known by the knowledge base including its physical specifications that are needed by the simulator, by providing the physical specifications of a previously unknown entity explicitly, or by providing an object type that can be generated by the object model factory.

make_projection(Scn, Prog, Params, Timeout, Timeline) Performs a simulation-based temporal projection for an asserted scenario, a robot control program, a set of formal control parameters including their possible values, a timeout that limits the overall simulation time, and returns an ID of the projected timeline. This program is realized by two sub-programs, namely *simulate* and *translate*.

simulate(Scn, Prog, Params, Timeout, Log) Sets up and runs the simulation. First, the Gazebo simulator is launched and all entities that were added to the given scenario by the *add_to_scenario* command are loaded successively. If necessary, entity specifications are generated on-the-fly by the object model factory and spawned into the simulator. Second, the robot control program is executed, where formal parameters are selected from their respective ranges. By utilizing PROLOG’s backtracking mechanism the cross product of all valid parameter instantiations is automatically generated. Third, after a given time (*Timeout*), the simulation is stopped and all processes are shut down. The output variable *Log* points to the log files of the robot control program and the task-related objects.

translate(Log, Timeline) Translates the logged simulations into a time-interval-base representation, i.e. a timeline, by using the first-order predicates *Holds(f,t)* and *Occurs(e,t)*. To differentiate between the individual timelines a unique ID (*Timeline*) is generated and attached to the individual fluents and events.

occurs(Event, Time, Timeline) Retrieves the given *Timeline* and evaluates it with respect to an event (*Event*) that might have occurred at a point in time (*Time*) during the simulation. If the specified event is found in the timeline the predicate evaluates to true.

holds(Fluent, Time, Timeline) Retrieves the given *Timeline* and evaluates it with respect to a fluent (*Fluent*) that might have hold at a point in time (*Time*) during the simulation. If the specified fluent is found in the timeline the predicate evaluates to true.

IV. EXPERIMENTS

In this section we show the feasibility of the implemented framework on three manipulation problems: grasping an egg, pouring a pancake mix, and turning a pancake. Multiple videos of the experiments are available online².

A. Grasping an egg

A robot would not know that an egg will break if it is grasped with a force that is too high, that it will fall down if it is grasped with a force that is too low, and that it will slip away if it is grasped at an inappropriate position.

We setup a grasping experiment where the parameters force and position were varied. A scenario (*grasping*) was created and entities like kitchen environment, PR2 robot, and an egg were added. The following code was used for running the experiments and evaluating their respective results:

```
?- make_projection(grasping,
   grasp_object,
   [(P1, force, [1, 3, 5, 7, 9, 11]),
    (P2, position, [-0.02, 0.00, 0.02])],
   Timeline),
   not(occurs(break,_,Timeline)),
   not(occurs(slip,_,Timeline)).
```

The qualitative results of this experiment are shown in Table I, where the grasping position is measured as the center of the gripper pad relative to the center of the egg (in meters) and the different grasping forces are indicated by the identifiers *f1–f11*. The different outcomes are denoted as *ok* if the robot could grasp the egg successfully, as *slip* if the egg slipped away, and as *break* if the the egg was broken within the trial. Figure 3 shows two examples where the egg could not be grasped by the robot.

TABLE I
QUALITATIVE RESULTS: GRASPING AN EGG.

position	f1	f3	f5	f7	f9	f11
-0.02	slip	ok	slip	ok	break	break
0.00	slip	ok	ok	ok	ok	break
0.02	slip	ok	ok	ok	break	break

²<http://ias.cs.tum.edu/download/naive-physics-simulation>

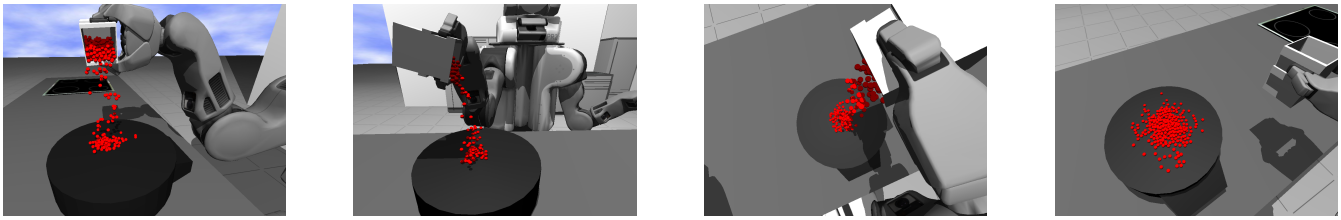


Fig. 5. The simulated pancake mix, made out of small spherical particles is poured onto the pancake maker.

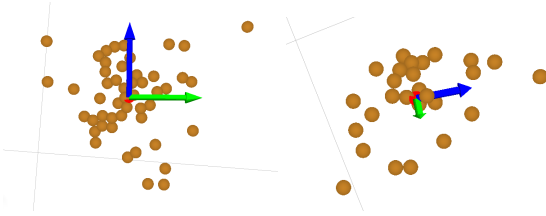


Fig. 6. PCA of particle configurations on the pancake maker. *Roundness* of the area left: 0.96 and right: 0.36.

deformable body (Figure 7). During this scenario, the PR2 robot uses a spatula to flip a pancake on the pancake maker. The parameter of interest in this case is the angle of the spatula with the horizontal and the outcome can either be failed if a pancake turn is not achieved or successful if it is. The qualitative results of the experiments performed in this scenario are shown in Table III. The experiments were launched using the following query:

```
?- make_projection(turning,
                  turn_pancake,
                  [(P1, angle, [0.1, 0.3, 0.4, 0.5, 0.7, 0.9])],
                  Timeline),
   holds(upsidedown, _, Timeline).
```

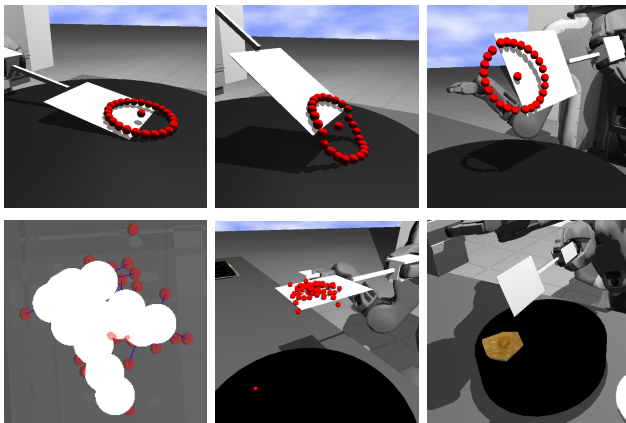


Fig. 7. The pancake model in different views: a generic circular model (up), a generated model showing the joint structure and flexible mesh (down).

Pouring and turning experiments can be combined by using the poured particles that end up on the pancake maker for generating a more complex pancake model. We start at the particle closest to the center of the cluster and create a graph like flexible joint structure. Joints are created between the seed particle and the particles found within a certain radius from it, and afterward these new particles become seeds themselves. To make the pancake model look more realistic, a flexible textured mesh created from the convex hull is attached to the structure (Figure 7).

V. DISCUSSION

In this section we discuss why autonomous robots should be endowed with methods allowing to make temporal projections about naive physics problems. We provide arguments to base these methods on detailed physical simulations and elaborate on the right fidelity of these simulations. Furthermore, we outline how this approach of logic programming using a simulation-based temporal projection can be used to adjust the behavior of robots. Finally, we examine how far the proposed approach can be taken, and also name some possible application scenarios.

One might argue that most robotic applications are developed for specialized tasks and thereby robots do not need robust commonsense reasoning capabilities, or as we propose, capabilities for naive physics reasoning. But in the context of autonomous personal robots the set of everyday manipulation tasks is not fixed, and furthermore, task and environment conditions change all the time and therefore robots need flexible mechanisms to reason about the appropriate parameters of their control programs.

In the literature there exist some approaches using symbolic reasoning methods for making inferences about simple physical problems ([2]). The main limitations of these approaches especially in the context of robotics are threefold: (a) important details like positions of manipulators and objects are abstracted away; (b) variants of problems like manipulating an object with different physical properties cannot be handled without extending the logical theory; and (c) consequences of concurrent actions and events are very difficult to foresee with pure symbolic reasoning, e.g., what does a robot see when turning its camera while navigating through its environment? All of these limitations do not occur in physics-based simulators. Even if the simulation do not reflect the physical world, parameters can be learned by applying machine learning technologies as in [3].

One important issue when using high-fidelity physics models in simulations is performance. Currently, the system cannot make predictions in a reasonable time that would allow us to use it for planning during execution. Nevertheless, it is a powerful tool for robots to *mentally* simulate (offline) the consequences of their own actions either to prepare themselves for new tasks or to reconsider task failures.

Related to the issue of performance is the issue of the *right* fidelity, i.e. how to make the physical models robust enough to enable effective behavior and yet small enough to be usable during execution. When creating physical models we are concerned about getting the qualitative behavior of objects right, i.e., we are not aiming at models that reflect

TABLE II
RESULTS: POURING A PANCAKE MIX.

pos ¹	time ²	angle ³	pan ⁴	spill ⁵	off _x ⁶	off _y ⁷	round ⁸
	0.0	1.84	2	0	-0.004	-0.098	0.00
		1.99	25	2	-0.028	-0.096	0.19
		2.14	106	6	-0.011	-0.095	0.55
-0.09	0.5	1.84	14	1	-0.021	-0.094	0.14
		1.99	95	6	-0.005	-0.087	0.89
		2.14	205	16	0.005	-0.086	0.51
	1.0	1.84	57	1	-0.007	-0.091	0.37
		1.99	163	6	0.000	-0.083	0.58
		2.14	247	28	0.000	-0.085	0.59
	0.0	1.84	1	0	0.008	-0.110	—
		1.99	41	0	-0.025	-0.004	0.51
		2.14	100	1	-0.015	-0.003	0.77
0.00	0.5	1.84	19	0	-0.010	-0.005	0.19
		1.99	81	0	-0.022	-0.004	0.57
		2.14	212	0	-0.009	-0.003	0.89
	1.0	1.84	36	1	-0.016	0.000	0.57
		1.99	132	0	-0.015	0.001	0.57
		2.14	255	2	-0.019	0.000	0.88
	0.0	1.84	0	0	—	—	—
		1.99	44	3	-0.017	0.067	0.59
		2.14	117	1	-0.002	0.062	0.76
0.07	0.5	1.84	24	1	-0.007	0.054	0.39
		1.99	109	2	-0.012	0.074	0.62
		2.14	222	3	-0.011	0.062	0.90
	1.0	1.84	40	0	-0.012	0.071	0.85
		1.99	156	4	-0.002	0.068	0.79
		2.14	287	5	0.001	0.070	0.76

Abbreviations

- ¹ Offset of cup position relative to the center of the pancake maker.
- ² Duration that the cup is maintained at the pouring angle.
- ³ Angle of the cup's inclination in radians (e.g.: 0 = cup upright; 1.57 = cup tilted 90 degrees counterclockwise).
- ⁴ Number of particles found on the pancake maker
- ⁵ Number of particles spilled onto the table.
- ⁶ X-offset in meters of particles' centroid relative to pancake maker.
- ⁷ Y-offset in meters of particles' centroid relative to pancake maker.
- ⁸ Degree of roundness. Ratio of first two eigenvalues of particles on pancake maker (range: 0.0–1.0, values closer to 1 mean more round).

TABLE III
QUALITATIVE RESULTS: TURNING A PANCAKE.

angle	0.1	0.3	0.4	0.5	0.7	0.9
	ok	ok	ok	fail	fail	fail

every sheer detail. Very detailed models do not readily provide the information needed to choose the appropriate action parametrization, therefore we abstract the reality into a smaller qualitative state space.

Although it would be desirable to use the presented approach for planning during execution by using more realistic physical models, we are currently not aiming at both, high performance and very realistic models. Rather the developed system represents a proof-of-concept how to use simulation technologies for symbolic reasoning. In the long run, we assume that issues regarding performance and the appropriate fidelity of physical models will be addressed by the game and character animation industry (e.g. [4]), which will provide

powerful technologies that could be employed.

The realized logic programming framework allow robots and programmers to automatically determine the appropriate action parameters by setting up a manipulation scenario, by executing differently parametrized control programs in simulation, and finally, by evaluating queries based on the resulting timelines. An interface to the logic programming framework is provided by both PROLOG's command-line and a ROS⁴ service, which takes arbitrary PROLOG queries as request and provides the respective variable bindings as response. Thereby, naive physics reasoning for manipulation tasks can be flexibly integrated into control programs and planners in order to effectively change the robot's behavior.

Finally, we want to approach the question of how far this approach can or should be taken, before we point to some potential application scenarios. It is clear that one would not want to do this kind of full-fidelity physics simulation for all kinds of problems, e.g. problems in motion planning can be solved by employing more specific planners as primitives. However, some kind of limited simulation seems to be very plausible, at least for some very hard problems. We believe that lifting physics-based simulations to a symbolic level is beneficial for deriving solutions for robot manipulation, and also other domains, where current methods are not effective.

Planning is increasingly considering physical platforms in complex, real world environments. The presented framework could provide a more precise guidance in the planning process since the simulation-based methods for making temporal projections are tightly linked to the technical details of platforms under question. Naive physics reasoning could also be used as a tool for developing robot control programs. Programmers could recognize and prevent problems occurring during execution more easily. Additionally, the presented framework could be used for benchmarking purposes. For example, data generated by the simulations could serve as basis for inference tasks. Thereby, different approaches to physical reasoning could be compared in a straight forward way. In general, the usage of the open source software like Gazebo and ROS allow to employ the naive physics reasoning to new problems including other robot platforms and different objects quite easily. Therefore we believe that logic programming using detailed physical simulations is a well-suited tool for making predictions about every manipulation tasks and also for other potential applications.

VI. RELATED WORK

The present work can be considered as interdisciplinary research of two fields: robotics and AI. With this research, we want to enable robots to reason about the consequences of action parametrizations and thereby allowing them to make appropriate decisions in their course of action by using well-established methods of AI and detailed physical simulations.

Only recently, [5] stressed the importance of using simulations in AI research. They developed the open source simulator *IsisWorld* for investigating problems in commonsense

⁴<http://www.ros.org>

reasoning. Although they also employ a physics engine for their simulations, they consider actions like *picking up an object* only at a very abstract level, whereas we focus on the physical details of such actions in order to recognize qualitative phenomena occurring during their execution.

In [6], a general simulation framework and logic-based reasoning methods, in particular tableau-based reasoning, are integrated in order to establish a practical approach to commonsense reasoning. In contrast to their work, we are not aiming at commonsense reasoning in general but rather at reasoning for naive physics problems in the context of everyday robot object manipulation.

Work by [7] describes the design and implementation of a programming system based on EusLisp that make use of a simulation for deformable objects. Thereby, robot control programs can easily exploit the specialized computations made by the simulation. Similarly, we use the logic programming environment PROLOG and utilize a physics-based robot simulator. In addition, we integrated methods for making simulation-based temporal projections into PROLOG's backtracking mechanism in order to perform reasoning about action parametrizations for robot manipulation tasks.

The interactive cooking simulator [8] is relevant for our work, since with the research aims at a deep understanding of cooking operations, which could bring new insights with respect to representations and reasoning mechanisms for manipulation actions in everyday meal-preparation tasks.

Exploiting physical simulators for effectively solving sub-problems in the context of robotics has become more attractive as shown by a number of recent investigations, where simulations are employed for planning in robocup soccer [9], for navigating in environments with deformable objects [10], and for reasoning about the consequences of everyday manipulation tasks [1]. A detailed evaluation for using physics engines for improving the physical reasoning capabilities of robots is given in [11]. But other fields also recognize simulators as valuable tools and utilize them, e.g., for character animation [12] and motion tracking [13].

VII. CONCLUSIONS

In this paper we presented a logic programming framework for reasoning about naive physics problems in the context of everyday robot manipulation tasks.

We developed a logical programming language for asserting the initial conditions of a scenario and for utilizing a simulation-based approach for making temporal projections about parametrized robot control programs. We conducted experiments for three scenarios, namely grasping an egg, pouring a pancake mix, and turning a pancake, in which formal parameters of robot control programs were systematically selected from ranges of possible values. These experiments, or more precisely their resulting timelines, were evaluated with respect to specified performance criteria, e.g. desired and undesired effects. In the discussion section (Section V), we explained the demand of equipping robots with means of naive physics reasoning and provided arguments

for basing this reasoning on detailed physical simulations. Furthermore, we also pointed to potential applications.

In future work we will extend the pancake making scenario so that it reflects more aspects of the real world manipulation problem. For example, we will integrate some kind of heat simulation that changes the state of liquid particles in order to let a pancake simply emerge from the previously poured pancake mix, and also we will improve the liquid model by approximating the liquid's surface tension. Additionally, we will integrate the reasoning framework on a real robot platform in order to compare and evaluate the mental simulations with the physical phenomena happening in the real world.

We believe that the presented framework provides an important functionality for robots by giving them the ability to autonomously determine the action parametrizations for unspecified and ambiguous instructions by the means of logic programs using simulation-based temporal projections.

Acknowledgments: We would like to thank our colleague Andrei Haidu for his help. This work is supported in part within the DFG excellence initiative research cluster *Cognition for Technical Systems* (<http://www.cotesys.org>).

REFERENCES

- [1] L. Kunze, M. E. Dolha, E. Guzman, and M. Beetz, "Simulation-based temporal projection of everyday robot object manipulation," in *Proc. of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, 2011.
- [2] L. Morgenstern, "Mid-sized axiomatizations of commonsense problems: A case study in egg cracking," *Studia Logica*, 2001.
- [3] B. Johnston and M. Williams, "Autonomous learning of commonsense simulations," in *Int. Symposium on Logical Formalizations of Commonsense Reasoning*, 2009.
- [4] J. H. Cho, A. Xenakis, S. Grosz, and A. Shah, "Anyone can cook – inside ratatouille's kitchen," in *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, 2007.
- [5] D. Smith and B. Morgan, "Isisworld: An open source commonsense simulator for ai researchers," in *AAAI Workshops*, 2010.
- [6] B. Johnston and M. Williams, "Comirit: Commonsense Reasoning by Integrating Simulation and Logic," in *Artificial General Intelligence 2008: Proc. of the First AGI Conf.*, 2008.
- [7] R. Ueda, T. Ogura, K. Okada, and M. Inaba, "Design and implementation of humanoid programming system powered by deformable objects simulation," in *Proc. of the 10th Int. Conf. on Intelligent Autonomous Systems*, 2008.
- [8] F. Kato, Y. Hanaoka, T. N. Ngoc, D. Keoki, H. Mitake, T. Aoki, and S. Hasegawa, "Interactive cooking simulator: to understand cooking operation deeply," in *ACM SIGGRAPH 2009 Emerging Technologies*, 2009.
- [9] S. Zickler and M. Veloso, "Efficient physics-based planning: sampling search via non-deterministic tactics and skills," in *AAMAS '09: Proc. of The 8th Int. Conf. on Autonomous Agents and Multiagent Systems*, 2009.
- [10] B. Frank, C. Stachniss, R. Schmedding, M. Teschner, and W. Burgard, "Real-world robot navigation amongst deformable obstacles," in *ICRA'09: Proc. of the 2009 IEEE int. conf. on Robotics and Automation*, 2009.
- [11] E. Weitnauer, R. Haschke, and H. Ritter, "Evaluating a physics engine as an ingredient for physical reasoning," in *Proc. of the Second int. conf. on Simulation, modeling, and programming for autonomous robots*, 2010.
- [12] P. Faloutsos, M. van de Panne, and D. Terzopoulos, "Composable controllers for physics-based character animation," in *Proc. of the 28th annual conf. on Computer graphics and interactive techniques*, 2001.
- [13] M. Vondrak, L. Sigal, and O. C. Jenkins, "Physical simulation for probabilistic motion tracking," in *CVPR*, 2008.