A Relational Approach to Tool-Use Learning in Robots

Solly Brown and Claude Sammut

School of Computer Science and Engineering
The University of New South Wales
Sydney, Australia 2052
claude@cse.unsw.edu.au

Abstract. We present a robot agent that learns to exploit objects in its environment as tools, allowing it to solve problems that would otherwise be impossible to achieve. The agent learns by watching a single demonstration of tool use by a teacher and then experiments in the world with a variety of available tools. A variation of explanation-based learning (EBL) first identifies the most important sub-goals the teacher achieved using the tool. The action model constructed from this explanation is then refined by trial-and-error learning with a novel Inductive Logic Programming (ILP) algorithm that generates informative experiments while containing the search space to a practical number of experiments. Relational learning generalises across objects and tasks to learn the spatial and structural constraints that describe useful tools and how they should be employed. The system is evaluated in a simulated robot environment.

Keywords: robot learning, planning, constrain solving, explanation-based learning, version space.

1 Introduction

A tool is any object that is deliberately employed by an agent to help it achieve a goal that would otherwise be more difficult or impossible to accomplish. We would like a robot to be able to learn to use a new tool in a manner similar to the way a human would learn. That is, after the learner observes the tool being used correctly by another agent, the learner forms an initial hypothesis about the properties and use of the tool. The hypothesis is refined by active experimentation, selecting different types of objects as the tool and manipulating them in different ways. By analysing the teacher's demonstration and then experimenting, the robot learns a new tool action that will allow the agent to perform variations of the task. We limit the robot to starting from a single observation since we are most interested in investigating active learning in an incremental manner.

We define a *tool action* to be one that changes the properties of one or more objects in a way that enables the preconditions of one or more other actions in the agent's plan library. The changed properties of the objects that result from the tool action are called the sub-goals of the tool action.

There are four things that the robot must learn so that it consistently solves a tool-use task: (1) what useful effect the tool action achieves, (2) how to identify a good tool (i.e. what object properties are necessary), (3) the correct position in which the tool should be placed, (4) how the tool should be manipulated after being placed. In this paper, we focus on learning the first three of these. For manipulation, the learnt action is translated into an executable behaviour by a spatial constraint solver and a motion planner.

The general setting for learning is as follows:

1. Observe trainer using a tool to achieve a goal
2. Form an initial hypothesis explaining the tool use
3. Construct an experiment to test the hypothesis
4. Update the hypothesis according to the success of failure of the experiment
5. repeat until no further successful hypotheses can be constructed

One of the challenges in active learning is how to choose an experiment that will yield useful information. In a batch learning setting, where many training examples are available, we typically aim to find the simplest concept description that has high accuracy. A general-to-specific search of the hypothesis space is often employed to achieve this. In active learning by a robot, concept formation is done incrementally, with few training example available because they are usually costly to obtain since the robot must expend time and resources to perform an experiment. As we are learning the description of an action, an experiment is an instance of the hypothesis. Were we to employ a purely general-to-specific search, we might generate an instance that is close to the boundary of the hypothesis. The problem with this approach is that we are likely to generate many more negative examples than positive. More importantly, when an experiment fails, we want to know why it failed. The usual scientific method perform experiments that change just one variable at a time so that we know what was the cause of the experiment's outcome. This is more easily done if we employ a specific-to-general search in which we perform a least general generalisation [1]. However, a specific-to-general search to tends yield a complex concept description. Thus, we have developed a bi-directional search, related to Mitchell's [2] version space, where the concept description is derived from the most general hypothesis consistent with the data, but experiment generation is based by the most specific hypothesis.

In the following section, we give an example of tool use, which we use to illustrate the learning method described in the following sections. We then describe our experimental method in section 6, followed by a discussion of related work and our conclusions. The methods described here are applicable to real robots but the experiments were conducted in simulation to avoid the time and cost of dealing with robot hardware. The simulation models a real robot and is sufficiently accurate that the results should translate to the real world.

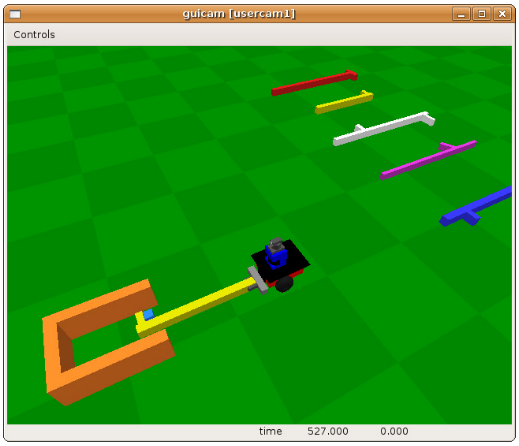

Fig. 1. Using a tool to reach an object in a closed "tube"

2 An Example of Learning to Use a Tool

A robot is set the goal of obtaining an object, in this case a small box, that is placed in a tube lying on the ground. The tube is open at one end and closed at the other, as shown in Figure 1. The robot is unable to reach directly into the tube to pick up the box because the tube is too narrow. The robot must use a hooked stick tool to pull the box out of the tube before it can be picked up.

As shown in Figure 1, the robot is provided with a selection of different objects that can potentially be used as a tool for accomplishing the task. Some of these objects are clearly inappropriate since they cannot be inserted into the tube, lack a suitable "hook" affordance or are not long enough. However, the robot does not know ahead, in advance, which objects make good tools. The only information the robot has is provided by its sensors, i.e. the object's dimensions and the relative orientation of its surfaces.

The robot is also provided with a set of behaviours that it can use to effect changes in the world. In this example, the robot is given **goto**, **pickup-object** and **drop-object** behaviours. We assume that the agent does not already possess any sort of **pull-with-tool** behaviour. Learning this behaviour and using it to obtain the box is the objective of the problem.

The behaviours are represented by action models that describe how executing each behaviour affects the world. The action models are written as STRIPS-like operators [3]. These are used by the robot's planner to create sequences of behaviours that achieve the robot's goals. A robot already competent in solving the tube problem might construct the following plan:

goto(stick-tool), pickup(stick-tool), goto(tube),
pull-with-tool(stick-tool, box),
drop(stick-tool), pickup(box).

The difficulty for our robot is that not only does it lack the appropriate tool-using behaviour, **pull-with-tool**, but it also lacks the action model for this behaviour. This means that the agent is initially unable to form a plan of how to achieve the goal of obtaining the box. It must learn this missing behaviour. We simplify learning by providing the agent with an observation trace of a "teacher" performing the same task. In the case of the tube, our teacher demonstrates by simply picking up an appropriate stick-hook and using it to pull the box from the tube. The robot uses this example to create an initial hypothesis for the tool action, which is the starting point for the robot's experimentation.

In the tube problem the necessary properties of the tool include having a right-angled hook at the end of the handle and that the hook is on the correct side of the handle (if the box is sitting in the left side of the tube, the agent needs a left-sided hook stick). Learning the correct spatial and structural relations for tool use requires experimenting with a variety of tools and poses. We will describe this process after giving details about the action representation.

3 Action Representation

As in the STRIPS representation, the robot's action model includes the preconditions and effects of the action. A **pickup** action is specified as:

pickup(Obj)

PRE	forall(Tube:tube, ¬in(Obj,Tube)),
	empty-gripper,
	forall(x:obj, ¬obstructing(x,Obj))
ADD	gripping(Obj)
DEL	empty-gripper,
PRIMTV	fwd, back, rotleft, rotright
MOVING	robot

The robot must approach an object so that the gripper surrounds the object and can close on it. The precondition states that the object must not be in any tube, the robot's gripper must be empty and there must not be any other object obstructing the target object. The additional lists of primitive motor actions and objects moved by this action are needed to turn the action into motor commands to the robot.

Action models do not specify behaviours in sufficient detail that they can be immediately executed by the robot. In the example above, the action model says nothing about the precise position that the robot must be in after the action is executed, nor does it specify the path that the robot must take to get itself into the goal position. However, actions can be made operational by using a constraint solver to create a range of solutions that satisfy the goal (e.g. any position such that the gripper surrounds the target object) then a motion planner can output a set of primitive motor actions to the robot (in this example, manoeuvring the robot to a position within the bounds set by the constraint solver). This process is illustrated in Figure 2.

We use Hoffmann and Nebel's FF [4] to generate plans and the constraint solving facilities in *ECLiPSe* [5] to generate the constraints. A Rapid Random Tree (RRT) path planner [6] produces the primitive actions for the robot. RRT requires a set of motion primitives, a list of the objects that are manipulated by the planner, and a specification of their relative spatial pose during the manipulation. This is the reason why we extend the STRIPS representation to include a list of the objects involved in the action and a set of robot motion primitives. The motion planner outputs a path for the objects that leads to the desired goal state. A simple controller outputs the motor commands required to follow the path. Brown [7] gives a detailed discussion of the implementation.

In the architecture depicted in Figure 2, nondeterminism in the world is handled by the control layer. The controller implements a feedback loop that monitors the progress of the robot and adjusts the motor actions to minimise errors. This hides much of the nondeterminism of the world from the higher, deliberative layers. Thus, at the abstract level of action models, we can use a deterministic planner and learn deterministic action models. We do not address the problem of nondeterminism at the planning level. For the class of problems described here, this is unnecessary. Future research may investigate learning for nondeterministic planning, possibly incorporating probabilistic ILP.

4 Learning by Explanation

The aim of learning by explanation is to identify novel tool actions in the teacher's demonstration and to construct an action model that describes it. This includes identifying the sub-goal that the tool achieves and some of the necessary preconditions for using the tool. It involves the following steps: (1) watch a teacher using a tool to complete a task; (2) identify actions in the teacher's demonstration by matching them against actions stored in the robots plan library; (3) any sections of the demonstration that cannot be matched to known actions are labelled as components of a novel action; (4) a STRIPS model for the novel action is constructed by identifying the subset of literals in the action's start and end states that are relevant to explaining how the teacher achieved the goal.

To recognise novel behaviours, the robot begins by trying to explain what the teacher is doing during its demonstration. The robot constructs an explanation by trying to match its own set of action models to the execution trace of the teacher's demonstration. Gaps in the explanation, where the agent is unable to match an existing behaviour to what the teacher is doing, are designated novel behaviours, which the robot can try to learn.

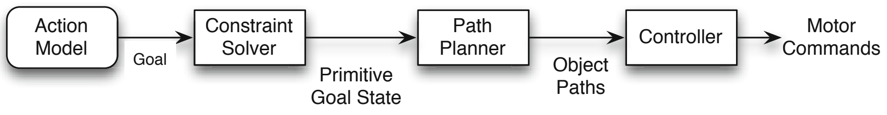

Fig. 2. Behaviour generation

The first difficulty faced by the learner is in labelling the parts of the demonstration that it recognises. The demonstration trace is provided to the agent as a discrete time series $w_1, w_2, \ldots, w_n$ of snapshots of the low-level world state taken every tenth of a second. This world state consists of the positions and orientations of each object in the world at a particular point in time. We do not provide the learner with a demonstration trace that is already segmented. Rather, the trace is a sampling of a continuous time series of object poses and the learner must arrive at a segmentation of the trace by itself.

The trace is segmented into discrete actions by applying an heuristic, that action boundaries occur at points at which objects start or stop moving. Thus, when the agent grips a stick-tool and starts moving it towards the tube, an action boundary is recognised. When the tool makes contact with the box and causes it to start moving also, another segment boundary is recognised. In this way, the robot constructs a very general movement-based description of the actions of the teacher. In the case of the tube problem, the segments correspond to the robot moving to pick up the stick, placing it against the box, pulling the box from the tube, placing the stick down and finally, moving to pick up and carry away the box.

Once the learner has segmented the trainer's trace, it attempts to match segments to its existing set of action models. Each of the robot's action models incorporates a STRIPS model along with a list of the objects that are moved during the action. A model is matched to a segment by checking that three conditions are met: the moving objects in the demonstration match the model; the preconditions of the model are true at the start of the action segment; the effects of the model have been achieved by the end of the action segment.

In the unusual case where more than one abstract action matches a segment, the segment is labelled with the action with the most specific set of preconditions. If the two sets cannot be distinguished, one is chosen at random. This may affect the learning time if a poor choice is made but will ultimately be corrected if an experiment fails. Segments that cannot be matched to any existing action model are labelled as components of a novel action. In the tube example, this produces the following labelling of the teacher's demonstration:

goto(stick), pickup(stick),
novel-action(stick, box),
drop(stick), goto(box), pickup(box).

where **novel-action(stick,box)** is actually a compound action composed of two unrecognised actions. The first is a positioning action in which the stick is moved by the teacher. The second is the actual tool action that pulls the box from the tube, i.e., the stick and box are moved together. The learner must now try to explain how these two consecutive actions were used to achieve the goal of acquiring the box. It does so by constructing two action models, for positioning and tool use, that are consistent with the other actions in its explanation of the demonstration.

We use an heuristic that actions occurring before a novel action should enable the novel action's preconditions. Similarly, the effects of the novel action should help enable the preconditions of actions occurring later in the demonstration. This heuristic is based upon the assumption that the teacher is acting rationally and optimally, so that each action in the sequence is executed in order to achieve a necessary sub-goal. The program constructs a STRIPS model by examining the start and end states of the novel action and identifies relevant literals in the actions executed prior to and after the novel action. The effects of the novel action are defined as any state changes that occur during the action that support an action precondition later in the demonstration.

In the case of the tube problem, **¬in(box,tube)** becomes true during the novel action segment. This effect enables the preconditions of the **pickup(box)** action that occurs later in the demonstration. In general, there may be many irrelevant effects. However, this explanation-based reasoning allows us to eliminate all but the effects that were important for achieving the goal.

The preconditions of the novel tool action are constructed by a similar argument. The learner examines the world state at the start of the novel action and identifies the subset of state literals that were produced by the effects of earlier actions. The effect **gripping(stick)** occurs before the novel action and is still true at the start of the action. Since it is a known effect of the **pickup(stick)** action, it is considered a relevant literal to include in the novel action preconditions.

A common pattern for many tool-use problems is that there is a positioning step, followed by the application of the tool. We use this as a template for constructing two action models for the tube problem. The preconditions and effects identified in the demonstration are converted to literals with parameters by simply substituting each instance of an object with a unique variable.

position-tool(Tool, Box)

PRE	in-gripper(Tool), gripping,
ADD	**tool-pose(Tool, Box)**, obstructing(Tool, Box)
DEL	–

The first action represents the robot getting itself into the correct position so that the tool can be applied. Note that we have not yet determined what that position is. The predicate **tool-pose(Tool,Box)**, which expresses this position, will be learned in the experimentation stage. A side-effect of this action is that the tool is obstructing the object. Later, when the robot tries to pick up the object, this side-effect will have to be undone. The tool action is:

pull-from-tube(Tool, Box, Tube)

PRE	**tool-pose(Tool, Box)**, gripping(Tool), in-tube(Box,Tube)
ADD:	–
DEL:	in-tube(Box, Tube)

The **tool-pose(Tool,Box)** effect of the position-tool action becomes a precondition of the tool action. There is only one subgoal of the tool action in this case, corresponding to an object being removed from the tube. This model becomes the starting point for learning the **tool-pose(Tool,Box)** predicate, which we describe next.

5 Learning by Experimentation

The robot must learn the *tool pose state*, i.e. the state in which the correct object has been selected as the tool and the pose in which it can be applied successfully. It does so by testing a variety of tools and tool poses in a series of learning tasks. Thus, the trial-and-error learning has two components: generating new learning tasks and updating the robot's hypothesis for the tool-pose concept depending on the outcome of the experiment. The process for learning the concept describing the tool pose state is as follows:

1. Select a tool that satisfies the current hypothesis and place it in a pose defined by the spatial constraints in the hypothesis. Generate a motion plan that solves the task from this state and execute it, observing whether the action sub-goal is achieved.
2. If the action achieved the desired sub-goal, label the initial state as a positive example. If the action failed to achieve the desired sub-goal, label it as a negative example.
3. If the example was positive, generalise the hypothesis. If the example was negative specialise the hypothesis.
4. If the current task was successfully solved, a new learning task is generated and presented to the robot. If the robot failed then the current task is reset.
5. The robot continues its experimentation until the agent is able to solve a pre-defined number of consecutive tasks without failure.

The robot's world contains a selection of tool objects available for solving the problem. The dimensions and shapes of the tools are chosen randomly according to a type definition specified by the user. Figure 1 shows some tools that are available for the tube task. Only the hook tool is suitable for solving the task.

We use a version-space representation for the hypothesis [2], where we maintain both a most-specific and most-general clause representing the concept.

$$\begin{aligned} &h_S \leftarrow \text{saturation}(e_0). \\ &h_G \leftarrow \text{true}. \end{aligned}$$

e_0 is the initial example derived from the teacher's demonstration. The saturation of e_0 is the set of all literals in e_0 plus all literals that can be derived from e_0 and the robot's background knowledge. The most-general clause, initially, covers every possible instance of the hypothesis space.

Since our learner must generate experiments to find positive examples, we generalise conservatively by trying to test examples that are close to the most-specific boundary, where positive examples are more likely to be found. However, the initial most-specific hypothesis contains many irrelevant literals. We need a way of creating a specific example that avoids these irrelevant literals as much as possible. We do this by starting from with most-general clause and then specialise it by adding literals from the most-specific clause. This is similar to Progol, except that we take advantage some domain knowledge.

This process is divided into two. The tool pose hypothesis incorporates both structural literals (describing the physical composition, structure, and shape of the tool) and spatial literals (describing how the tool should be placed relative to other objects). We first select a tool that best matches h_S and then try to select a tool pose that matches as closely as possible that specified in h_S. To select a tool, each available tool object is scored for suitability according to the number of literals in h_S it satisfies. It must also satisfy all of the structural constraints in h_G.

Pose selection, similarly, tries to maximise the number of satisfied literals in h_S. In this case, however, we are interested in satisfying the greatest number of spatial literals. This allows the agent to place its selected tool in a pose that is as similar as possible to the previous positive examples it has observed. The process for selecting a suitable subset of spatial literals requires that all spatial constraints in h_G are satisfied. This ensures that the most important constraints are applied first. We then successively apply spatial constraint literals from h_G, using the constraint solver to check, at each step, whether the cumulative constraints can be satisfied.

Once the tool and pose have been selected, a plan is generated and executed by the robot. Figure 3 shows three examples generated for the tube problem. If the plan succeeds, we have a new positive example, which is used to generalise h_S. If it fails, h_G must be specialised to exclude the new negative example.

The most-specific clause is built using only positive examples. We use a constrained form of least general generalisation [1] to find the minimal generalisation that covers the positive examples. The most-general clause is built by generalising the most-specific clause. Negative-based reduction is used to simplify the most-specific clause without covering additional negative examples. This is a process of successively eliminating literals that do not change the coverage of the clause. We use negative-based reduction to recreate h_G when a new negative example arrives.

The resulting algorithm borrows heavily from GOLEM [8]. The main difference is that GOLEM's bias is to learn the shortest, most-general hypothesis

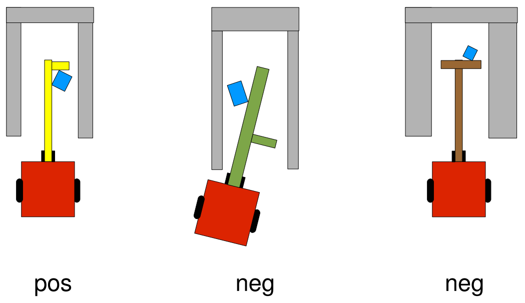

Fig. 3. Positive and negative examples of tool use

possible. In contrast, we maintain both a most-specific and most-general boundary on the hypothesis space. The algorithm is rerun after each example is received and the current hypothesis is rebuilt.

6 Evaluation

This system has been implemented and tested on several learning problems using the Gazebo robot simulator [9]. The robot is a Pioneer 2, equipped with a simple gripper. In this section, we trace the robot's learning on the pull-from-tube-task. After the robot observes the trainer's demonstration, the first step is to segment the trace of the trainer's behaviour into discrete actions. Figure 4 shows the results of clustering object motions, as described earlier.

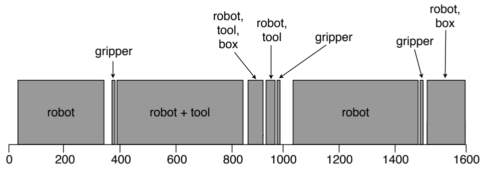

Fig. 4. Robot and object motion during the teacher's demonstration. Units are 10ths of seconds.

The *primitive state* of the world is given by the positions and orientations of all objects and is shown in Figure 5. From this, the robot builds an *abstract state* description that describes the properties and relationships between objects and agents in the world. The abstract state is expressed in predicate logic and is generated from a set of definitions provided to the agent as background knowledge, which consists of action models for known actions. These are shown in Figure 6. In addition to these abstract actions, the agent also has two manipulation recognition models, shown in Figure 7.

The robot constructs an explanation of the teacher's activities (Table 1) by matching action models to the motion segments it has identified. A model

Table 1. Explanations of teacher's demonstration

Seg	*Moving objects*	*Explanation*
1	robot	*put_in_gripper(hookstick)*
2	gripper	*grip(hookstick)*
3	robot, hookstick	*recognise_carry_obj(hookstick)*
4	robot, hookstick, box	*??*
5	robot, hookstick	*move_obstacle(hookstick,box)*
6	gripper	*ungrip(hookstick)*
7	robot	*remove_from_gripper(hookstick),put_in_gripper(box)*
8	gripper	*grip(box)*
9	robot, box	*recognise_carry_obj(box)*

```
tool_pose_S(Tool,Box,State):-

% static literals:

attached(Tool,Hook),
num_attachments(Tool,1),
num_attachments(Box,0),
longest_component(Tool),
narrower(Tool,Box),
shorter(Box,Tool),
shape(Tool,sticklike),
shape(Box,boxlike),
closed_tube(Tube,TubeLeft,TubeRight,TubeBack),
attached_side(Tool,Hook,right),
attached_side(TubeLeft,TubeBack,right),
attached_side(TubeRight,TubeBack,left),
attached_end(Tool,Hook,front),
attached_end(TubeLeft,TubeBack,front),
attached_end(TubeRight,TubeBack,front),
attached_angle(Tool,Hook,right_angle),
attached_angle(TubeLeft,TubeBack,right_angle),
attached_angle(TubeRight,TubeBack,right_angle),
attached_type(Tool,Hook,end_to_end),
attached_type(TubeLeft,TubeBack,end_to_end),
attached_type(TubeRight,TubeBack,end_to_end),
num_attachments(Hook,0),
num_attachments(TubeLeft,1),
num_attachments(TubeRight,1),
num_attachments(TubeBack,0),
narrower(Hook,Tool),
narrower(Hook,Box),
narrower(Tool,TubeLeft),
narrower(TubeLeft,Box),
narrower(Hook,TubeLeft),
narrower(Tool,TubeRight),
narrower(TubeRight,Box),
narrower(Hook,TubeRight),
narrower(Tool,TubeBack),
narrower(TubeBack,Box),
narrower(Hook,TubeBack),
narrower(TubeBack,TubeLeft),
narrower(TubeBack,TubeRight),
shorter(Hook,Tool),
shorter(Tool,TubeLeft),
shorter(Tool,TubeRight),
shorter(TubeBack,Tool),
shorter(Box,Hook),
shorter(Box,TubeLeft),
shorter(Hook,TubeLeft),
shorter(TubeBack,TubeLeft),
shorter(Box,TubeRight),
shorter(Hook,TubeRight),
shorter(TubeBack,TubeRight),
shorter(TubeBack,TubeLeft),
shorter(Box,TubeBack),
shorter(Hook,TubeBack),
shape(TubeLeft,sticklike),
shape(TubeRight,sticklike),
shape(TubeBack,sticklike),

% dynamic literals:

in_gripper(Tool,State),
touching(Tool,Box,right,State),
at_oblique_angle(Tool,Box,State),
in_tube(Box,Tube,State),
in_tube_end(Box,Tube,front,State),
in_tube_side(Box,Tube,right,State),
touching(Hook,Box,back,State),
at_right_angles(Hook,TubeLeft,State),
at_right_angles(Hook,TubeRight,State),
at_right_angles(Tool,TubeBack,State),
at_oblique_angle(Box,Hook,State),
at_oblique_angle(Box,TubeLeft,State),
at_oblique_angle(Box,TubeRight,State),
at_oblique_angle(Box,TubeBack,State),
parallel(Tool,TubeLeft,State),
parallel(Tool,TubeRight,State),
parallel(Hook,TubeBack,State),
in_tube(Hook,Tube,State),
in_tube_end(Hook,Tube,front,State),
in_tube_side(Hook,Tube,right,State).
```

Fig. 5. The initial most-specific hypothesis clause h_S, formed from the trainer's example

```
grip(Obj)

PRE     ¬gripping,
        in_gripper(Obj)
ADD     gripping
DEL     -
PRIMTV  closeGrip
MOVING  -
```

```
ungrip(Obj)

PRE     gripping,
        in_gripper(Obj)
ADD     -
DEL     gripping
PRIMTV  openGrip
MOVING  -
```

```
remove_from_gripper(Obj)

PRE     in_gripper(Obj),
        ¬gripping
ADD     empty_gripper
DEL     in_gripper(Obj)
PRIMTV  back
MOVING  robot
```

```
put_in_gripper(Obj)

PRE     forall(Tube:tube, ¬in(Obj,Tube)),
        empty_gripper,
        ¬gripping,
        forall(Obstacle:obj,
          ¬obstructing(Obstacle,Obj)
ADD     in_gripper(Obj)
DEL     empty_gripper
PRIMTV  fwd,back,rotleft,rotright
MOVING  robot
```

```
move_obstacle(ObjA,ObjB)

PRE     moveable_obj(ObjA)
        obstructing(ObjA,ObjB)
        in_gripper(ObjA),
        gripping
ADD     -
DEL     obstructing(ObjA,ObjB)
PRIMTV  fwd,back,rotleft,rotright
MOVING  robot, ObjA
```

Fig. 6. Action models provided as background knowledge for the pull-tool problem

```
recognise_goto

PRE     empty_gripper
MOVING  robot
```

```
recognise_carry_obj(Obj)

PRE     in_gripper(Obj),
        gripping
MOVING  robot, Obj
```

Fig. 7. Manipulation recognition models provided to the agent for the pull-tool problem

matches a segment if the action's preconditions are true at the beginning of the segment and the effects have been achieved by the end of the segment. The moving objects in the segment must also match the moving objects named in the corresponding action model.

At this point, the explanations are turned into the incomplete action models **position-tool** and **pull-from-tool**. The robot now enters its experimentation phase in which it constructs h_G and h_S for the definition of the **tool-pose** predicate. Each new example causes the learner to update the version space. The sequence of experiments is show in figure 8. After twelve experiments, the h_G is:

```
tool-pose$_G$(Tool,Box,State) $\leftarrow$
    in-tube-side(Box,Tube,Side,State),
    attached-side(Tool,Hook,Side),
    touching(Hook,Box,back,State),
    attached-angle(Tool,Hook,rightangle),
    attached-end(Tool,Hook,back).
```

This states that the tool must have a hook attached at a right angle and be on the same side as the target object in the tube. The hook must be touching the back of the object.

We evaluate the robot's performance by the number of experiments required before it can consistently solve a new task. Twelve experiments were required to learn **pull-from-tube**. A similar task, learning to extract an object from an open tube by pushing it through the tube, requires between 10 and 15 experiments, depending on random variations in the tools made available to the robot for its experiments. A third experiment was the classic Shakey problem of pushing a ramp up to a platform so that the robot can push a block off the platform.

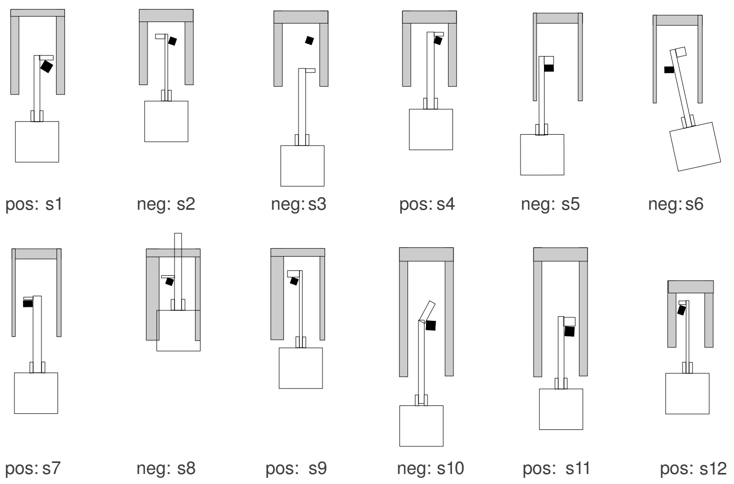

Fig. 8. Examples generated during learning by experimentation

Between 9 and 13 experiments were required to learn that the ramp can be used as a tool to mount the platform.

7 Related Work

Tool use has received relatively little attention in Artificial Intelligence and robotics research. Bicici *et al* [10] provides a survey of earlier AI research related to the reasoning and functionality of objects and tools. Perhaps the first attempt to build a robot agent specifically tailored towards learning and tool-use was report by Wood *et al.*, [11]. In this work an artificial neural network was used to learn appropriate postures for using reaching and grasping tools, on board the Sony Aibo robot platform. Stoytchev [12] has implemented an approach to learning tool-using behaviours with a robot arm. The robot investigates the effects of performing its innate primitive behaviours (including pushing, pulling, or sideways arm movements) whilst grasping different reaching tools provided by the user. The results of this exploratory learning are used to solve the task of using the tool to move an object into a desired location.

Although there is little other research that directly addresses tool use learning, work in the area of learning models of agent actions is relevant to our approach. Gil [13] used learning by experimentation to acquire planning knowledge. Benson [14] used ILP to learn action models of primitive actions and then combined them into useful behaviours. Other work has since focused on learning action models for planning in more complex environments, allowing for stochastic action [15] or partial observability [16]. Levine and DeJong [17] also used an explanation-based approach to acquiring planning operators.

8 Conclusion

This research contains several novel features. We integrate tool-use learning and problem solving in a robot that learns new tool actions to help it solve a planning problem. We have devised a novel action representation that integrates symbolic planning, constraint solving and motion planning. The system incorporates a novel method for incremental learning that uses relative least general generalisation but adopts a version-space representation. This is the basis for a new method for generating examples in an incremental setting where cautious generalisation is desirable. Our approach is based upon sampling near the most-specific hypothesis boundary in the version space. In addition, our robot is able to infer the approximate form of a new action model by observing a teacher and examining the context of the plan in which it is executed. The robot also uses trial-and-error to refine this approximate model.

There are several important issues that we do not address in this work. Some tool-use behaviours require complex non-linear control of the tool or target object. Learning these types of behaviours is a field of research in itself and we do not address it here. Instead, we have chosen tasks requiring simple manipulation that can be achieved with a generic controller. The learning tasks we attempt are restricted to those that can be easily implemented in a rigid-body simulation. Problems involving liquids or deformable bodies are not considered. Nevertheless, the general learning system is intended to be applicable to a wide range of scenarios. We do not try to handle learning by analogy. For example, using a ladder to reach a light bulb and using a stick to reach an object on a shelf are conceptually similar problems.

References

1. Plotkin, G.: A note on inductive generalization. In: Meltzer, B., Mitchie, D. (eds.) Machine Intelligence, vol. 5, pp. 153–163. Edinburgh University Press (1970)
2. Mitchell, T.: Generalization as search. Artificial Intelligence 18, 203–266 (1982)
3. Fikes, R., Nilsson, N.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2(3-4), 189–208 (1971)
4. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research 14 (2001)
5. Apt, K.R., Wallace, M.G.: Constraint Logic Programming using ECLiPSe. Cambridge University Press (2007)
6. Kuffner, J., LaValle, S.: RRT-Connect: An efficient approach to single-query path planning. In: International Conference on Robotics and Automation (April 2000)
7. Brown, S.: A relational approach to tool-use learning in robots. PhD thesis, School of Computer Science and Engineering, The University of New South Wales (2009)
8. Muggleton, S., Feng, C.: Efficient induction of logic programs. In: Muggleton, S. (ed.) Inductive Logic Programming, pp. 281–298. Academic Press (1992)
9. Koenig, N., Howard, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: International Conference on Intelligent Robots and Systems, vol. 3, pp. 2149–2154 (September 2004)
10. Bicici, E., St Amant, R.: Reasoning about the functionality of tools and physical artifacts. Technical Report 22, NC State University (2003)
11. Wood, A., Horton, T., St Amant, R.: Effective tool use in a habile agent. In: IEEE Systems and Information Engineering Design Symposium, pp. 75–81 (April 2005)
12. Stoytchev, A.: Behaviour-grounded representation of tool affordances. In: International Conference on Robotics and Automation (April 2005)
13. Gil, Y.: Learning by experimentation: Incremental refinement of incomplete planning domains. In: International Conference on Machine Learning (1994)
14. Benson, S.: Learning action models for reactive autonomous agents. PhD thesis, Department of Computer Science, Stanford University (1996)
15. Pasula, H.M., Zettlemoyer, L.S., Kaelbling, L.P.: Learning symbolic models of stochastic domains. Journal of Artificial Intelligence Research 29, 309–352 (2007)
16. Amir, E.: Learning partially observable deterministic action models. In: International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, Scotland, UK, pp. 1433–1439 (August 2005)
17. Levine, G., DeJong, G.: Explanation-based acquisition of planning operators. In: ICAPS, pp. 152–161 (2006)