Learning Interactions Among Objects, Tools and Machines for Planning

Mustafa Ersen and Sanem Sariel Talay
Department of Computer Engineering
Istanbul Technical University
Istanbul, Turkey
{*ersenm,sariel*}*@itu.edu.tr*

***Abstract*— We propose a method for learning interactions among objects when intermediate state information is not available. Learning is accomplished by observing a given sequence of actions on objects. We have selected the Incredible Machine game as a suitable domain for analyzing and learning object interactions. We first present how behaviors are represented by finite state machines using the given input. Then, we analyze the impact of the input type corresponding to relations on the overall performance. Our analysis includes four different types of input: a knowledge base including part relations; spatial information; temporal information; and spatio-temporal information. We show that if a knowledge base about relations is provided, learning is accomplished to the desired extent. Our analysis also indicates that spatio-temporal analysis is superior to spatial and temporal analyses and gives similar results to that of the knowledge-based approach.**

Keywords- agent-based systems; automated planning; learning; knowledge representation

I. INTRODUCTION

Automated action planning is the process of finding a sequence of actions to achieve a given goal. Whenever the planning problem (i.e., initial and goal states) and operators corresponding to domain actions are defined properly, a planner may easily find a solution. However, the problem is complicated if the planning agent has not complete information about preconditions and effects of the domain operators. In this case, the problem turns into a learning task where relations of actions and objects (in parameters of actions) are learned through observation of behaviors.

The main goal of this research is to develop a method for learning object interactions through actions. Learning is to be performed by observing a given sequence of actions. We also analyze the performance of the learner based on the completeness of the knowledge base about relations. We believe the outcomes of this study are also useful for robot learning tasks. Our future work includes implementation of this approach on real robots for learning affordances.

We have selected the Incredible Machine [1] as a domain for analyzing these interactions because this domain allows the use of various objects and relations to achieve a given goal. The Incredible Machine is a series of computer games in which the player tries to solve given puzzles by constructing contraptions in cartoonist Rube Goldberg's style [2]. In a typical scenario of this game, the player is given a limited number of objects, tools and machines and a goal to accomplish by using these resources. An example screenshot is provided in Fig. 1 to illustrate the environment of the game. In this example, the aim is opening the can, starting the mixer and heating up some coffee. To complete the given structure to reach the goal state, the player is provided with some objects in the parts bin on the right side, and these parts must be used without changing the position of already installed objects in the environment. The puzzles in the game are interesting as they require building complex systems by considering various interactions among the objects to achieve the given goals.

This problem fits into planning framework as it involves actions to transform a given initial state into a goal state. Both progression and regression planning is applicable if domain actions are perfectly defined. A planner needs to consider the game in two aspects to be able to devise a consistent plan. First, considering interactions among objects, tools and machines are crucial for making decisions while connecting given parts altogether to reach the goal state. Second, physics models of objects (e.g., the effect of the gravity on the motion of a ball) should be taken into account to estimate interactions through motions.

In this work, we focus on solving the stated learning problem considering object interactions given in a tutorial to understand the rationale behind the Incredible Machine puzzles. We assume that the tutorial presents object interactions in the form of chain reactions (e.g., starting a mixer, running a motor, lighting a lamp, etc.) by connections among each other. No further background information is available about the types of objects and their

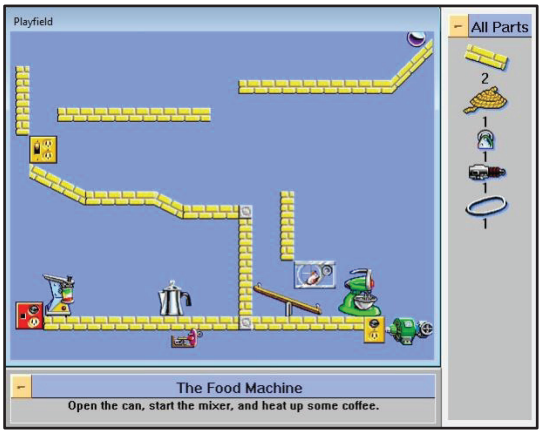


Figure 1. An example puzzle from the Incredible Machine 3.

semantics, nor intermediate state information. Particularly, in this work, tutorials are presented as text-based action sequences. Extension of the work by processing visual observations (as in the actual game) is left as a future work. The problem is investigated from four perspectives and an integrated solution is provided. If relations of parts are provided to the agent, a logic-based reasoning mechanism could be used. Otherwise, we show that considering spatial, temporal and spatio-temporal aspects of the problem makes it feasible to reason about interactions for further planning tasks. These four perspectives may correspond to different learning problems. The Incredible Machine domain provides a common framework to integrate these approaches.

The rest of the paper is organized as follows. First, we discuss some related works in the field of learning action schema. Next, we give an overview of our approach and show the details step by step on a running example. Third, we show how to use resulting finite state machines (FSM) and connections among them on a planning example. Finally, we discuss the results of four different perspectives and conclude the paper.

II. Background

In recent years, efficient approaches have been developed to learn planning operators from example plans [3-12]. From these, SLAF (Simultaneous Learning and Filtering) [4-6] is a tractable system for identifying preconditions and effects of actions in partially observable domains. It builds exact action schema efficiently by using a relational logical representation and uses partial observations between executed actions to identify consistent action models. However, this approach requires the use of observations on the states of objects between actions.

Unlike SLAF, ARMS (Action-Relation Modeling System) [7,8] can learn action models without using intermediate state information. ARMS is based on solving a weighted MAX-SAT problem on the statistical distribution of related action and predicate sets having some common parameters. It is an effective approach to find approximate STRIPS action schema by using only actions and predicates in the plan trace. However, this approach cannot learn actions with conditional effects for identifying interactions among objects, tools and machines in our problem.

The LAMP (Learning Action Models from Plan traces) [9] system, learns more expressive action schema having quantifiers and logical implications in accordance with PDDL standards. This approach is based on enumerating six different patterns of candidate formulas and their quantified cases on training data to be able to learn preconditions, effects and conditional effects. Then, weights are assigned to generated formulas by using a Markov Logic Network. In LAMP, partial observations between action executions are important to be able to decrease error rate in the results.

LOCM (Learning Object-Centred Models) [10,11] is an object-centered approach for learning action models from plan traces. It's unique in its input as it only requires the action sequence to be given to learn action schema. Moreover, LOCM outputs FSMs showing behaviors of different type of objects in the domain, as well as planning operators. Thus, for our work, LOCM is the most suitable system to use for determining behaviors of objects. This approach is discussed in detail in the following sections. LOCM2 [12] is proposed to solve the state generalization problem of LOCM for objects showing behaviors in separate aspects. By analyzing example plans in a transition-based representation, LOCM2 weakens the assumptions of LOCM and produces additional state machines to capture behaviors that cannot be modeled by LOCM.

We extend LOCM in two ways. First, we include different orientations of the same object in the input to be able to distinguish change of behavior due to changing orientations. Second, we propose an approach to determine interactions among objects when relational information is given. To be able to produce conditional effects in a chain reaction, our approach uses a forward chaining mechanism. Furthermore, we analyze the use of spatial and temporal information when a knowledge base of relations is not available.

III. Learning Interactions from Examples

Learning is essential for an agent with limited knowledge about its environment. The problem that we investigate in this work asks for learning interactions of actions and objects for the Incredible Machine game, given some tutorials about this game. Tutorials include sequential actions on various objects represented in a text-based format. Aside from these tutorials, there exists no prior knowledge about actions, objects and their interactions.

A tutorial is a tuple $T = (\mathcal{A}, \mathcal{F})$ where $\mathcal{A}$ is an action sequence with n actions $A_{1:n}$ in their order of occurrence where concurrent actions affect different objects $\left(\forall i,j\ i < j \Rightarrow A_i \preccurlyeq A_j\right)$ and $\mathcal{F}$ is a list of predicates showing different geometric features of objects (e.g., $facing\_left$ or $facing\_right$) in the environment. Each action A_i has m_i arguments where each argument is an object in the domain $\left(\mathrm{Arg}_{1:m_i}(A_i) = O_{i,k},\ 1 \le k \le m_i\right)$.

The main objective is to model behaviors of different type of objects represented as FSMs (per object type). Each FSM involves a state set $\mathcal{S}$ and a state-transition function δ. Furthermore, conditional relations among these FSMs should be established to capture interactions among different type of objects and determine preconditions of actions given in $\mathcal{A}$. Eventually, these preconditions together with behavior models are to be used for generating new plans.

IV. Proposed Approach

A. Overview and a Running Example

Our approach consists of two phases. In the first phase, objects are grouped into sets with respect to their type; and a finite state machine is generated for each set to model

behaviors of objects belonging to that set. State machines are created using the steps of LOCM. We propose a modification to address interactions based on different geometric features of objects.

In the second phase, interactions among objects from different sets are determined by establishing conditional links between related states. Connections and relations among objects, tools and machines are used while generating these conditional structures to model interactions.

Throughout the paper we illustrate our approach on an example tutorial whose initial state is visualized in Fig. 2. This example tutorial includes various objects and interactions. The objects presented in the tutorial (except the balls) are not affected by the gravity and motion laws. In this scenario, after dropped from a certain height, $ball_1$ turns on $switch_1$, and this initiates two concurrent chains of reactions. The action sequence given below represents this tutorial in a text-based format.

A_1: $push\_down(ball_1, switch_1)$
A_2: $start(motor_1)$
A_3: $start(motor_2)$
A_4: $spin\_clockwise(conveyorbelt_1)$
A_5: $spin\_counterclockwise(conveyorbelt_2)$
A_6: $slide\_right(ball_2)$
A_7: $slide\_left(ball_3)$
A_8: $press(ball_1, switch_3)$
A_9: $light(flashlight_1)$
A_{10}: $start\_mixing(mixer_1)$
A_{11}: $push(ball_3, switch_2)$
A_{12}: $activate(ball_2, remotecontroller_1)$
A_{13}: $blow\_up(dynamite_1)$
A_{14}: $make\_toast(toaster_1)$
A_{15}: $lower(bucket_1)$
A_{16}: $start\_running(mandrillmotor_1)$

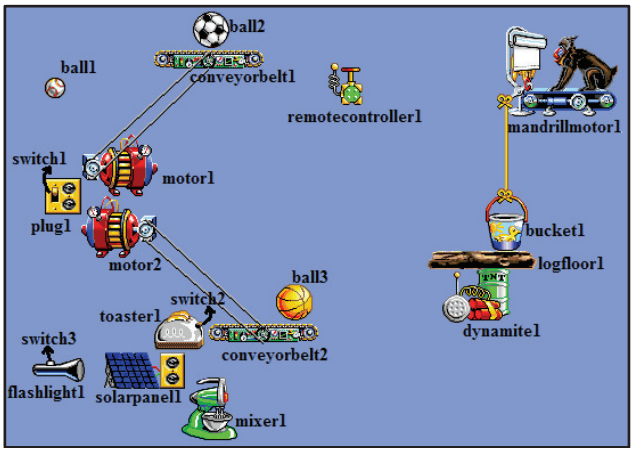


Figure 2. An example scenario including interactions among various objects. $motor_1$, $motor_2$, $mixer_1$ and $toaster_1$ require electricity to work. $motor_1$ and $motor_2$ are plugged to $plug_1$ which is controlled by $switch_1$. $toaster_1$ and $mixer_1$ are plugged to $solarpanel_1$ which can generate electricity from light. $flashlight_1$ is directed at $solarpanel_1$. To make toast, $switch_2$ on $toaster_1$ needs to be pushed. $motor_1$ drives $conveyorbelt_1$ by a belt and similarly $motor_2$ drives $conveyorbelt_2$. $dynamite_1$ is remotely controlled by $remotecontroller_1$. $logfloor_1$ dissolves when $dynamite_1$ explodes. The monkey runs $mandrillmotor_1$ if the shade covering bananas is pulled by a rope.

B. *Phase-1: Creating FSMs Reflecting Behaviors*

In the first phase of our approach, objects are grouped with respect to their type; and behaviors are modeled by using a unique finite state machine for each group of objects. As our system initially applies the steps given in LOCM, we first repeat the existing underlying assumptions here, for convenience. Then, we present our extension to model different features of objects.

LOCM groups objects into sets called *sorts*; and the behavior of each *sort* is modeled by using a single state machine per *sort*. FSMs are created based on the following assumptions made by LOCM:

1. Each argument of the same type of action is restricted to contain objects of the same *sort*.
2. Each action causes a transition on each object in its arguments. Some transitions may not change the state of an object.
3. The same transition cannot appear more than once in a FSM belonging to the *sort* it affects.

By using Assumption 1, LOCM divides objects appearing in action parameters into a set structure called *sorts* where each *sort* is a disjoint subset of all the objects in the domain. If an action name $name(A)$ with p parameters appears in both i^{th} and j^{th} entries in the action sequence $\mathcal{A}$, the corresponding objects $O_{i,k}$ and $O_{j,k}$ for each parameter $1 \leq k \leq p$ of actions A_i and A_j share the same *sort*. In this work, we use this assumption to determine the type of each object.

When we consider $ball_1$, $motor_1$, $motor_2$ and $plug_1$ objects in Fig. 2, after $ball_1$ pushes $switch_1$ of $plug_1$ down, both $motor_1$ and $motor_2$ start running. This is given in the action sequence $\mathcal{A}$ as actions $A_{1:3}$. In this sequence $name(A_2) = name(A_3) = start$, thus the objects in $\text{Arg}_1(A_2) = O_{2,1}$ and $\text{Arg}_1(A_3) = O_{3,1}$ are grouped into the same *sort* as $\{motor_1, motor_2\}$. Grouping is done similarly for the other related objects in the domain.

After grouping objects into *sorts*, LOCM uses the input action sequence $(\mathcal{A})$ to generate finite state machines that model the behavior of each *sort*. Each action $A_i \in \mathcal{A}$ is assumed to cause a transition for each object k in its arguments where $1 \leq k \leq p$ (Assumption 2). Each transition $A_{i,k}$ has a start and an end state, namely $start(A_{i,k})$ and $end(A_{i,k})$. The end state of a transition could be the same as its start state. All transitions are assumed to be one-to-one, thus $start(A_{i,k}) = start(A_{j,k})$ and $end(A_{i,k}) = end(A_{j,k})$ for each occurrence of the same action A in $\mathcal{A}$. $\big(A = name(A_i) = name(A_j)\big)$.

After assigning a start and an end state for each transition in the action sequence, LOCM utilizes the continuity of transitions on the same object instance to determine equivalent states in the FSM of each *sort*. Transitions $A_{i,k}$ and $A_{j,l}$ are said to be consecutive with

respect to an object O iff $O = O_{i,k} = O_{j,l}$, and O does not occur in the parameters of action instances between i and j.

To illustrate how this assumption is used, consider $ball_1$ object in the example action sequence. This object is included in the first parameter of both $A_1 = push\_down$ and $A_8 = press$. In the beginning, the corresponding transitions are initialized as,

$$state_{1,1} \xrightarrow{push\_down_1} state_{2,1}$$
$$state_{3,1} \xrightarrow{press_1} state_{4,1}$$

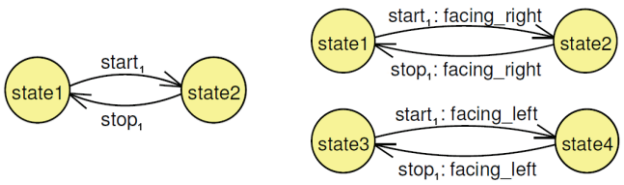
By using the continuity of transitions, end state of $push\_down_1$ and start state of $press_1$ are unified as

$$state_{1,1} \xrightarrow{push\_down_1} state_{2,1}$$
$$state_{2,1} \xrightarrow{press_1} state_{4,1}$$

where $state_{i,j}$ means the i^{th} state of the j^{th} *sort* in the training set and $ball_1$ is assigned 1 as its *sort* identifier because it is the first object in the action sequence.

The basic steps of LOCM are useful to create corresponding state machines. However, it is not capable of modeling geometric features of an object defined in $\mathcal{F}$. For example, $motor_1$ and $motor_2$ are instances from the same *sort*, but they are placed in opposite directions. As a result, $conveyorbelt_1$ and $conveyorbelt_2$ (which are connected to $motor_1$ and $motor_2$ by using belts, respectively) rotate in opposite directions. For the motor *sort*, LOCM produces the FSM given in Fig. 3(a). This FSM does not capture the difference between these rotations. Neither LOCM2 features this property as action names are the same in both cases.

We have extended LOCM to deal with these issues. When objects are placed in opposite directions, their effects may be different as for the two motors given in Fig. 2. If their features are stated as reflecting a change in their orientations, our proposed extension can detect the difference in effects. When the input includes the following facts: $facing\_left(motor_1)$ and $facing\_right(motor_2)$, the desired output is formed as in Fig. 3(b). This is achieved by considering the difference in features in the FSMs. We assume that each different feature of the same object causes a change in the effects of transitions on this object. Thus, in our approach, transitions on objects in changing forms are assigned different start and end states. Therefore, two FSMs are created for the motor *sort*. *state1* and *state3* are different from each other as the same motor object cannot change its form runtime between $facing\_right$ and $facing\_left$. Similarly, the end states are different.



(a) FSM produced by LOCM (b) FSMs produced by our system

Figure 3. Resulting finite state machines for the motor *sort*.

C. Phase-2: Modeling Interactions through Relations

In the second phase of our system, connections among objects from different *sorts* are used to determine and model interactions among objects through actions. We analyze four different input types. First, we investigate the case where background knowledge on relations is given. In this case, a forward chaining approach could exploit these relations to make conclusions. In the second and third analyses, we investigate how the results change if spatial locality of objects or temporal locality of actions is taken into account without prior domain knowledge. Finally, we analyze the use of spatial and temporal information together to infer interactions among objects, tools and machines.

1) Using a Knowledge-Based Representation

To capture interactions among different state machines, first we encoded a knowledge base consisting of predicates showing directly observable connections among objects for the given tutorial. Our system uses these predicates to reason about interactions among objects. These predicates are shown in Table 1 along with their direction of connection and some related examples. Note that, the knowledge base does not contain semantic information about the types of objects.

TABLE 1: TYPE OF CONNECTIONS AMONG OBJECTS

Predicate	Examples from Fig. 2	Direction
has(obj1,obj2)	has(plug1,switch1) has(toaster1,switch2) has(flashlight1,switch3)	obj2 → obj1
plugged(obj1,obj2)	plugged(motor1,plug1) plugged(motor2,plug1) plugged(mixer1,solarpanel1) plugged(toaster1,solarpanel1)	
belt(obj1,obj2)	belt(conveyorbelt1,motor1) belt(conveyorbelt2,motor2)	
on(obj1,obj2)	on(ball2,conveyorbelt1) on(ball3,conveyorbelt2) on(bucket1,logfloor1)	
facing(obj1,obj2)	facing(flashlight1,solarpanel1)	obj1 → obj2
rope(obj1,obj2)	rope(mandrillmotor1,bucket1)	obj1 ↔ obj2
near(obj1,obj2)	near(dynamite1,logfloor1)	

When these facts are provided, our system applies a forward chaining mechanism to create a list of connections for each object in the domain. For example, there are two applicable connections for $toaster_1$ in Fig. 2: $has(toaster_1, switch_2)$, $plugged(toaster_1, solarpanel_1)$. Thus, the forward chaining system creates two subsets of connections for $toaster_1$ as $\{[switch_2], [solarpanel_1]\}$ in the first iteration. The future iterations of forward chaining uses transitivity of these connections to determine chains of relations. Considering new relations in the chain, the connection set for $toaster_1$ is extended to $\{[switch_2], [solarpanel_1, flashlight_1, switch_3]\}$.

As some of the objects are not available in the action sequence $\mathcal{A}$, they are removed from the list of connections. For example, objects $solarpanel_1$, $plug_1$and $logfloor_1$ are removed from final connection sets because there is not any information on these objects. The final connection set for $toaster_1$ becomes {[$switch_2$], [$flashlight_1, switch_3$]} after these eliminations.

Finalized connection sets for all objects are used in a process to determine interactions among objects through actions given in $\mathcal{A}$. It is important to determine the preconditions as a complete set of facts to execute an action given a chain of reactions. However, this is not completely possible because state information (i.e., states of objects) is not available as an input. The intuitive idea that we apply here includes the use of the last effect for each subset of the resulting connection set. Therefore, consecutive transitions among connected objects are extracted from the action sequence $\mathcal{A}$ by using this intuition. For example, the preconditions of $A_{14} = make\_toast(toaster_1)$ can be finalized by using this approach on the resulting connection set {[$switch_2$], [$flashlight_1, switch_3$]}. The last observed action on $switch_2$ is $A_{11} = push(ball_3, switch_2)$ and the last observed action on the second subset is $A_9 = light(flashlight_1)$. Hence, the preconditions of action $make\_toast$ include both $push$ and $light$ actions under the following conditions: $facing(flashlight_1, solarpanel_1)$, $plugged(toaster_1, solarpanel_1)$, $has(toaster_1, switch_2)$.

2) Using Spatial Locality of Objects

When a knowledge base presenting connection types is not provided, the only input to the system is the action sequence. Human players are given the visual scene of objects from which interactions should be determined. An automated agent should also extract the relevant location information of objects to reason correctly. Computer vision techniques are useful for this purpose. The approach that we consider here is using 2D template matching techniques [13] followed by fitting Minimum Bounding Rectangles (*MBR*) for each object to capture spatial information [14]. We assume that location information for each object is provided including its horizontal and vertical dimensions in pixels and the coordinate of the left-top corner. By using this information, two kinds of predicates are extracted: $near$ and has. As shown in Fig. 4, this is done by reasoning as,

$$distance(rec_i, rec_j) < \varepsilon \Rightarrow near(object_j, object_i)$$

$$rec_i \subset rec_j \Rightarrow has(object_j, object_i)$$

where rec_i and rec_j are used for the minimum bounding rectangles of $object_i$ and $object_j$ respectively.

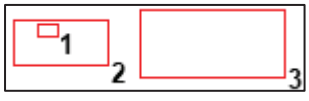


Figure 4. MBRs for $switch_3$ (1), $flashlight_1$ (2) and $solarpanel_1$ (3). The following conclusions can be made for a threshold $\varepsilon > 15$ pixels: has($flashlight_1$, switch$_3$) and $near(flashlight_1, solarpanel_1)$.

Using spatial locality information is useful because some connections could be extracted. However, using just this information for reasoning has the following drawbacks:

- False positives due to close proximity of unrelated objects. For example, $near(motor_1, motor_2)$ relation causes $start(motor_1)$ appear as a precondition for $start(motor_2)$ which is irrelevant.
- Some relationships cannot be extracted as they do not require the related objects to be close to each other (e.g., belt, rope, remote controlled dynamite).
- This approach does not consider the direction of objects (e.g., a flashlight only affects a solar panel if it is directed at that solar panel). Advanced vision techniques are needed to overcome this problem.

3) Using Temporal Locality of Actions

As spatial reasoning does not cover all type of relations and has some drawbacks, we also analyze the results using temporal locality. In this analysis, there is no knowledge base representing relations and spatial information. Instead, timing (start time) of each action $A_i \in \mathcal{A}$ is included in the input action sequence $\mathcal{A}$. If an action A_i occurs after another action A_j in a small interval, we assume that there is a relation between the objects $O_{i,1:m_i}$ and $O_{j,1:m_j}$. This is similar to *meets* relation in Allen's Interval Algebra [15]. Actions $A_1, A_2, A_4, A_6, A_{12}, A_{13}, A_{15}$ and A_{16} in our running example, form a chain reaction as illustrated in Fig. 5.

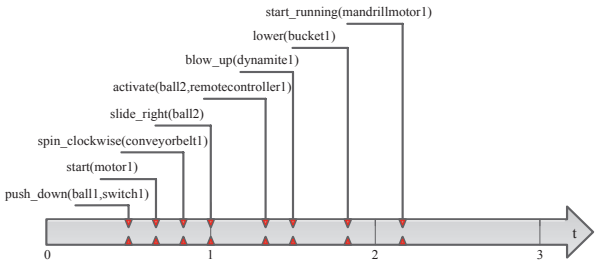


Figure 5: Start time of actions in a chain reaction. An example related action pair: $activate(ball_2, remotecontroller_1)$ *meets* $blow\_up(dynamite_1)$

Temporal analysis can find several useful relationships (e.g., belt and rope) for remote objects when there is a temporal relation. Particularly, remote controlled objects (e.g., $remotecontroller_1$ and $dynamite_1$) are also captured. However, this approach suffers from the following problems:

- False positives due to concurrent independent chain reactions.
- Choosing the related parameter among all parameters for an action. For example, when the relationship between actions $push\_down$ and $start$ is considered, there is no clue on whether the ball or the switch is responsible to activate $start(motor_1)$.
- Objects that are activated by more than one event cannot be modeled (e.g., $toaster_1$ runs when both electricity is provided and $switch_2$ is pushed.).

4) Spatio-Temporal Reasoning

Integration of spatial and temporal analysis takes advantages of both approaches. Usually temporal analysis gives better results, although for certain cases using spatial information is superior. These cases are listed below:

- If any action satisfying *meets* pattern in Allen's Interval Algebra has more than one argument (e.g., $push\_down(ball_1, switch_1)$) and spatial analysis generates a connection (e.g., $switch_1$) including one of these arguments.
- If more than one spatial connection is detected for an object (e.g., $toaster_1$).

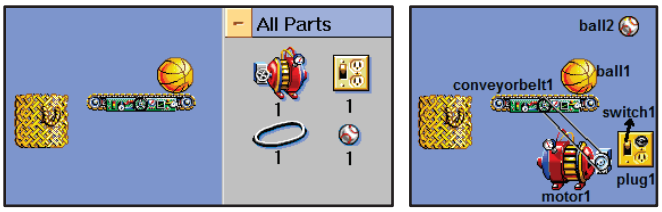
This procedure gives similar results as using knowledge base for connections. However, concurrent independent chain reactions cannot be captured by using the integrated reasoner.

V. Generating a Plan Using Learned Interactions

To show the applicability of the learning results for planning, we use the example scenario illustrated in Fig. 6(a). In this example, the aim is positioning and connecting given objects in an appropriate way to make a chain reaction which will cause $ball_1$ to fall into the basket. Our system can devise a plan to solve this puzzle when the goal is given as $slide\_left(ball_1)$. The chain of possible actions involving given objects (parts) and necessary relations are shown in the resulting plan given in Fig. 7. These relations must be satisfied as in Fig. 6(b) in order to reach the goal state. Here, $related$ is a general relation between two objects. Depending on the input type (knowledge base, temporal/spatial information), it could be represented as a corresponding relation (e.g., has, $plugged$, etc.).

VI. Conclusion

We have presented how interactions among objects are learned by using a given sequence of actions without semantic information on objects, tools and machines. These interactions are later to be used by a planner for solving new puzzles in the Incredible Machine domain. Our analysis shows that when a knowledge base about relations is provided, the interactions to devise new plans are learned. Our analysis also indicates that spatial and temporal analysis has some drawbacks when applied separately. An integrated approach outperforms these two approaches. We have shown that the integrated approach gives similar results as the knowledge-based approach when simultaneous chain reactions are not involved in the tutorials. This is promising because the spatio-temporal analysis does not require great amount of knowledge on relations. Furthermore, the approach could be applicable for the original version of the game where only a visual input is provided to the user. Currently, inputs are provided in a text-based action format. Integrating computer vision techniques is left as a future work.



(a) (b)

Figure 6. (a) A sample puzzle (b) The solution of the puzzle

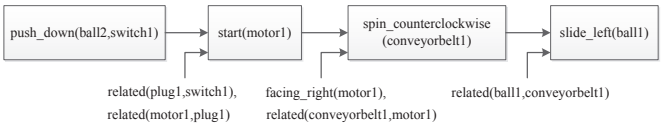


Figure 7. The plan for the given puzzle in Fig. 6 by using the learned interactions

References

[1] *The Incredible Machine 3*, [CD-ROM], Dynamix Inc., 1995.

[2] The official Rube Goldberg website. [Online]. Available: www.rubegoldberg.com. [Accessed: Jan. 12, 2012].

[3] Y. Gil, "Learning by experimentation: Incremental refinement of incomplete planning domains", in *Proc. 11th Int. Conf. on Machine Learning (ICML'94)*, Jul. 1994, pp. 87-95.

[4] D. Shahaf and E. Amir, "Learning partially observable action schemas", in *Proc. 21st Conf. on Artificial Intelligence (AAAI'06)*, Jul. 2006, pp. 913-919.

[5] D. Shahaf, A. Chang, and E. Amir, "Learning partially observable action models: Efficient algorithms", in *Proc. 21st Conf. on Artificial Intelligence (AAAI'06)*, Jul. 2006, pp. 920-926.

[6] E. Amir and A. Chang, "Learning partially observable deterministic action models", *Journal of Artificial Intelligence Research*, vol. 33, pp. 349-402, Nov. 2008.

[7] Q. Yang, K. Wu, and Y. Jiang, "Learning actions models from plan examples with incomplete knowledge", in *Proc. 15th Int. Conf. on Automated Planning and Scheduling (ICAPS'05)*, Jun. 2005, pp.241-250.

[8] K. Wu, Q. Yang, and Y. Jiang, "ARMS: an automatic knowledge engineering tool for learning action models for AI planning", *Knowledge Engineering Review*, vol. 22, pp.135-152, Jun. 2007.

[9] H. H. Zhuo, Q. Yang, D. H. Hu, and L. Li, "Learning complex action models with quantifiers and logical implications", *Artificial Intelligence*, vol. 174, pp.1540-1569, Dec. 2010.

[10] S. Cresswell, T. L. McCluskey, and M. M. West, "Acquisition of object-centred domain models from planning examples", in *Proc. 19th Int. Conf. on Automated Planning and Scheduling (ICAPS'09)*, Sep. 2009, pp. 338-341.

[11] S. Cresswell, T. L. McCluskey, and M. M. West, "Acquiring planning domain models using LOCM", *Knowledge Engineering Review*, 2010, in press.

[12] S. Cresswell and P. Gregory, "Generalised domain model acquisition from action traces", in *Proc. 21st Int. Conf. on Automated Planning and Scheduling (ICAPS'11)*, Jun. 2011, pp. 42-49.

[13] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley, 2009.

[14] D. Chaudhuri and A. Samal, "A simple method for fitting of bounding rectangle to closed regions", *Pattern Recognition*, vol. 40, pp. 1981-1989, Jul. 2007.

[15] J. F. Allen, "Maintaining knowledge about temporal intervals", *Commun. ACM (CACM)*, vol. 26, pp. 832-843, Nov. 1983.