

# Learning Qualitative Action Models in Complex Virtual Worlds

## Submission to ACDC'09

**Author:** Adam Clarke  
**Affiliation:** School of Mathematics, Statistics and Computer Science,  
Victoria University of Wellington  
**Address:** 90A Glenmore St, Wellington, New Zealand  
**E-mail:** adam77@gmail.com  
**Phone:** 021-133-0853  
**Enrolment Date:** March 2007  
**Keywords:** learning in worlds with objects, qualitative reasoning, action models  
**Key References:** (Zettlemoyer et al. 2005, Coghill et al. 2002, Forbus 1984)

### 1 Research Problem

A long standing goal of artificial intelligence research is to create autonomous agents that are able to learn rules describing how their environment works (e.g. the effects of actions), and not have to rely on inflexible 'hard-coded' knowledge. This project aims to create an agent that learns such rules in real time by observing and experimenting. The agent is situated in a complex virtual world that provides a rich simulation of real world physics, processes, other agents, and an unlimited variety of objects. The agent is able to interact with its environment by executing actions and making observations, it 'sees' the world in terms of objects, properties and relationships. The agent will learn symbolic qualitative models that describe how objects behave over time and the effects of actions. This differs from the usual approach of treating actions and behaviour as separate learning problems. The simulation is designed to be as realistic as possible by incorporating many features that make learning difficult. For example, the direct effects of a particular action may not be visible to the agent.

The goal of this research is to answer the question of how an agent can simultaneously learn models of action effects and qualitative behaviour in a complex environment. The problem is challenging because realistic environments contain a variety of features that each lead to non-trivial learning problems. These include:

- There are an unlimited number of objects
- Other agents operate in the world
- There are many interacting complex processes
- Properties can be continuous or discrete
- Actions can have multiple effects
- Actions can have different effects in different situations
- Action effects can be stochastic, unobservable and/or delayed
- Observations are noisy

The aim is not to solve all these problems by creating an agent that learns perfect models; rather, the aim is an agent that learns models which are sufficient to plan future actions. An inaccurate model may cause the agent to replan or abandon a goal, this is acceptable.

The problem addressed here (of learning actions and behaviour simultaneously in an unconstrained

environment), is more general than that attempted in previous work, however, a simplified approach is taken by learning a state machine based model, rather than the more sophisticated action and behaviour representations normally used. Also, my system partitions the world into simple mechanisms and learns each individually rather than trying to extract general rules that apply in every situation. The remainder of this paper explains these ideas in greater detail.

### Related Work

This work builds on learning methods used in two existing areas of research: algorithms that learn the effects of actions, and algorithms that learn qualitative rules. A body of work exists in both these areas but there is no work that addresses the problem of simultaneously learning both action effects and qualitative behaviour.

Frameworks for reasoning about systems qualitatively were first formalised in the 1980's (see Forbus (1984)). Since then, a number of algorithms have been proposed for extracting qualitative constraints from sets of example behaviours (Coiera 1989, Richards et al. 1992). These algorithms work in highly constrained environments and assume only a single process. More recently, powerful learning techniques have been adapted to qualitative learning, including inductive logic programming (Coghill et al. 2002, Srinivasan & King 2008), Bayesian networks (Boxer 2001), and induced decision trees (Bratko & Šuc 2004). However, these systems still require relatively constrained environments and do not allow for the effects of actions.

An early formalisation of action models (including their pre-conditions and effects) was a planning system called STRIPS (Fikes & Nilsson 1971). A variety of methods have since been developed for learning the effects of actions, all of which use a STRIPS-like representation. The focus of this area has been to learn actions in increasingly difficult environments. For example, Amir (2005) proposes a technique for dealing with unobservable action effects. However, there are none that work in a truly complex environment: they rely on one or more aspects of the environment being controlled and therefore eliminated as a learning difficulty. For example, the system described by Wang (1994) requires there to be no changes to the environment between actions, which eliminates all processes with ongoing behaviours.

The work most closely aligned to the goals of this project is an agent that learns probabilistic rules by

observing a virtual environment (Zettlemoyer et al. 2005). It also addresses the same problem of modelling a 3-dimensional world with realistic physics, in particular the actions involved in building a tower of blocks. The key difference is that it need only learn the immediate effects of actions; it does not model how objects behave once the action has completed (e.g. a wobbly tower eventually falling over).

## 2 System Design/Methodology

The learning system consists of four parts:

- A 3-dimensional simulated virtual environment
- A mechanism for interacting with the environment
- A representation for modelling the behaviour of the environment
- An algorithm for learning models from observations

*The simulation:* The agent’s world is built using a free off-the-shelf game engine ([www.blender.org](http://www.blender.org)) with a real time rigid body physics engine ([www.bulletphysics.com](http://www.bulletphysics.com)). The flexibility of the game engine means that any number of everyday scenarios can be easily constructed. The learning difficulty can also be easily controlled by introducing features such as other agents, clumsy actuators, etc. The agent’s current world is a kitchen containing a multitude of everyday objects including a sink, oven and dirty dishes. One limitation of the game engine is that liquids are not simulated and therefore collections of small spheres are used to approximate qualitative liquid behaviour. Figure 1 shows a typical kitchen environment built in Blender for the agent to operate in.



Figure 1: Interactive 3D Kitchen Environment.

*Interaction:* An interaction module built on top of Blender allows the agent to act on and observe objects. The agent receives a regular series of snapshots (at least one every second) containing all objects, properties and relationships in the scene (unless they are marked as unobservable). For example, a cup object may have a blue colour property and be related to a plate as ‘touching’. Relationships and properties requiring measurement are observed qualitatively — the agent cannot observe exact quantities. For example, the speed of a falling ball might be perceived as the following triple  $\langle 0 \dots \infty, inc, + \rangle$ , indicating that the current speed is greater than 0 ( $0 \dots \infty$ ), the speed is currently increasing ( $inc$ ), and has increased since the last snap-shot ( $+$ ). Actions are implemented by

allowing the agent to attempt any pre-defined action (push, move, drop, etc) on any object.

*The model:* The agent uses a representation that supports partitioning the world into ‘systems’ of objects with a recognisable behaviour. Each of these systems contains a set of objects, a context (specifying a set of property values that must hold true for the objects to constitute the system), a set of states (each specifying a set of properties that are true for this state), and finally, a set of transitions between states (specifying actions or events that move the system between states). Two categories of transitions are used to model the qualitative behaviour of the system: action transitions and ‘time passes’ transitions. Action transitions represent the immediate effect of an action; time passing transitions represent qualitative state changes in the absence of an action.

There are a number of advantages to this approach. Firstly, actions and qualitative behaviour can be learned together rather than employing separate algorithms for each. Secondly, by focusing on a single system the agent can learn a small subset of the behaviour of its environment: extraneous events involving objects not considered part of a system can be ignored during the learning process. Thirdly, complex systems can be broken down into easier to learn subsystems and then combined to describe the behaviour of the system as a whole.

### Learning Algorithm

The purpose of the learning algorithm is to construct a qualitative system from real-time observations of a small number of examples. It will learn from the results of the agent’s own actions and those of a teacher. The algorithm proceeds by incrementally building the system’s context, states and transitions. The algorithm detects qualitative changes in the environment and uses them as evidence to enhance the model. Learning terminates when the model has stabilised (i.e. no new states are being added); alternatively the teacher can signal that learning should cease.

A key component of the algorithm is the heuristic used to determine if a change in the environment is relevant to the current system or should be ignored. Irrelevant observations can arise from several sources including the actions of other agents, background processes and noisy sensors. The current implementation makes this distinction by focusing only on objects with properties that change immediately after the agent (or teacher) performs an action. This approach fails to identify correct differences when the agent performs an action and an irrelevant change co-incidently occurs immediately after the action. (A more robust heuristic is in development that takes into account a number of factors that indicate an observation is more likely to be relevant. For example, an object is considered more likely to be relevant if it is physically connected to or touching an object that is already a part of the system.)

Figure 2 shows the pseudocode for the learning algorithm. The rest of this section describes the key steps in detail...

*Step 1, initialisation.* The three core components of the system are: the context that specifies which objects are currently considered relevant to the system, the qualitative states (which are specified by assigning values to properties of objects in the context), and finally the transitions between states. These all begin as empty sets.

*Step 2,* the next snapshot of all objects in the current scene is obtained by making a request to the vision system. The response is a set of object observations and a set of action observations (if any).

### LearnQModel

Let  $\mathbf{C}$  be the set of objects in the system context  
 Let  $\langle \mathbf{S}, \mathbf{T} \rangle$  be a directed graph of qualitative states  $\mathbf{S}$  and transitions  $\mathbf{T}$   
 Let  $\mathbf{s}_{current} \in \mathbf{S}$  be the current state of the system

1) Initialise  $\mathbf{C}, \mathbf{S}, \mathbf{T}$  to  $\emptyset$

Repeat:

2) Get next snapshot of observations

3) Update context with any new objects...

If an action occurred:

Find the set of objects  $\mathbf{O}$  that have changed

Update the context,  $\mathbf{C} = \mathbf{C} \cup \mathbf{O}$

For each state  $\mathbf{s} \in \mathbf{S}$ :

Add properties of objects in  $\mathbf{O}$  to  $\mathbf{s}$

4) Determine new state...

$\mathbf{s}_{new}$  = the set of all properties of objects in  $\mathbf{C}$

Set all properties in  $\mathbf{s}_{new}$  to current observations

5) Add new state and transition...

If  $\mathbf{s}_{new}$  and  $\mathbf{s}_{current}$  are different:

If  $\mathbf{s}_{new} \notin \mathbf{S}$ :

Add state,  $\mathbf{S} = \mathbf{S} \cup \{\mathbf{s}_{new}\}$

Find transition  $\mathbf{t}_{new}$  from  $\mathbf{s}_{new}$  and  $\mathbf{s}_{current}$

If  $\mathbf{t}_{new} \notin \mathbf{T}$ :

Add transition,  $\mathbf{T} = \mathbf{T} \cup \{\mathbf{t}_{new}\}$

Else:

Increment  $\mathbf{t}_{new}.count$

Update current state,  $\mathbf{s}_{current} = \mathbf{s}_{new}$

6) Stop if system is complete

Figure 2: *LearnQModel* Pseudocode.

*Step 3*, update the context. If an action occurred then we retrieve the set of objects whose properties have changed since the last snapshot. These objects are assumed to have changed as a result of the action and should therefore be included in the system. Therefore, any of these objects that are not already in the context are added to it. All existing states are updated with the properties of the new objects. The value of the each property is set to ‘don’t know’ because the objects were ignored up to this point.

*Step 4*, determine the new state. The new system state is specified using the properties of all objects in the context. Each property is set to its currently observed value.

*Step 5*, add new state and transition. The new state is compared with the state of the previous snapshot. If they are different, then a state change has taken place and a new state and/or transition may be added. If the new state is not one of the system’s existing states (i.e. the state has not been seen before) then it gets added. Similarly, a new transition is added if the action (or lack of action) has not previously caused the current state change.

If the transition is not new then a counter is incremented (any transitions observed only once can be considered anomalous). Finally, the new state is set as the current system state.

*Step 6*, a check is made to see if the system is complete and learning can finish. The system is considered complete if no updates have been made for  $X$  number of minutes, where  $X$  is a parameter of the algorithm. An alternative is to have the teacher manually stop the learning. (The requirement that an observable change must happen every  $X$  minutes prevents the agent from learning about processes that take a long time. This is currently an unresolved is-

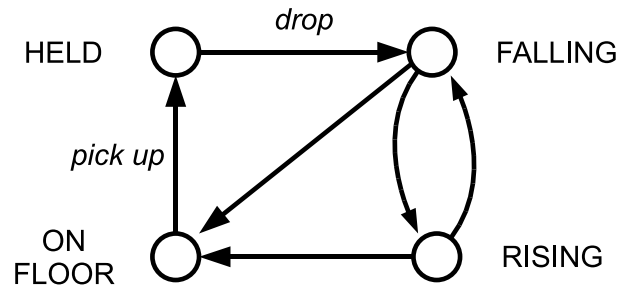


Figure 3: The State Model Learned in the Bouncing Ball Scenario. Unlabelled transitions are ‘time passes’ events where no action occurred.

sue.)

### 3 Preliminary Results

Early results indicate that the outlined approach is feasible. The algorithm has been used to successfully learn a number of simple systems within a constrained environment. In these scenarios the agent observes a teacher performing actions relevant to the system being learned.

The first scenario concerns a tennis ball and the effects of gravity. A ball is dropped by the teacher and then bounces several times before coming to rest. The agent observes the ball in terms of its qualitative speed, acceleration and direction (direction is observed relative to gravitational pull, i.e. ‘up’ or ‘down’); the agent also observes the qualitative distance of the ball to the ground. The resulting model (see figure 3) shows the expected behaviour of the ball. The ball can be in one of four states: held above the ground, or falling (and accelerating), or rising (and decelerating), or on the floor. Dropping the ball causes it start falling; over time, the ball moves between the falling and rising states; eventually the ball will be on the ground.

The transitions of the model are as expected with the exception of the transition *rising*  $\rightarrow$  *on floor*. It is present because the time interval between snapshots is not fine-grained enough. The vision system missed the *falling* state causing the agent to believe that the ball moved directly from *rising* to *on floor*. This issue is also the reason why the model does not have a ‘suspended in mid-air’ state between the *rising* and *falling* states. How to deal with the agent missing these kinds of transitions is a current issue; one idea is to exploit the fact that a smoothly changing continuous variable (e.g. the speed of the ball) must transition through its qualitative quantity space without skipping any values (i.e. if the ball had positive speed and now has negative speed it must have been stopped at some point).

The second scenario is more complex because it involves several objects — a sink, some water, a tap and a plug — and various different actions — turning the tap clockwise, turning the tap counter-clockwise, pushing the plug button. A further complication is that the state of the plug (‘opened’ or ‘closed’) is hidden. The agent must learn how the water level in the sink behaves. Figure 4 shows the model after just a few actions. The agent has learned that pushing the tap will cause an empty sink to start filling; pulling the tap will cause the sink to stop filling; opening the plug will cause the sink to start emptying and will eventually be completely empty. The model is accurate with the exception that it doesn’t distinguish between states where the plug is either open or closed. A

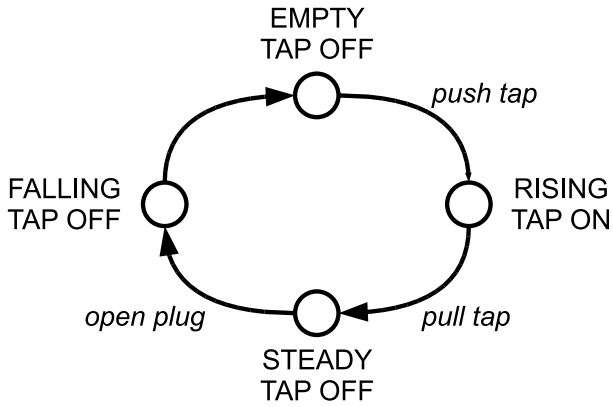


Figure 4: The Tap and Sink Model After a Few Actions.

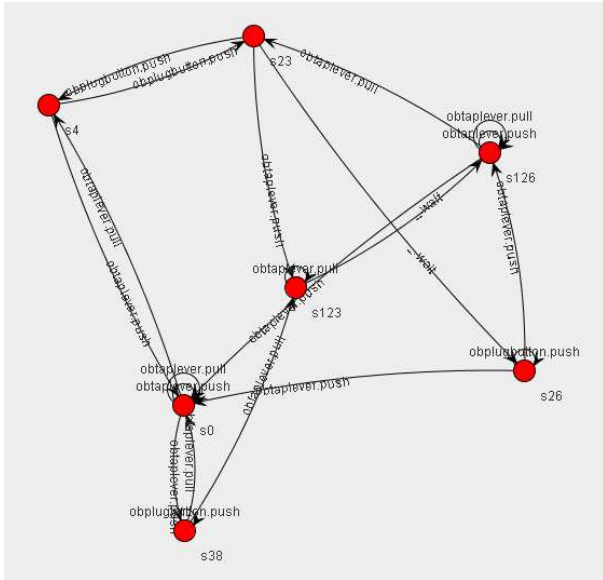


Figure 5: A Screenshot Illustrating the Complexity of the Complete Tap and Sink Model. There are 7 states and 22 transitions.

mechanism is required for detecting hidden variables.

The complexity of the complete tap model is illustrated by the screenshot in figure 5. It shows the system after all of its different behaviours have been explored by many actions. This is an accurate model but the combination of inflow and outflow processes has meant that a large number of transitions, 22 in total, were learned. A more useful pair of models may result from learning the inflow and outflow processes as separate systems.

The next phase of work focuses on the enhancements already mentioned: a better focus heuristic, dealing with missing states, and hidden variables. Subsequent goals include creating a mechanism for breaking complex systems into sub-systems (and recombining them), and introducing more challenging scenarios.

#### 4 Expert Advice Sought

I seek advice in two areas. Firstly, how to determine the appropriate scope of research in an area that encompasses several hard sub-problems, and secondly, how best to evaluate my results given that there is no existing work addressing precisely the same problem.

The problem of scope applies to this project because it investigates a general problem (how to learn

common sense models in complex environments) that requires solutions to several difficult sub-problems, such as, how to deal with noisy observations, probabilistic outcomes, hidden states, etc. One approach would be to fully explore a single sub-problem. This would involve creating an agent that learns perfect (or at least very accurate) models in a constrained environment. Work would then proceed by incrementally increasing the complexity of the environment. Alternatively, it may be more fruitful to use the most complex environment from the outset, and then proceed by building and improving a more general (but less accurate) algorithm.

Advice is also sought on how to evaluate results. There is no existing work addressing the problem of simultaneously learning actions and qualitative behaviour. The only option for comparative analysis is to compare against the existing work that focuses on learning either actions or behaviour as isolated problems. What might the pitfalls of this be? A second evaluation technique is to assess the quality of the models learned in specific scenarios against those created by an expert (i.e. a human). What considerations should be taken into account when choosing these scenarios?

#### References

- Amir, E. (2005), Learning partially observable action models, *in* '19th International Joint Conference on Artificial Intelligence'.
- Boxer, P. A. (2001), Towards learning naive physics by visual observation: Qualitative spatial representations, *in* 'AI '01: Proceedings of the 14th Australian Joint Conference on Artificial Intelligence', Springer-Verlag, London, UK, pp. 62–70.
- Bratko, I. & Šuc, D. (2004), 'Learning qualitative models', *AI Magazine* **24**(4), 107–119.
- Coghill, G. M., Garrett, S. M. & King, R. D. (2002), Learning qualitative models in the presence of noise, *in* 'QR'02 Workshop on Qualitative Reasoning'.
- Coiera, E. (1989), Learning qualitative models from example behaviors, *in* 'Proceedings of the Third Qualitative Physics Workshop', Stanford University, Palo Alto, California.
- Fikes, R. & Nilsson, N. (1971), 'Strips: A new approach to the application of theorem proving to problem solving', *Artificial Intelligence* **2**, 189–208.
- Forbus, K. D. (1984), 'Qualitative process theory', *Artificial Intelligence* **24**, 85–168.
- Richards, B. L., Kraan, I. & Kuipers, B. J. (1992), Automatic abduction of qualitative models, *in* 'Proceedings of the Fifth International Workshop on Qualitative Reasoning about Physical Systems', pp. 723–728.
- Srinivasan, A. & King, R. D. (2008), 'Incremental identification of qualitative models of biological systems using inductive logic programming', *Journal of Machine Learning Research* **9**, 1475–1533.
- Wang, X. (1994), Learning planning operators by observation and practice, *in* 'Artificial Intelligence Planning Systems', pp. 335–340.
- Zettlemoyer, L. S., Pasula, H. & Kaelbling, L. P. (2005), Learning planning rules in noisy stochastic worlds, *in* 'National Conference on Artificial Intelligence (AAAI)', pp. 911–918.