DEPARTMENT OF COMPUTING

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

H-PERL: The Hybrid Property, Event and Relation Learner

Author:
Ross Irwin

Supervisors:
Prof. Alessandra Russo
Dr. Krysia Broda
Dr. Jorge Lobo
Mr. David Tuckey

June 14, 2020

Abstract

Writing algorithms which can answer questions about a video - a task known as VideoQA - has long been a goal of artificial intelligence (AI) researchers. In order to correctly answer these questions, such algorithms are required to reason about physical and spatial properties of objects, as well as temporal sequences of frames.

Most existing VideoQA implementations make use of large neural networks, which require significant time and resources to train. These models also suffer from opacity; it is very difficult to understand what these models learn and how they come to their conclusions. Hybrid models - models which make use of both neural networks and symbolic reasoning - offer an alternative approach. Implemented correctly, hybrid models have the ability to recognise complex patterns in the video, as well as the ability to store knowledge, and reason about this knowledge, in an interpretable manner.

In this report, we present a novel, algorithmically generated VideoQA dataset, which models objects, relations and interactions between objects. The dataset contains seven different types of questions, each of which tests a model's intelligence in a particular way.

We also present a hybrid architecture for VideoQA tasks. This architecture combines the strengths of neural networks and symbolic reasoning, in order to understand: properties of objects, relations between objects and the occurrence of events in a video. We train a number of different models of this architecture on our VideoQA dataset, and evaluate each model's relative strengths and weaknesses. We show that one of these models can reason about uncertainty caused by errors and is able to answer over 99% of questions in the testing data correctly. Finally, as part of our qualitative comparisons to existing implementations, we show that one of our models would be more adaptable to new environments than another recently proposed hybrid VideoQA model.

Acknowledgements

I am very grateful for the advice, support and patience of all four of my supervisors: Prof. Alessandra Russo, Dr. Krysia Broda, Dr. Jorge Lobo and Mr. David Tuckey. Thank you for investing so much time and effort in this project.

I would also like to say thanks to my family for their continued support. I would also particularly like to say thanks to my parents for teaching me the importance of education.

Contents

List of Figures

List of Tables

Bibliography

[1] Michael Gelfond and Yulia Kahl. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press, 2014.

Appendix A

ASP encoding of $\mathcal{AL}$

In this appendix we present a method for encoding an $\mathcal{AL}$ system description in ASP, as described in [1]. We need to encode three different parts of the system description: the signature, the $\mathcal{AL}$ statements and some domain-independent axioms.

We encode the signature of the system description, $sig(SD)$, as follows:

- For each constant symbol c of sort $sort\_name$ other than $fluent$, $static$ or $action$, $sig(SD)$ contains

$$sort\_name(c). \tag{A.1}$$

- For every static g of SD, $sig(SD)$ contains

$$static(g). \tag{A.2}$$

- For every intertial fluent f of SD, $sig(SD)$ contains

$$fluent(inertial, f). \tag{A.3}$$

- For every defined fluent f of SD, $sig(SD)$ contains

$$fluent(defined, f). \tag{A.4}$$

- For every action a of SD, $sig(SD)$ contains

$$action(a). \tag{A.5}$$

In the following we refer to the ASP encoding of the $\mathcal{AL}$ system description as $\Pi(SD)$, where $\Pi(SD)$ includes $sig(SD)$. We introduce a relation $holds(f,i)$ which says that fluent f is true at timepoint i. We also introduce the notation $h(l,i)$ where l is a domain literal and i is a step, which will not be used in the ASP program, but will instead be replaced by either $holds(f,i)$ if $l = f$, or by $\neg holds(f,i)$ if $l = \neg f$.

We encode the $\mathcal{AL}$ statements as follows:

- If the maximum number of steps is $< max >$, then $\Pi(SD)$ includes

$$\#const\ n = \ < max > . \tag{A.6}$$
$$step(0..n). \tag{A.7}$$

- For every causal law, a **causes** l **if** $p_0, ..., p_m$ $\Pi(SD)$ contains

$$h(l, I+1) \text{ :- } h(p_0, I), ..., h(p_m, I), occurs\_action(a, I), I < n. \tag{A.8}$$

- For every state constraint, l **if** $p_0, ..., p_m$ $\Pi(SD)$ contains

$$h(l, I) \text{ :- } h(p_0, I), ..., h(p_m, I). \tag{A.9}$$

- For every executability condition, **impossible** $a_0, ..., a_k$ **if** $p_0, ..., p_m$ $\Pi(SD)$ contains

$$\neg occurs\_action(a_0, I); ...; \neg occurs\_action(a_k, I) \text{ :- } h(p_0, I), ..., h(p_m, I). \tag{A.10}$$

The ; in the head of rule A.10 stands for logical disjunction and can be read as meaning at least one of $a_0, ..., a_k$ must not occur at timepoint I.

In order to complete our encoding of an $\mathcal{AL}$ system description we need to add a number of domain-independent axioms. These axioms are not specific to any system or task, but rather convey commonsense knowledge that should apply to many systems. It is worth noting, however, that in certain situations some or all of these axioms may not make sense and should not be used.

We encode the domain-independent knowledge as follows:

- The inertia axiom states that inertial fluents will keep their state unless explicitly changed:

$$\begin{aligned} holds(F, I+1) \text{ :- } & fluent(inertial, F), \\ & holds(F, I), \\ & not\ \neg holds(F, I+1), \\ & I < n. \end{aligned} \tag{A.11}$$

$$\begin{aligned} \neg holds(F, I+1) \text{ :- } & fluent(inertial, F), \\ & \neg holds(F, I), \\ & not\ holds(F, I+1), \\ & I < n. \end{aligned} \tag{A.12}$$

- The closed-world assumption (CWA) for defined fluents states that defined fluents which are not known to be true are assumed to be false.

$$\neg holds(F, I) \text{ :- } fluent(defined, F), step(I), not\ holds(F, I). \tag{A.13}$$

- The CWA for actions states that actions that are not known to occur are assumed to not occur.

$$\neg occurs\_action(A, I) \text{ :- } action(A), step(I), not\ occurs\_action(A, I). \tag{A.14}$$

Finally, in order to make use of the system description encoding we would include information on which events occured, using the $occurs\_action(A, I)$ predicate, and information on the different states of the system using $holds(F, I)$ and $\neg holds(F, I)$.

Briefcase Example

We now consider a simple example domain: a briefcase with two clasps (from [1]). There is a single action, toggle, which moves a given clasp into the up position if it is down, and vice-versa. If both clasp are up the briefcase is open, otherwise, it is closed.

The signature of the domain consists of sort clasp, which can take value 1 or 2, inertial fluent $up(C)$, defined fluent $open$ and action $toggle(C)$, where C is one of the clasps.

A schema for the system description is as follows:

$$toggle(C) \textbf{ causes } up(C) \textbf{ if } \neg up(C) \tag{A.15}$$
$$toggle(C) \textbf{ causes } \neg up(C) \textbf{ if } up(C) \tag{A.16}$$
$$open \textbf{ if } up(1), up(2) \tag{A.17}$$

This is known as a schema for a system description since it contains variables. Individual rules can be obtained by grounding the variables. So each of the first two schema rules would produce two ground rules, one where $C = 1$ and another where $C = 2$.

Following the procedure outlined above, we would encode this system description into the following ASP program:

```
% Possible timesteps
#const n = 1.
step(0..n).

% Signature
clasp(1).
clasp(2).

fluent(inertial, up(C)) :- clasp(C).
fluent(defined, open).
action(toggle(C)) :- clasp(C).
```

```
% Domain dependent rules
holds(up(C), I+1) :- occurs_action(toggle(C), I),
                     -holds(up(C), I),
                     I<n.

-holds(up(C), I+1) :- occurs_action(toggle(C), I),
                      holds(up(C), I),
                      I<n.

holds(open, I) :- holds(up(C), I), holds(up(2), I).

% Domain independent rules
holds(F, I+1) :- fluent(inertial, F),
                 holds(F, I),
                 not -holds(F, I+1),
                 I<n.

-holds(F, I+1) :- fluent(inertial, F),
                  -holds(F, I),
                  not holds(F, I+1),
                  I<n.

-holds(F, I) :- fluent(defined, F), step(I),
                not holds(F, I).

-occurs_action(A, I) :- action(A), step(I),
                        not occurs_action(A, I).
```

In a similar fashion to system description schemas, ASP programs allow variables. ASP solvers will first ground the program by computing the the *relevant grounding* - the set of ground rules which could be used by the program. The relevant grounding can be infinite so care needs to taken to ensure that it can be computed.

Appendix B

OceanQA Examples

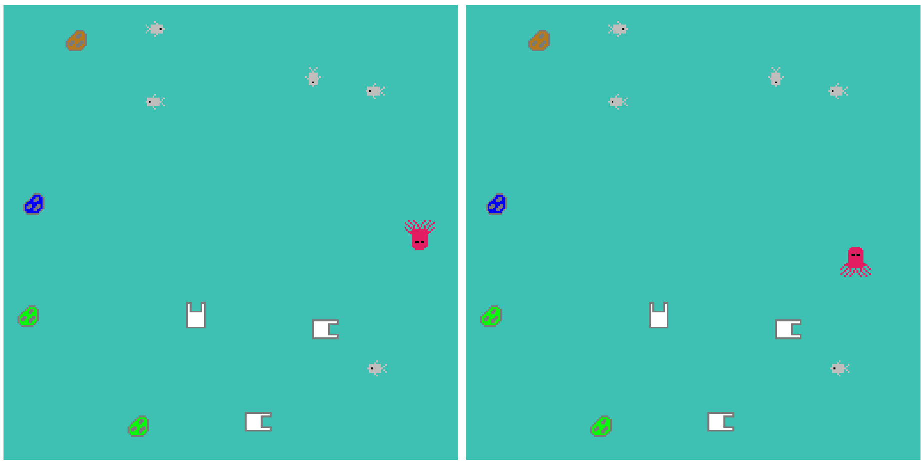
This supplementary material provides a number of example questions and video frames from the OceanQA dataset. The appendix is split into sections based on the question type. Each section outlines a number of example questions and some of the video frames which relate to each question.

Property Questions

Q: What rotation was the brown rock in frame 21?
A: upward-facing

Q: What colour was the right-facing fish in frame 27?
A: silver

Figure B.1 shows frames 21 and 27 from these videos.



(a) Frame 21. **(b)** Frame 27.

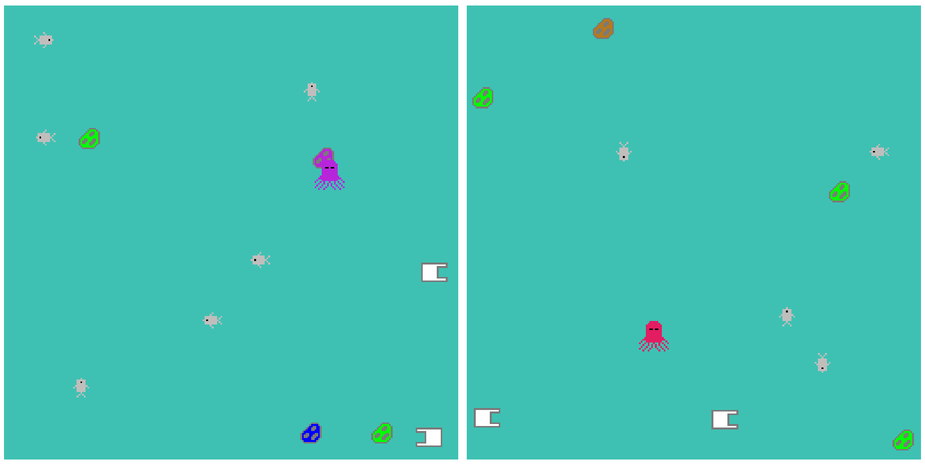
Figure B.1

Relation Questions

Q: Was the purple rock close to the octopus in frame 7?
A: yes

Q: Was the upward-facing fish close to the brown rock in frame 20?
A: no

Figure B.2 shows the two frames these questions are referring to.



(a) Frame 7. **(b)** Frame 20.

Figure B.2

Action Questions

Q: Which action occurred immediately after frame 12?
A: move

Q: Which action occurred immediately after frame 16?
A: rotate anti-clockwise

Figure B.3 gives an example of an octopus rotating anti-clockwise.

Changed-Property Questions

Q: What happened to the octopus immediately after frame 0?
A: Its rotation changed from upward-facing to left-facing

Figure B.3 shows the first and second frame from this video.

Q: What happened to the octopus immediately after frame 20?
A: Its colour changed from red to blue

Figure B.4 shows frame 20 and frame 21 from this video.

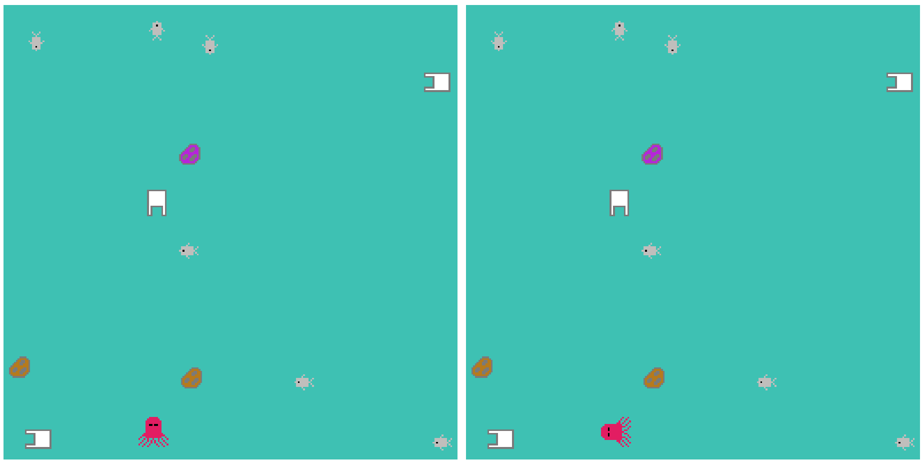


Figure B.3: An example of an octopus rotating clockwise.

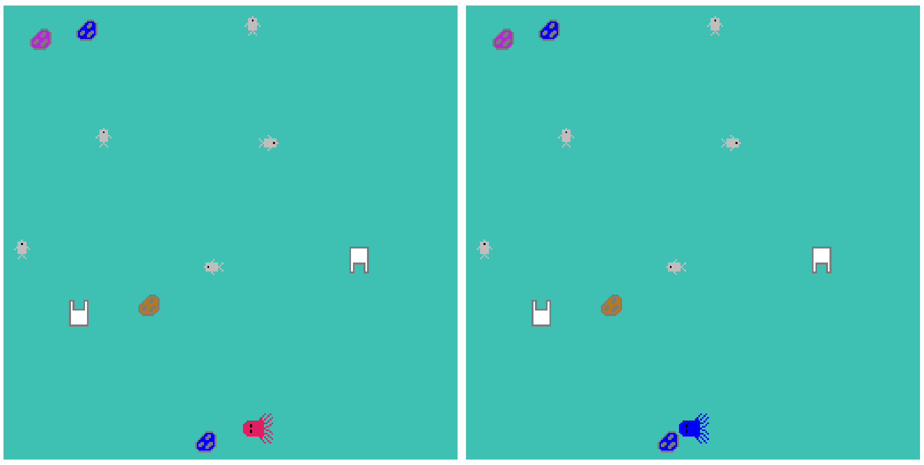


Figure B.4: An example of an octopus changing colour.

Repetition Count Questions

Q: How many times does the octopus rotate clockwise?
A: 0

Figure B.7 gives an example of an octopus rotating clockwise and then moving.

Q: How many times does the octopus eat a bag?
A: 1

Figure B.5 shows the octopus eating a bag.

Repeating Action Questions

Q: What does the octopus do 1 times?
A: eat a fish

Figure B.6 shows an octopus eating a fish.

Q: What does the octopus do 4 times?
A: rotate clockwise

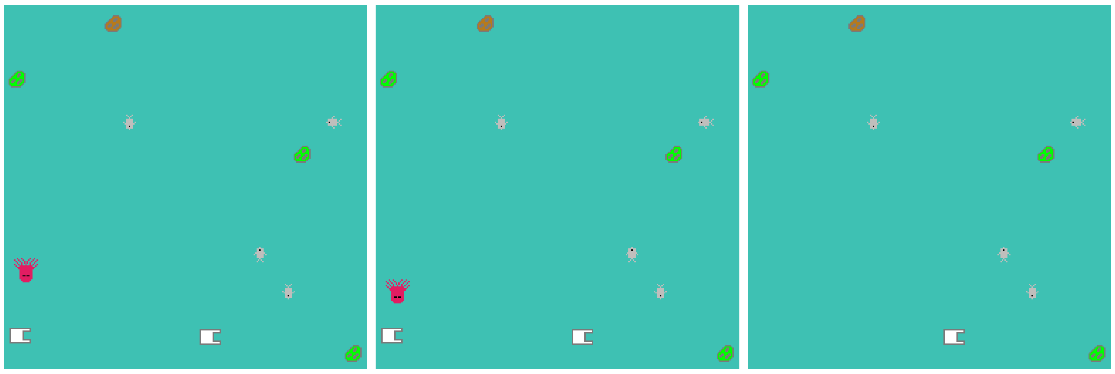


Figure B.5: An example of an octopus eating a bag.

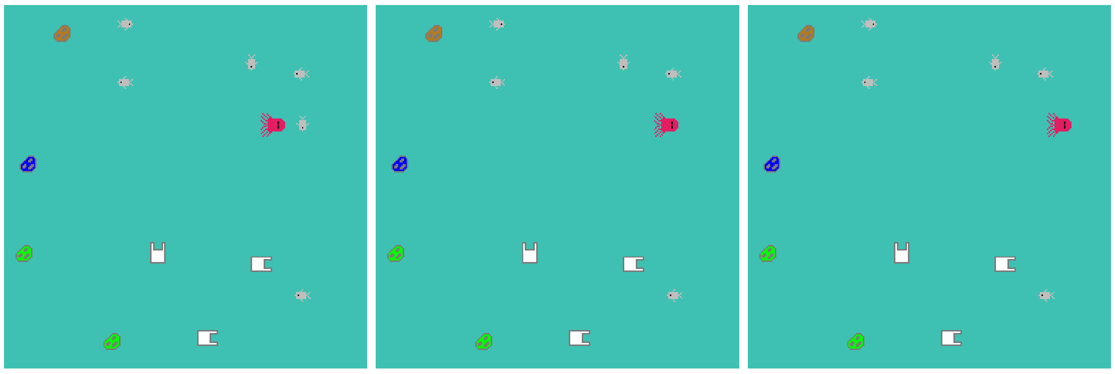


Figure B.6: An example of the octopus eating a fish.

State Transition Questions

Q: What does the octopus do immediately after rotating clockwise for the fourth time?
A: move

Figure B.7 shows the video frames which this question is asking about.

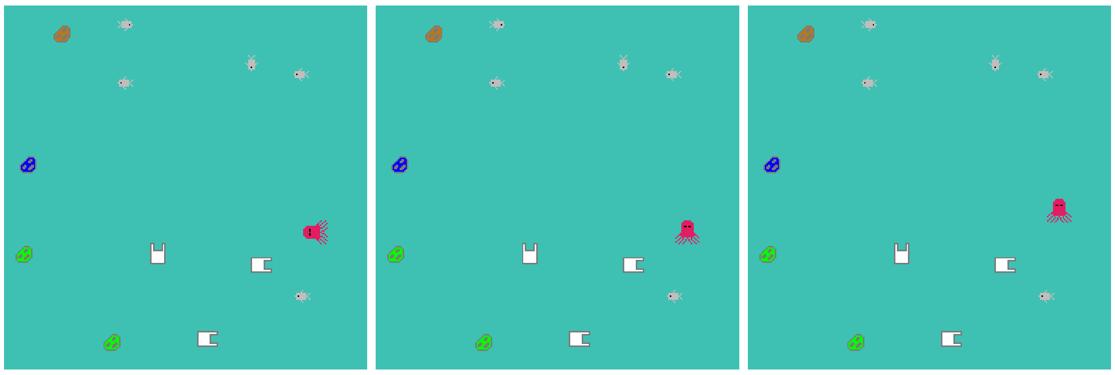


Figure B.7: An example of an octopus rotating clockwise before moving.