Learning Symbolic Action Definitions from Unlabelled Image Pairs[*]

Helen Harman, Pieter Simoens
Department of Information Technology - IDLab
Ghent University - imec
Technologiepark 126, B-9052 Ghent, Belgium
{helen.harman, pieter.simoens}@ugent.be

Abstract

Task planners and goal recognisers often require symbolic models of an agent's behaviour. These models are usually manually developed, which can be a time consuming and error prone process. Therefore, our work transforms unlabelled pairs of images, showing the state before and after an action has been executed, into reusable action definitions. Each action definition consist of a set of parameters, effects and preconditions. To evaluate these action definitions, states were generated and a task planner invoked. Problems with large state spaces were solved using the action definitions learnt from smaller state spaces. On average, the task plans contained 5.46 actions and planning took 0.06 seconds. Moreover, when 20 % of transitions were missing, our approach generated the correct number of objects, action definitions and plans 70 % of the time.

Introduction

Symbolic models of an agent's behaviour are required by tasks planners and goal/plan recognisers. To achieve a given goal, task planners (Helmert 2006; Weber and Bryce 2011) find a sequence of discrete actions. Whereas, plan and goal recognisers (Kautz and Allen 1986; Ramírez and Geffner 2010; Pereira, Pereira, and Meneguzzi 2019) attempt to infer an observed agent's goal and/or plan from a sequence of observations.

Rather than enumerating all possible actions an agent can perform, symbolic action definitions are developed. The signature of an action definition consists of a name and a parameter list, and its body contains preconditions and effects. Action definitions are grounded (i.e., transformed into actions) by providing objects as arguments. The aim of our work is to automatically generate generic (i.e., reusable) action definitions from unlabelled pairs of images, namely, transitions, which show the state before and after an action has been executed.

Symbolic action definitions are usually manually written. This can be a burdensome task and often requires domain specific knowledge (Kambhampati 2007; Weber and Bryce 2011). Therefore, methods that attempt to learn the preconditions and effects of actions have been proposed (Yang, Wu, and Jiang 2007; Aineto, Jiménez, and Onaindia 2018) but, until recently, some symbolic knowledge was still necessary. Transitions were provided to LatPlan (Asai and Fukunaga 2018), which employed a deep autoencoder to generate the action definitions. Nevertheless, to generate symbolic actions (i.e., PDDL), all transitions were required. Moreover, deep-learning approaches often have a lengthy training time, and the decisions made by the algorithms are likely to be unexplainable.

In this paper, objects are discovered by finding pixels that always change value simultaneously; both the location of these pixels and their values are defined as objects. Subsequently, the transitions are transformed into actions by generating predecessor and successor states from the images. These states enable the actions' preconditions and effects to be determined. Actions are then converted into a set of action definitions, which can be provided to task planners and goal/plan recongisers. Our learnt action definitions are evaluated by calling a task planner on a generated initial and goal state. This paper expands on the work of Harman and Simoens (2020), which focused on a single domain and did not involve experimenting with missing transitions.

Seemingly simple problems can contain thousands of transitions. For instance, a 3 by 3 puzzle problem (see Figure 1) contains 362 880 states and 967 680 transitions. Therefore, we do not assume that all transitions are present in the image dataset. As a result, because there is no way to determine if an unseen transition is invalid or just missing, there are likely to be domains our method does not create valid action definitions for. Nevertheless, the decisions made by our method are explainable, and thus to handle

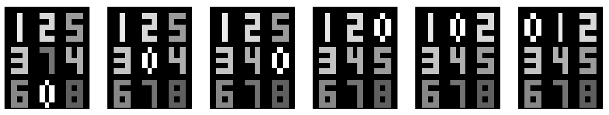


Figure 1: Example plan for solving a Puzzle problem. The initial state is on the left and the goal is on the right. Tiles can only be swapped with the clear tile (i.e., the 0) if they are adjacent to it. All example plans were produced using the action definitions created by our approach. The transitions, which were provided as input, are based on the work of Asai and Fukunaga (2018).

[*]A version of this paper has been accepted at ICAAI.

these cases, alterations can be made to our state and precondition generation algorithms. Like images, many other sources of sensed data simply contain locations with values; thus, the presented processes can be applied to alternative data sources. For simplicity, all objects within an image are assumed to be rectangular, no objects are occluded, and images contain no noise. Preprocessing continuous motion within videos, e.g., discretising it into grid squares (rectangles), is beyond the scope of our paper. Although lifting these assumptions would increase the computational cost of the image processing steps, many of the concepts we introduce would still be applicable.

Problem Formalisation

Task planning and goal recognition problems are often defined in Planning Domain Definition Language (PDDL) (McDermott 2000), a popular domain-independent language for modelling the behaviour of deterministic agents. A PDDL defined problem includes objects, predicates, action definitions, and states. Our approach takes a set of transitions as input and transforms them into PDDL. This section provides a definition for a transition and for the components of a PDDL defined problem. Subsequently, the example domains are introduced.

Definition 1. *Transitions:* A transition ($t \in T$) is a pair of unlabelled images, i.e., a predecessor image (t_{pi}) and a successor image (t_{si}).

Definition 2. *Objects:* There are two types of objects in our system, *locations* (L) and *image objects* (I). Locations are the areas of an image which can change value. The values these locations can be set to are called image objects. A location can also, optionally, have a *clear* value, which indicates that no image object is at that location. An image object's value is a location. We use the term "value" rather than "state" as "state" describes the entire environment's state (i.e., all object's values).

Definition 3. *Predicates and Atoms:* A predicate consist of a name and a typed parameter list. For instance, when grounded the `(at ?1 - location ?2 - image_object)` predicate indicates an image object is at a location. An atom is a grounded predicate, and can be fluent or static.

Definition 4. *States:* A state consists of fluent atoms, which represent all objects' values.

Definition 5. *Actions:* Each action ($a \in A$) can be cast to a transition ($t \in T$), and vice versa. An action modifies a subset of the fluents within a predecessor state (a_{ps}) to reach a successor state (a_{ss}). It is comprised of a name, arguments, preconditions (a_{pre}) and effects (a_{eff}). Preconditions include known preconditions, containing the fluent atoms the action modifies, and possible preconditions, which include both static and fluent atoms. Effects contain fluent atoms, denoting the modified objects' value after the action has been executed.

Definition 6. *Action definitions:* An action definition is an ungrounded action; Listing 1 provides an example.

The aim of our work is to automatically generate a set of generic action definitions that can be provided to task planners, along with an initial and goal state.

```
(:action move
  :parameters (?disc - disc ?from ?to - tower)
  :precondition (and  (clear ?disc) (clear ?to)
    (smaller ?to ?disc) (on ?disc ?from)  )
  :effect (and  (clear ?from) (not (clear ?to)))
    (not (on ?disc ?from)) (on ?disc ?to)  )
)
```

Listing 1: An example action definition. This action definition is the ground truth for the ToH domain, and was originally produced by Geffner and Assanie (2012). The original used untyped parameters; we have inserted the types.

Throughout this paper, examples from a Towers of Hanoi (ToH) domain are provided. In this domain, there are three towers and four discs of differing sizes. Larger discs cannot be placed on smaller discs. An example is provided in Figure 2. Our experimental results also show our approach successfully learns the action definitions for Puzzle domains (e.g., Figure 1) and Lights-Out domains (e.g., Figure 3). For all domains, the transitions were created from the work of Asai and Fukunaga (2018).

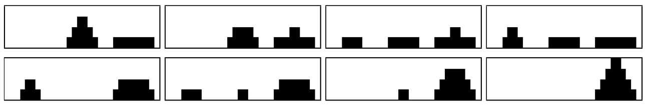


Figure 2: Example of a plan for solving a ToH problem. The initial state is shown in the top-left and the goal state in the bottom-right.

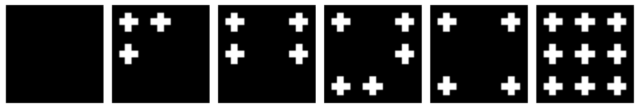


Figure 3: Example plan for solving a Lights-Out problem. When a location is pressed, it and its vertically and horizontally adjacent locations change value (i.e., switch on/off).

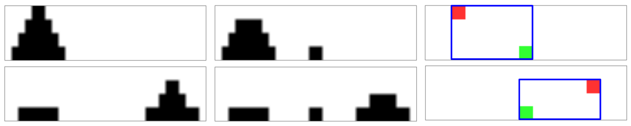
Discover Objects

Our system starts by discovering the locations (L) and image objects (I). Although we do not assume the set of transitions is complete, for this step to work transitions (T) must include multiple instances of each location changing state and each image object (e.g., disc) must change state at least once. All objects are provided with a unique ID to enable them to be transformed to/from PDDL.

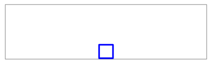
Discover Locations

A location has a (x, y) position, a width and a height, and encompasses a group of pixels that change values simultaneously. The algorithm that discovers the locations is described below.

Each transition is processed in turn. The area of the images that changes is discovered by finding the minimum and maximum x, and minimum and maximum y of the pixels that change value ($\forall(x,y) : t_{si}[x,y] - t_{pi}[x,y] \neq 0$). This area is then checked, to see if it overlaps any of the previously discovered areas. If so, the overlapping area is taken and shrank to cover just the pixels (within the overlap) that change value (e.g., as depicted in Figure 4). These areas, as well as the original area, are appended onto the list of discovered areas.



(a) Predecessor (left) and successor (middle) images of two transitions. The difference between the predecessor and successor is represented in the right images; red represents positive pixel values and green shows negative values. The blue box incorporates all pixels whose value is non-zero, i.e., whose value has changed.



(b) The changed image area, shared by the two transitions.

Figure 4: Two transitions and their overlapping changed image area.

The discovered areas are then iterated over, starting with the smallest, to create the set of locations (L). If the image area does not overlap an already created location ($l \in L$), it is inserted into the set of locations. If the image area only overlaps a single previously created location ($l' \in L$) and it fully encompasses that location, it is appended to the set of locations and the location (l') is removed. For the ToH domain, this procedure results in the locations depicted in Figure 5.

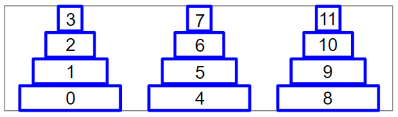


Figure 5: The discovered location definitions for a ToH domain.

The resulting locations are provided with a list of possible values. If a location always transitions to/from a certain value, that value becomes its *clear* value; `clear` is the name of a predicate. If a location only ever has one of two values, one of those values is randomly selected as the clear value. The locations' remaining possible values are image objects.

Discover Image Objects

Image objects are extracted from the transitions. For each transition, the locations which encompass the pixels that change are discovered (e.g., Figure 6). If a location's value matches its clear value, it is ignored. Otherwise, the image is cropped to the changed pixels, contained within the location. These pixels make up an image object. If a location's centre and an image object's centre do not match, the location will also contain an offset (per image object). For the example ToH domain, this results in 4 image objects (i.e., black rectangles) being discovered.

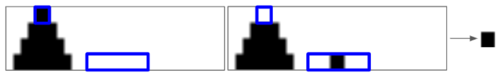


Figure 6: Predecessor (left) and successor (middle) images of a transition. Blue boxes indicate the locations that encompass the changed pixels. The discovered image object is shown on the right; duplicate image objects are ignored as they are equivalent.

Generate States

This section describes how a state (i.e., set of fluent atoms) is generated from an image. Further to performing this on the predecessor and successor images of the transitions, the initial and goal image of a planning problem, and the observations from a goal/plan recognition problem, can be transformed into PDDL states by running this process.

Fluent atoms are created from an image by iterating over the location definitions, and determining which image object is at that location. To prevent the smallest image object from matching all occupied locations, image objects are sorted by area, largest first. This process results in a state containing groundings of the predicates `(`*`at`*` ?1 - location ?2 - image_object)` and `(`*`clear`*` ?1 - location)`.

Generate Action Definitions

The predecessor (a_{ps}) and successor (a_{ss}) states of an action are generated by the process described in the previous section. From these states, each action's effects (a_{eff}) and preconditions (a_{pre}) are determined. Preconditions include known preconditions and possible preconditions. The known preconditions are the fluents that change value when the action is executed; and the effects are their resulting values. All fluent atoms, not in the set of known preconditions, are set as the possible preconditions. The use of possible preconditions was inspired by the works on goal recognition and planning in incomplete domains (Weber and Bryce 2011; Pereira, Pereira, and Meneguzzi 2019). Before transforming actions into action definitions, static atoms are created and the set of possible preconditions is reduced. These static atoms prevent invalid actions being produced from the resulting action definitions.

Linked Locations

The first static atoms to be created, link together the locations that change value simultaneously. For instance, in the transition depicted in Figure 6, locations `loc_3` and `loc_4` change value. Therefore, these locations are linked by a static atom, i.e., `(linked_2 loc_3 loc_4)`. The index within the atom name indicates the number of objects; it is used because predicates have a fixed sized parameter list and differing numbers of locations can change value simultaneously. This atom is appended to the corresponding action's preconditions. Creating these atoms prevents invalid actions being generated from the action definitions produced for domains in which only certain locations can change value simultaneously.

Finding Locations' Dependencies

A location's ability to change value could depend on the state of another object (i.e., location or image object). For instance, in the ToH problem (Figure 5), for `loc_2` to change state, `loc_1` must be occupied and `loc_3` must be clear. Therefore, `loc_2` depends on `loc_1` and `loc_3`, i.e., the static atom `(depends_on_3 loc_2 loc_1 loc_3)` is required. This section details how this atom is created.

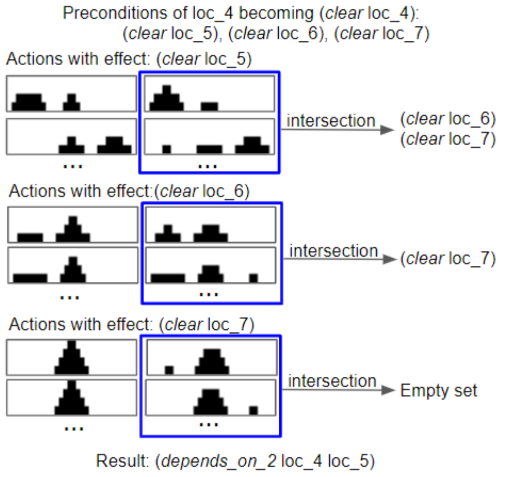
For each location ($l \in L$), the actions that result in each of its possible values are found. If a location always transitions between clear and another value, for this process only, the location has two possible values, clear and unclear. From the actions which set the same value for a location (l^v), e.g., all the actions that set `loc_4` to clear, the common possible preconditions are extracted. These, extracted atoms, become the preconditions of the location (l^v_{pre}).

The location's preconditions are reduced by removing the preconditions that are also preconditions of other preconditions. For each precondition ($p \in l^v_{pre}$), the actions that set the precondition are discovered (e.g., left side of Figure 7). The intersection of these actions' predecessor atoms and the location's preconditions (l^v_{pre}) is taken (e.g., right side of Figure 7). The resulting atoms of the equally largest sets are removed from the location's preconditions (l^v_{pre}); the precondition (p) these belong to is not removed. The arguments of the remaining preconditions become location's dependencies (D^l), and thus the arguments of a `depends_on` atom. These atoms are appended to the actions', which set the location's (l's) value, possible preconditions. Note, locations can depend on image objects as well as other locations, e.g., when `loc_3` changes state `image_14` is always at `loc_2`, thus `loc_3` depends on `loc_2` and `image_14`.

Finding Image Objects' Dependencies

Image objects also require dependencies to, for example, prevent a disc from being placed on a smaller disc. The process described in this section, is performed on each image object.

The actions ($A^i \subseteq A$) that change the image object's (i) value are discovered and, for each of these actions ($a \in A^i$), a set of dependencies (D^i) is created. The locations ($L^c \subseteq L$) that change state (i.e., locations mentioned in a_{eff}) are extracted and, for each location ($l \in L^c$), their dependencies (D^l) are iterated over. The image objects that are either at the location of a dependency ($d \in D^l$) or are a dependency ($d \in D^l$) themselves are inserted into the new dependency set (D^i). If none of the location's dependencies (D^l) have an image object (i.e., they are all clear), then the location (l) itself is inserted into the dependency set (D^i). After processing all locations that change state, a `depends_on` atom is created from D^i. The arguments of the `depends_on` atom are the image object (i) itself, followed by, the set of dependencies (D^i). This atom is inserted in the action's (a's)



Preconditions of loc_4 becoming (*clear* loc_4):
(*clear* loc_5), (*clear* loc_6), (*clear* loc_7)
Actions with effect: (*clear* loc_5)
intersection
(*clear* loc_6)
(*clear* loc_7)
Actions with effect:(*clear* loc_6)
intersection
(*clear* loc_7)
Actions with effect: (*clear* loc_7)
intersection
Empty set
Result: (*depends_on_2* loc_4 loc_5)

Figure 7: Whenever `loc_4` becomes clear, `(clear loc_5)`, `(clear loc_6)` and `(clear loc_7)` are true. For each of these atoms, the actions whose effects include the atom are shown on the right; actions are depicted by (horizontally adjacent) predecessor and successor images. Taking the intersection of `loc_4`'s preconditions and the predecessor states of actions with `(clear loc_5)` as an effect, results in [`(clear loc_6)`, `(clear loc_7)`]. As this is the largest set of resulting atoms (see right column), `(clear loc_6)` and `(clear loc_7)` are removed from `loc_4`'s preconditions; and `(clear loc_5)` is set as irremovable. Thus, loc_4 is said to depend on `loc_5`, i.e., `(depends_on_2 loc_4 loc_5)`.

possible preconditions. Several examples are provided in Figure 8.

Remove Unrequired Possible Preconditions

So far the preconditions of the actions have not been reduced, and thus contain the entire state space. An action ($a \in A$) only requires, as its preconditions (a_{pre}), the atoms of the objects that change value and the atoms of the objects those objects depend on. Therefore, all other fluent atoms are removed from the action's possible preconditions.

Possible preconditions are used because when transitions are missing, additional `depends_on` atoms could be created, and thus appear in the possible preconditions (along with objects' fluent atoms). In future work, this set of possible preconditions can be provided to goal recognisers (Pereira, Pereira, and Meneguzzi 2019) and task planners (Weber and Bryce 2011) designed for incomplete domains.

Generate Action Definitions

Action definitions are generated from the set of actions (A). These are generated by iterating over all actions, checking if a grounding of an already generated action definition is equivalent to that action (i.e., same arguments, preconditions and effects), and if not, creating a new action definition. A new action definition is created by ungrounding the action, i.e, replacing actual objects (e.g., `loc_1`,

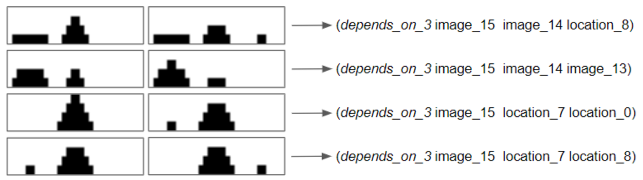

(depends_on_3 image_15 image_14 location_8)
(depends_on_3 image_15 image_14 image_13)
(depends_on_3 image_15 location_7 location_0)
(depends_on_3 image_15 location_7 location_8)

Figure 8: The image objects' `depends_on` atoms (right) that have been created from a subset of actions (left). `image_15` is the smallest disc and `image_14` is the second to smallest disc. In the first transition, `loc_6` and `loc_8` change state. `loc_6` depends on `loc_5` and `loc_7`. As `image_14` is at `loc_5` and `loc_7` is clear, only `image_14` is appended to image object's dependencies (D^i). `loc_8` depends on `loc_9`, and as `loc_9` is empty, `loc_8` is append to D^i. Thus, `image_15` depends on `image_14` and `loc_8`.

`image_12`) with typed parameters (e.g., `?l - location`, `?i - image_object`).

Handling Large State Spaces

For large state spaces, we propose generating the action definitions from a smaller state space (e.g., a 2 by 2 puzzle). Then, using the knowledge gained, create the objects and static atoms of a larger state space (e.g., a 3 by 3 puzzle) from the transitions applicable to a single state. This is currently only possible for domains containing objects of equal size and whose adjacent locations are linked. If these conditions are not met, the method described in the previous sections is invoked.

The size of the objects is discovered by finding the smallest overlapping area of a change. This is performed using the already described process but only the transitions from a single state are processed. To create all locations (L), the algorithm iterates over the pixels of a single image, using the size of the objects plus the gap between objects as a step size. The value of these locations (in the image), are set as the image objects (I). Note, the gap between locations is set using the number of objects in the linked atoms and the size of the changed image area. Static atoms are then created. If in the original problem the `linked` atoms contain two objects, then for every location a `linked` atom is created for each of its adjacent locations (e.g., `(linked_2 loc_0 loc_1) (linked_2 loc_0 loc_3)`). Otherwise, a single `linked` atom is created for each location, to link it to all its adjacent locations (e.g., `(linked_3 loc_0 loc_1 loc_3)`).

Experiments and Discussion

This section discusses the action definitions that were produced by our approach, the time taken to generate the actions definitions, the time it takes a task planner to find a plan and how missing transitions affected the production of PDDL. Experiments were ran on server with Intel Xeon 2.27 GHz CPU and 11 GB of RAM.

Action Definitions

This section describes the action definitions produced for the ToH, Puzzle and Lights-Out domains and mentions the minimum number of transitions and state space required to produces these action definitions. The full sets of action definitions are available at: https://doi.org/10.5281/zenodo.3595871.

Towers of Hanoi Domains For the ToH domains containing 3 or more discs, 6 action definitions were generated. One for when the two modified locations are within the middle of the towers (8 parameters); two for when one modified location is at the bottom of the image and the other is in the middle (i.e., a block being moved to a bottom location and a block being moved from a bottom location) (6 parameters); one for when both locations are at the bottom (4 parameters); and two to handle a block being moved to/from a top location (5 parameters). To create these action definitions, a minimum of 3 towers and 3 discs is required. Due to the complexity of this domain (i.e., many `depends_on` atoms being required), nearly all transitions must be provided as input. An example action definition is provided in Listing 2.

```
(:action action_1
 :parameters (?0 ?1 ?2 ?3 ?4 ?5 - location ?6 ?7 ?8 -
     ↪image_object)
 :precondition (and
   (clear ?0)
   (depends_on_3 ?1 ?0 ?3)  (depends_on_3 ?1 ?3 ?0)
   (at ?6 ?1)  (not (clear ?1))
   (linked_2 ?1 ?4)  (linked_2 ?4 ?1)
   (at ?7 ?2)  (not (clear ?2))
   (depends_on_3 ?4 ?2 ?5)  (depends_on_3 ?4 ?5 ?2)
   (not (clear ?3))
   (at ?8 ?3)  (not (at ?6 ?4))
   (clear ?4)  (clear ?5)
   (depends_on_3 ?6 ?8 ?7)  (depends_on_3 ?6 ?7 ?8)
 )
 :effect (and
   (not (at ?6 ?1))  (clear ?1)
   (at ?6 ?4)  (not (clear ?4))
 )
)
```

Listing 2: Example action definition. The two locations being modified are within the middle of the towers.

Puzzle Domains An example plan for a Digital puzzle problem is shown in Figure 1; a picture of a mandrill and a spider was split into titles to produce Mandrill and Spider puzzles, respectively. For the Digital, Mandrill and Spider puzzle domains, a single correct action definition was created. The minimum Puzzle size to generate this action definition is 2 by 2. A 1 by 2 Puzzle results in the creation of unnecessary `depends_on` atoms.

For a 3 by 3 puzzle domain, only 12 transitions are required as input; however, these 12 transitions must be carefully selected. Each location must be acted on at least twice; the location must either transition to clear or not-clear in both transitions and each of the remaining locations must have a different value for both transitions. The likelihood of

12 randomly selected transitions meeting these requirements is less than 2 %.

Lights-Out Domains For Lights-Out, 4 action definitions were generated for a 2 by 2 domain and 15 actions were generated from a 3 by 3, or larger, domain. In these domains, either 3, 4 or 5 locations can change state simultaneously; the action definitions include (for each of these values) handling when no lights are on, when 1 light is on, when 2 are on, etc.. The action definitions of a 3 by 3 Lights-Out domain can be generated from 21 selected transitions.

Time Complexity

The most time complex aspects of our implementation are: discovering the locations and creating the locations' `depends_on` atoms. Discovering locations has a quadratic time complexity of $O(n\ log2^n)$ and creating the locations' dependencies is $O(lno)$; where $n = |T|$, $l = |L|$ and $o = |L|+|I|$. Increasing the number of image pixels also impacts the location discovery time. For instance, to generate the action definition of a Puzzle problem with 10*12 pixels it takes 0.64 seconds, whereas 43*43 pixels takes 9.83 seconds. The action definition generations times for various domains are provided in Table 1. All times are the average of 5 runs and are in seconds. For the 3 by 3 Digital and Lights-Out domains the minimum number of transitions are provided as input because processing all transitions, i.e., 967 680 transitions for Digital and 4608 for Lights-Out, would be computationally expensive.

Table 1: Action definition generation results. Res is the image resolution, $|T|$ is the number of transitions that were provided as input, $|L|$ the number of discovered locations, $|I|$ the number image objects and $|A^D|$ the number of generated action definitions.

Domain	Config	Res	$\lvert T\rvert$	$\lvert L\rvert$	$\lvert I\rvert$	$\lvert A^D\rvert$	Avg Time
Digital	2 by 2	10*12	48	4	3	1	0.64 ± 0.00
Digital	3 by 3	15*18	*12*	9	8	1	0.37 ± 0.01
Mandrill	2 by 2	43*43	48	4	3	1	9.83 ± 0.06
Spider	2 by 2	57*57	48	4	3	1	17.15 ± 0.10
Lights-Out	2 by 2	10*10	64	4	1	4	1.51 ± 0.00
Lights-Out	3 by 3	15*15	*21*	9	1	15	1.47 ± 0.02
ToH	2 discs	36*8	24	6	2	3	0.89 ± 0.00
ToH	3 discs	48*12	78	9	3	6	12.04 ± 0.05
ToH	4 discs	60*16	240	12	4	6	115.38 ± 1.55
ToH	5 discs	72*20	726	15	5	6	1012.99 ± 6.08

After using a 2 by 2 Digital puzzle to generate the actions definitions, the objects and static atoms of 3 by 3 Puzzle (i.e., Digital, Mandrill or Spider) domains were discovered in an average of 1.25 seconds. For Lights-Out, the objects and static atoms of a 4 by 4 domain were created in 4.48 seconds.

Task Planning Results: No Missing Transitions

During the experiments, initial and goal states were generated from two images and these states, along with the static atoms, objects and action definitions, were provided to a task planner, i.e., Fast Downward (FD) (Helmert 2006). 100 goal and initial images were selected at random from a set of all possible images. No goal image and initial image pairing is repeated, and the problem could be unsolvable (i.e., the goal unreachable). Each experiment was ran 5 times; the average total time (and standard deviation), reported by FD, is provided in Table 2.

Table 2: Time spent planning. $|probs|$ is the number of problems (i.e., goal and initial images) and $\overline{|A^P|}$ is the average number of actions in the produced plans. In the two disc ToH domain, there are only 72 unique problems.

Domain	Config	$\lvert probs\rvert$	$\overline{\lvert A^P\rvert}$	Avg Time	Total Time
Digital	2 by 2	100	1.44 ± 1.93	0.00 ± 0.00	0.35
Digital	3 by 3	100	11.22 ± 11.17	0.47 ± 0.23	46.87
Mandrill	2 by 2	100	1.49 ± 1.89	0.00 ± 0.00	0.35
Spider	2 by 2	100	1.48 ± 1.89	0.00 ± 0.00	0.35
Lights-Out	2 by 2	100	2.11 ± 0.92	0.00 ± 0.00	0.42
Lights-Out	3 by 3	100	4.59 ± 1.50	0.02 ± 0.00	2.39
Lights-Out	4 by 4	100	0.12 ± 0.68	0.24 ± 0.03	23.66
ToH	2 discs	72	2.00 ± 0.82	0.00 ± 0.00	0.24
ToH	3 discs	100	4.19 ± 2.01	0.00 ± 0.00	0.47
ToH	4 discs	100	8.57 ± 3.99	0.01 ± 0.00	0.96
ToH	5 discs	100	16.52 ± 6.92	0.02 ± 0.00	2.32

The longest time it took FD to generate a plan was 0.71 seconds and on average 0.06 seconds were taken. The state of the art approach, LatPlan (Asai and Fukunaga 2017; 2018), reported that it took 4 hours for an adapted version of FD to solve a generated PDDL problem. Thus, the PDDL produced by our approach is more efficient.

After finding a plan, the planned actions were translated into images. The first actions effects were applied to the initial image, then the subsequent action's effects were applied to the resulting image and so on. `(clear ?location)` effects were applied to the image by setting the pixels, corresponding to the location, to the location's clear state. `(at ?location ?image-object)` effects were applied by setting the corresponding pixels to the corresponding image object. For each experiment, this took less than 0.01 second.

Task Planning Results: Missing Transitions

Experiments were ran for various percents of missing transitions. For each percentage, 5 sets of transitions were randomly selected and, as before, 100 goal and initial state images were selected. The results are presented in Table 3.

For Puzzle problems, when $\leq$70 % of transitions were missing, the correct action definitions and plans were generated. At 80 % the performance greatly deteriorated; plans were only generated 53 % of the time. Nevertheless, when the plan was generated, it was the same length as the plan produced when no transitions are missing.

When 20 % of transitions were missing, Lights-Out was unsolvable 3 % of the time and for 18 % of solved problems, the plan was an incorrect length. Most incorrect plans were shorter than the actual plan length. This is because some locations were not identified, and thus not included within the

Table 3: Accuracy for identifying locations (L_c), image objects (I_c), action definitions (A^D_c) and the plan when differing percentages of transitions are missing ($m\%$). C is the ratio of times a plan was successfully created. A^P_c shows how often a successfully created plan's length matches the actual plan's length and D is the average length difference.

Domain	**Config**	$m\%$	L_c	I_c	A^D_c	C	A^P_c	D
Puzzle	2 by 2	70	1.00	1.00	1.00	1.00	1.00	0.00
		80	0.80	0.80	0.80	0.53	1.00	0.00
		90	0.13	0.13	0.13	0.04	1.00	0.00
Lights-Out	2 by 2	20	0.60	1.00	0.60	0.97	0.82	-0.21
		40	0.00	1.00	0.00	0.93	0.53	-0.56
		60	0.40	1.00	0.40	0.96	0.72	-0.26
		80	0.40	1.00	0.60	0.95	0.81	-0.23
ToH	5 discs	20	0.60	1.00	1.00	1.00	0.93	1.08
		40	0.20	1.00	0.60	1.00	0.74	1.53
		60	0.00	0.20	0.00	0.79	0.23	-4.73
		80	0.00	0.60	0.00	0.31	0.27	4.01

initial and goal states. For ToH, the discovered plan was often longer because additional `depends_on` atoms (and action definitions) were generated.

Related Work

Many prior works (Yang, Wu, and Jiang 2005; 2007; Walsh and Littman 2008; Amir and Chang 2008; Cresswell, McCluskey, and West 2013; Mourão et al. 2012; Bonet and Geffner 2019) have attempted to learn the actions' preconditions and effects when provided with some symbolic knowledge. In ARMS (Yang, Wu, and Jiang 2007) the initial state, goal state, predicates and a plan, containing actions along with the objects that change state (i.e., the action signature), are provided as input. Their work attempts to guess the action model (i.e., the actions' preconditions and effects) that best matches the plan. Whereas, SLAF (Amir and Chang 2008) enables the actions effects and preconditions to be discovered from a sequence of actions and partially observable states; and the action model is updated online. LOCM (Cresswell, McCluskey, and West 2013) also takes plans (and partial plans) as input but does not require the predicates, initial state or goal state. Nevertheless, all these approaches require a domain engineer to provide some symbolic information.

Previous methods of transforming unlabelled image pairs into symbolic models, such as LatPlan (Asai and Fukunaga 2018) and the work of Amado et al. (2018), involved an deep autoencoder and, for the production of PDDL, required all transitions. Both AMA$_1$ (Asai and Fukunaga 2018) and (Amado et al. 2018) perform a bitwise comparison of pairs of encoded images to determine the actions' effects. They do not attempt to determine what objects are present. Moreover, training an deep autoencoder can be computationally expensive, and when action definitions are produced from the actions, they are likely to be problem (as well as domain) specific, i.e., only work on problems containing the same objects.

LatPlan's AMA^2 (Asai and Fukunaga 2018) only required a subset of transitions; however, does not produce PDDL, and thus was incompatible with existing planners. Moreover, during the training phase, AMA$_2$ requires examples of (possibly) invalid states. LatPlan's AMA$_1$ has also been expanded to work with learnt predicates (Asai 2019). This greatly reduced the planning time; however, the objects the images contain were assumed to be known. In future work, we will experiment with applying the techniques introduced in this paper to LatPlan. This will potentially increase the generalisability of our approach whilst reducing the size of the representation produced by LatPlan.

Many researches have attempted to transform images into symbolic representations. For instance, the labels (symbolic annotations) produced by object recognition, detection and segmentation algorithms (Liu et al. 2020). These algorithms could replace our object discovery method; however, often these methods are unexplainable and require a large amount of labelled training data.

Conclusion

Our work transforms unlabelled pairs of images into PDDL. Locations and image objects are discovered by finding the pixels that change value simultaneously, which enables the images to be transformed into symbolically represented states. Actions, containing predecessor and successor states, are generated from transitions. The atoms of the predecessor and successor states that change, are set as the known preconditions and effects of an action. All other atoms form the action's set of possible preconditions. Subsequently, static atoms are created, including atoms that link together locations that change state simultaneously, and atoms that express the locations' and image objects' dependencies. These, static atoms, are used to reduce the actions' possible preconditions. Actions are then converted into action definitions, which can be processed by goal/plan recognisers and task planners. The produced action definitions for Puzzle, Lights-Out and ToH domains were evaluated by calling a task planner.

In future work, we will evaluate the produced PDDL using goal/plan recognition approaches. Moreover, how well goal recognition design approaches (Keren, Gal, and Karpas 2014; 2018; Harman and Simoens 2019) perform on our learnt PDDL will be assessed; these methods modify the PDDL to reduce the number of observations required to determine the observee's goal. We also intend to enhance the image processing method currently integrated into our approach. Rather than cropping all objects into rectangles, a higher resolution boundary should be created. Further, processing video rather than pairs of images will be investigated. Learning symbolic action definitions from images is still a relatively unexplored research area; we hope that our work can inspire others to further advance this research field.

Acknowledgements

The research leading to this article was funded through an SB Fellowship of the FWO-Vlaanderen (project no. 1S40217N). Helen Harman is currently affiliated with the University of Lincoln.

References

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning strips action models with classical planning. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*, ICAPS'18. AAAI Press.

Amado, L.; Pereira, R. F.; Aires, J.; Magnaguagno, M.; Granada, R.; and Meneguzzi, F. 2018. Goal recognition in latent space. In *2018 International Joint Conference on Neural Networks*, IJCNN'18, 1–8.

Amir, E., and Chang, A. 2008. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research* 33:349–402.

Asai, M., and Fukunaga, A. 2017. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. *arXiv preprint arXiv:1705.00154*.

Asai, M., and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, AAAI'18. AAAI Press.

Asai, M. 2019. Unsupervised grounding of plannable first-order logic representation from images. *arXiv preprint arXiv:1902.08093*.

Bonet, B., and Geffner, H. 2019. Learning first-order symbolic planning representations from plain graphs. *arXiv preprint arXiv:1909.05546*.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using locm. *The Knowledge Engineering Review* 28(2):195–213.

Geffner, H., and Assanie, M. 2012. The towers of hanoi problem (formalisation by hector geffner). https://github.com/SoarGroup/Domains-Planning-Domain-Definition-Language/blob/master/pddl/hanoi.pddl. Accessed: 2019-11-13.

Harman, H., and Simoens, P. 2019. Action graphs for performing goal recognition design on human-inhabited environments. *Sensors* 19(12):2741.

Harman, H., and Simoens, P. 2020. Generating symbolic action definitions from pairs of images: Applied to solving towers of hanoi. In *AAAI 2020 Workshop on Plan, Activity, and Intent Recognition (PAIR)*, 1–8.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Kambhampati, S. 2007. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, volume 2 of *AAAI'07*, 1601–1604. AAAI Press.

Kautz, H. A., and Allen, J. F. 1986. Generalized plan recognition. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI'86, 32–37. AAAI Press.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, ICAPS'14, 154–162. AAAI Press.

Keren, S.; Gal, A.; and Karpas, E. 2018. Strong stubborn sets for efficient goal recognition design. In *International Conference on Automated Planning and Scheduling*, ICAPS'18, 141–149. AAAI Press.

Liu, L.; Ouyang, W.; Wang, X.; Fieguth, P.; Chen, J.; Liu, X.; and Pietikäinen, M. 2020. Deep learning for generic object detection: A survey. *International journal of computer vision* 128(2):261–318.

McDermott, D. 2000. The 1998 ai planning system competition. *AI Magazine* 21(2):35.

Mourão, K.; Zettlemoyer, L.; Petrick, R. P. A.; and Steedman, M. 2012. Learning strips operators from noisy and incomplete observations. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, UAI'12, 614–623. AUAI Press.

Pereira, R. F.; Pereira, A. G.; and Meneguzzi, F. 2019. Landmark-enhanced heuristics for goal recognition in incomplete domain models. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling*, ICAPS'19, 329–337. AAAI Press.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, 1121–1126. AAAI Press.

Walsh, T. J., and Littman, M. L. 2008. Efficient learning of action schemas and web-service descriptions. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence*, volume 2 of *AAAI'08*, 714–719. AAAI Press.

Weber, C., and Bryce, D. 2011. Planning and acting in incomplete domains. In *Proceedings of the Twenty-First International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'11, 274–281. AAAI Press.

Yang, Q.; Wu, K.; and Jiang, Y. 2005. Learning action models from plan examples with incomplete knowledge. In *Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'05, 241–250. AAAI Press.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence* 171(2):107 – 143.