# Generating Symbolic Action Definitions from Pairs of Images: Applied to Solving Towers of Hanoi

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

### Abstract

Goal recognisers attempt to infer an observed agent's intentions from a sequence of observations. Whereas, task planners, when provided with an initial and goal state, attempt to create a plan, i.e., sequence of actions. Both goal recognisers and task planners require a symbolic representation of the actions an agent can perform; however, the development of these representations can be time consuming and complex. Therefore, our work takes unlabelled pairs of images, namely, transitions, as input and transforms them into symbolic actions, which are subsequently reduced to a set of action definitions. These action definitions contain a set of effects and preconditions, including known preconditions (i.e., atoms representing the objects whose value changes) and possible preconditions. We aim to automatically generate generic (reusable) action definitions and, through the use of possible preconditions, aim to reduce the number of transitions that must be provided as input. To test the produced action definitions a task planner is called. This paper presents an early (conceptual) version of our work and focuses on a Towers of Hanoi domain.

## Introduction

The actions agents can perform, to modify the state of the environment, are often modelled in symbolic languages, such as STRIPS (Fikes and Nilsson 1971) and PDDL (McDermott 2000). Rather than defining every possible action, 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 main aim of our work is to automatically generate action definitions from unlabelled pairs of images, which show the state before and after an action has been executed.

Symbolic models of an agent's behaviour are required by goal/plan recognisers and tasks planners. Plan and goal recognisers (Pereira, Pereira, and Meneguzzi 2019; Pereira, Oren, and Meneguzzi 2017; E-Martin, R-Moreno, and Smith 2015; Ramírez and Geffner 2010; Kautz and Allen 1986) attempt to infer an observed agent's goal and/or plan from a sequence of observations. Whereas, task planners (Weber and Bryce 2011; Helmert 2006) find a sequence of discrete actions, which achieves a given goal.

Currently, symbolic action definitions are usually manually written. This can be a burdensome task and often requires domain specific knowledge (Weber and Bryce 2011; Kambhampati 2007). 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); however, until recently, some symbolic knowledge was still necessary. Unlabelled pairs of images, namely, transitions, were provided as input to LatPlan (Asai and Fukunaga 2018), which employed a deep autoencoder to generate the action definitions. Nevertheless, to generate PDDL, all transitions were supplied as input. Moreover, deep-learning approaches can require a lengthy training time and the decisions made by the algorithms are likely to be unexplainable.

Our work aims towards creating generic (reusable) action definitions. 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' known preconditions, possible preconditions and effects to be determined. Actions are then converted into a set of action definitions, which can be provided to goal/plan recongisers and task planners. In this paper, the produced action definitions are evaluated by generating a initial and goal state from images, then calling a task planner. The resulting task plan is converted back into images.

This paper presents an early version of our work and focuses on solving a Towers of Honoi (ToH) domain. An experiment in which all transitions are provided is presented; in future work, experiments with a subset of transitions will be performed. Moreover, as we do not assume all transitions are present (and invalid transitions are not provided) there are likely to be domains our approach does not create valid action definitions for. This is because there is no way to determine if an unseen transition is invalid or just missing. Nevertheless, the decisions made by our approach are explainable and to handle these cases, alterations can be made to our state and precondition generation algorithms.

Although pairs of images are provided as input, the presented processes can be applied to other types of sensor data. Moreover, for simplicity, all objects (within an image) are

assumed to be rectangular, no objects are occluded, and every time an object is present its pixel values are identical (i.e., no noise).

The remainder of this paper is structured as follows. First, a formalisation of the action definition creation problem is provided and the ToH domain is introduced. Second, our object discovery approach is detailed. Subsequently, how an image is translated into a set of fluent atoms is outlined. Fourth, the action generation process, including the creation of static atoms, removal of unrequired preconditions and the action definition generation, is described. In the penultimate section, our preliminary results are presented. Finally, a brief overview of related works is provided.

## Problem Formalisation

Goal recognition and task planning 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 ToH domain is described.

**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" 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; and `(clear ?1 - location)` indicates no image objects are at that 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.

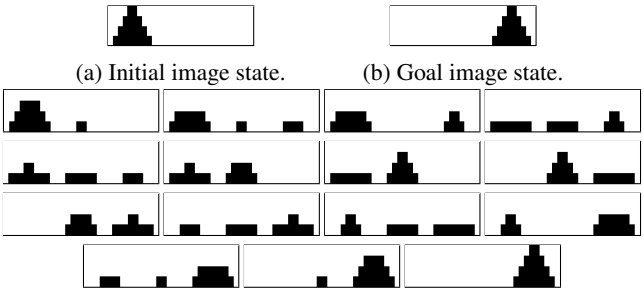
**Definition 5.** *Actions:* Each action ($a \in A$) can be cast to a transition ($t \in T$) and vice versa. An action modifies a subset of the fluents within a predecessor state ($a_{ps} = pddl(t_{pi})$) to reach a successor state ($a_{ss} = pddl(t_{si})$). It is comprised of a name, a set of objects (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 definitions is an ungrounded action; Listing 1 provides an example. The aim of our work is to generate a set of generic action definitions that can be provided to goal/plan recognisers (along with a sequence of observations) and 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. The transitions for this domain were created by Asai and Fukunaga (2018). In this domain, there are three towers and four blocks of differing sizes. Larger blocks cannot be placed on smaller blocks. An example of a initial and goal image, and the planned images to get from the initial to goal image are provided in Figure 1.



(a) Initial image state. (b) Goal image state.

(c) Planned image states.

Figure 1: Example of a plan for solving a ToH problem. This plan (image sequence) was produced using the action definitions created by our approach.

## Discover Objects

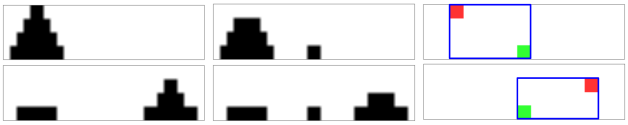
Our system starts by discovering the locations ($L$) and image objects ($I$). Although we do not assume the set of transitions is complete, for this step to work, the transitions must include multiple instances of each location changing state and each image object (e.g., block) must change state at least once. In this section, (x, y) represents the coordinates of a pixel.

### Discover Locations

Locations have a minimum (x, y) position, a width and a height, and, in PDDL, are defined using a unique ID (e.g., `loc_0`). A location 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 checking which pixels change value (i.e., $\forall(x, y) : t_{si}[x, y] - t_{pi}[x, y] \neq 0$) and finding the minimum and maximum x, and minimum and maximum y of a change. 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 Figures 2 and 3). These areas, as well as the original area, are appended into 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 2: Two transitions and their overlapping changed image area.

(a) Predecessor (left) and successor (middle) images of two transitions, and their difference (right).

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

Figure 3: 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 example, the image area depicted in Figure 2b will be removed when the image area of Figure 3b is processed. For the ToH domain, this procedure results in the locations depicted in Figure 4.

The resulting locations are provided with an ID, and a list of possible values. If a location always transitions to/from a certain value, that value becomes its *clear* value. If a location only ever has one of two values, one of those values is selected as the clear value. The process of discovering

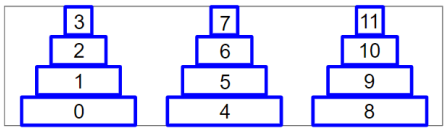



Figure 4: The discovered location definitions. The location IDs produced by our approach are not in this order, this order is used for simplicity.

the other values of a location, i.e., the image objects, is described in the subsequent subsection.

### 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 5). 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 center and an image object's center do not match, the location will also contain an offset (per image object). For the ToH domain, this results in 4 image objects (i.e., black rectangles) being discovered.

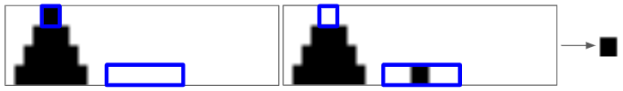


Figure 5: Predecessor and successor images of a transition (left). Blue boxes indicate the locations that encompass the changed pixels. The area at the bottom of the successor image is cropped, so that only the changed pixels are included in the image object. The entirely white areas are ignored as these are the locations' clear values. The discovered image object is shown on the right (duplicate image objects are ignored as they are equivalent).

Rather than extracting the image objects from a single image, all transitions are processed. This is because, if a single image is selected, the resulting image object could contain extra pixels, for example, it could be that the smallest block is at location 4 and thus it would gain all pixels within this location.

Like locations, image objects are provide with a unique ID to enable them to be transformed to/from PDDL. The PDDL produced, by discovering the objects in the example ToH domain, is displayed in Listing 2.

```
(objects: loc_0 loc_1 loc_2 ... loc_11 - location
    image_12 image_13 image_14 image_15 - image_object )
```

Listing 2: PDDL defined objects for the ToH domain.

## Generate States

A state consists of fluent atoms, which represent the objects' values. This section describes how the fluent atoms of a state are generated from a 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.

The 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 following predicates:

- `(at ?1 - location ?2 - image_object)`
- `(clear ?1 - location)`

For the ToH domain, this results in states consisting of 4 `at` atoms (i.e., one for each object) and `clear` atoms for the remaining locations. An example is shown in Figure 6.

```
(at loc_0 image_12) (at loc_1 image_13)
(at loc_2 image_14) (at loc_3 image_15)
(clear loc_4) (clear loc_5) (clear loc_6)
(clear loc_7) (clear loc_8) (clear loc_9)
(clear loc_10) (clear loc_11)
```

Figure 6: The fluent atoms (right) for image shown on the left.

## Generate Actions

A grounded action is created from each of the transitions. 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, the 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).

This section details the creation of static atoms. This includes atoms that link together locations and atoms that express the locations' and image objects' dependencies. Subsequently, which possible preconditions are removed is explained. The transformation, from actions to action definitions, is described in the last subsection.

### 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 5, locations `loc_3` and `loc_4` change value. Therefore, these locations are linked by a static atom, i.e., `(linked loc_3 loc_4)`. This atom is appended to the corresponding action's possible preconditions. Creating these atoms prevents invalid actions being generated from the action definitions produced for domains (e.g., a puzzle domain (Asai and Fukunaga 2018)) in which only certain (e.g., adjacent) 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 4), 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. The `3`, within the atom name, indicates the number of objects; it is used because predicates have a fixed sized parameter list and objects can depend on differing numbers of objects. This section details how this atom is created; and an example is provided in Figure 7.

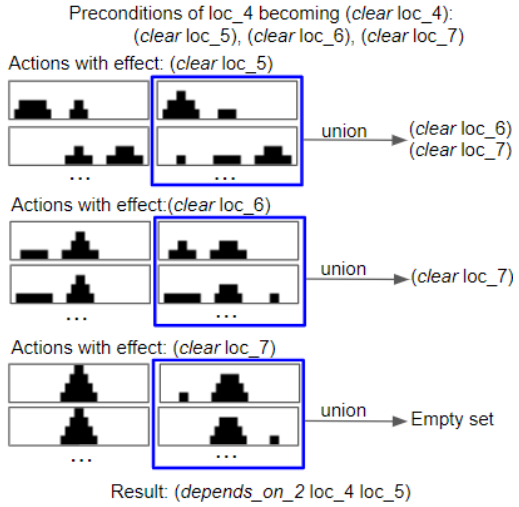



Figure 7: Whenever `loc_4` becomes clear, `(clear loc_5)`, `(clear loc_6)` and `(clear loc_7)` are true. For each of these atoms, the actions whose effects include the atom are shown on the right; actions are depicted by (horizontally adjacent) predecessor and successor images. Taking the union of `loc_4`'s 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)`.

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 (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 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 union 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 set(s), are removed from the location's preconditions ($l^v_{pre}$); the precondition ($p$) these belong to is not removed. The arguments of the remaining (unremoved) preconditions become location's dependencies ($D^l$), and thus the arguments of a depends on atom. These atoms are append 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 larger block from being placed on a smaller block. 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) possible preconditions. Several examples are provided in Figure 8.

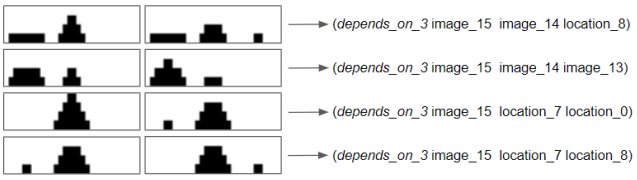



Figure 8: The image objects' `depends_on` atoms (right) that have been created from a subset of actions (left). In the first transition, `loc_6` and `loc_8` change state. `loc_6` depends on `loc_5` and `loc_7`. As `image_14` is at `loc_5` and `loc_7` is clear, only `image_14` is appended to image object's 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`.

### 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 (its static atoms are not altered).

Possible preconditions are used because when transitions are missing, it is likely that additional `depends_on` atoms will be created, and thus appear in the possible preconditions (along with objects' fluent atoms). This set of possible preconditions can be provided to goal reconisers (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, and if not, creating a new action definition. A generated action definition is equivalent an action ($a$) if a grounding of the definition ($a^d$) is equal to the action (i.e., same arguments, preconditions ($a_{pre} = a^d_{pre}$) and effects ($a_{eff} = a^d_{eff}$)).

## Preliminary Results

The action definitions, produced from the ToH domain, are discussed in this section. Subsequently, our initial experimental results are described. These experiments were performed by creating states from initial and goal images, then calling a task planner.

### Action Definitions

For the ToH domain, 6 action definitions are 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). An example, action definition, is provided in Listing 3. The full domain file and an example problem file is available at: `http://doi.org/10.5281/zenodo.3538910`.

Our resulting action definitions differs from the ground truth, which contains a single action definition (see Listing 1) and three towers (as objects). We could adapt our approach, so that vertically aligned positions are detected (i.e., the towers); however, to remain generic, horizontally aligned positions would also require detecting. Therefore, additional action definitions and predicates are still likely to be generated.

```
(:action action_2
 :parameters ( ?0 ?1 ?2 ?3 ?4 ?5 - location
               ?6 ?7 ?8 - image  )
 :precondition (and
     (at ?6 ?0)   (not (clear ?0))
     (clear ?2)   (not (at ?6 ?2))
     (linked ?0 ?2) (linked ?2 ?0)
     (depends_on_3 ?6 ?8 ?7)  (depends_on_3 ?6 ?7 ?8)
     (depends_on_3 ?0 ?4 ?5)  (depends_on_3 ?0 ?5 ?4)
     (at ?8 ?5)   (not (clear ?5))  (clear ?4)
     (depends_on_3 ?2 ?1 ?3)  (depends_on_3 ?2 ?3 ?1)
     (at ?7 ?1)  (not (clear ?1))  (clear ?3)
 )
 :effect (and
     (clear ?0) (not (at ?6 ?0))
     (at ?6 ?2) (not (clear ?2))
 )
)
```

Listing 3: Example action definition. The two locations being modified are within the middle of the towers. As we evaluate our approach using a task planner designed for complete (rather than incomplete) domains, the known and possible preconditions form a single set of preconditions.

The produced action definitions are generic (i.e., will work on ToH problems with differing numbers of blocks and towers); however, it would be very challenging to manually create the objects and states. In future work, we will automatically generate the objects (and states) from a subset of transitions, and then use our, already, generated action definition to perform goal recognition and task planning.

We also ran our system on a puzzle domain in which tiles are rearranged by swapping them with a single clear location. This resulted in a single (correct) action definition, which can also be found at the provided link. As we were reading all images from file, due to space limitations, only a 2 by 2 problem was used. Nevertheless, this action definition is valid for puzzle problems with more pieces.

## Experimental Results for Generating a Plan

To generate a plan, a task planner requires an initial state and a goal state. Therefore, our state generation method (described earlier) created these states from an initial image and a goal image. All static atoms were also inserted into the initial state. These states, along with our generated objects and action definitions, were provided to a task planner, i.e., Fast Downward (FD) (Helmert 2006).

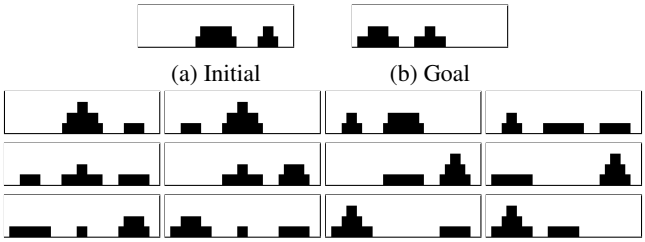
After finding a plan, the planned actions need to be translated into images. The first actions effects were applied the initial image, then the subsequent action's effects were applied to the resulting image and so on. `(clear ?location)` effects were applied to the image by setting the pixels, corresponding 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 our experiment, 10 goal and initial images were selected at random from a set of all possible images; once selected the image was removed from the set. Each experiment was ran 5 times and the average result is provided in Table 1. These experiments were ran on server with Dual-Core AMD Opteron(tm) Processor 2212 and 3.7GB of RAM.

Table 1: The preliminary experimental results.

| Description | Time (seconds) |
|---|---|
| Generating action definitions | 185.34 s |
| Generating initial and goal states | 0.00 s |
| Planner's reported time | 0.01 s |
| Transforming plan to images | 0.00 s |

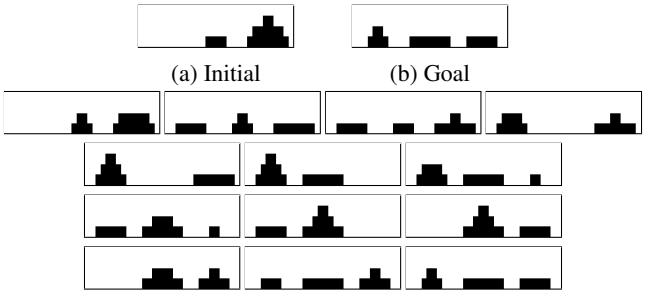
The ToH domain has 240 transitions, which took, on average, 185.34 seconds to transform into objects and action definitions. The total time to generated a plan, as reported by FD, was on average 0.01 seconds. The produced plans had an average length of 9.90 actions. Three example plans, produced when running the experiment, are provided in Figures 9, 10 and 11. These demonstrate valid optimal plans can be produced from our automatically generated action definitions.



(a) Initial (b) Goal

(c) Planned image states.

Figure 9: Example of an image sequence, generated by our approach.



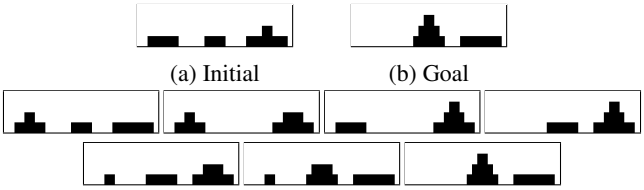
(a) Initial (b) Goal

(c) Planned image states.

Figure 10: Example of an image sequence, generated by our approach.

## 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



(a) Initial (b) Goal

(c) Planned image states.

Figure 11: Example of an image sequence, generated by our approach.

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. In contrast, Yordanova (2017) proposed learning action models from text, then optimising these models using plans. 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 is incompatible with existing (PDDL) 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).

## Conclusion and Future Work

This paper presents an early version of our work, which transforms unlabelled pairs of images (i.e., transitions) into PDDL. We aim to automatically produce generic (reusable) action definitions and, through the use of possible preconditions, aim to reduce the number of image pairs that must be provided as input. In our method, objects (i.e., locations and image objects) are discovered by finding the pixels that change value simultaneously, which enables each image to be transformed into a symbolically represented state. 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 were tested by calling a task planner. This experiment demonstrated that, for the ToH domain, valid action definitions and states were created by our approach.

In future work, we will evaluate the produced PDDL using goal and plan recognisers, which require a set of hypotheses goal states and a sequence of observed states or actions. Therefore, using the method described above, the set of hypothesis goal images and the sequence of images will transformed into states. For goal recognition algorithms requiring observed actions, the sequence of states will be transformed into actions by, for each state, finding and grounding the applicable action definition that reaches the subsequent state. The goal state returned by a recogniser can then simply be mapped back to a hypothesis goal image.

Although our preliminary experiments show promising results, much more extensive experiments are necessary. In future work, our approach will be evaluated using different domains and various amounts of missing transitions. The time complexity for when the number of transitions, size of the images and/or number of objects rises, will be reported. Further, we will investigate the metrics used by other PDDL learning 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 used by our approach. Rather than cropping all objects into rectangles, a higher resolution boundary should be used. Further, rather than pairs of images, processing video will be investigated.

## 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. 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.

E-Martin, Y.; R-Moreno, M. D.; and Smith, D. E. 2015. A fast goal recognition technique based on interaction estimates. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI'15. Buenos, Aires, Argentina: AAAI Press.

Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3):189 – 208.

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.

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.

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.; Oren, N.; and Meneguzzi, F. 2017. Landmark-based heuristics for goal recognition. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, 3622–3628. AAAI 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.

Yordanova, K. 2017. Texttohbm: A generalised approach to learning models of human behaviour for activity recognition from textual instructions. In *AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR)*.