Planning Domain Generation from Natural Language Step-by-Step Instructions

Maurício Steinert, Felipe Meneguzzi
Pontifical Catholic University of Rio Grande do Sul
6681 Ipiranga Avenue
Porto Alegre, Rio Grande do Sul - Brazil
mauricio.steinert@edu.pucrs.br, felipe.meneguzzi@pucrs.br

Abstract

Classical planners generate plans to achieve a specific goal, given a well-defined planning domain, described in a formal language to be processed into a computer. Learning domain descriptions from unstructured raw data is desirable to avoid the knowledge-acquisition bottleneck that emerges from translating problems into a symbolic representation. In this work we propose using natural language input to define planning domains given a sequence of step-by-step instructions to accomplish a task. Specifically, we develop a rule-based solver capable of converting step-by-step instructions into symbolic representation that can be used as input to off-the-shelf planners.

Introduction

Classical planners generate plans to achieve a specific goal, given a well-defined planning domain, described in a formal language, such as PDDL. Given the niche aspect of PDDL modeling, we want to allow users to be able to employ other data formats as input (Amado et al. 2018), preferably ones that resemble human communication like images and natural language, avoiding possible the knowledge-acquisition bottleneck of translating from raw data into symbolic representation (Granada et al. 2017). To overcome this knowledge-acquisition bottleneck, in this work we propose using natural language instructions as input to describe planning domains. To solve this problem, we developed a rule-based solver that operates over sentences that describe step-by-step instructions to accomplish a goal, automatically translating these instructions into a symbolic representation that can be immediately used by off-the-shelf planners.

In order to evaluate our work, we generate domain descriptions from the WikiHow Dataset (Steinert and Meneguzzi 2020)[1], a dataset we assembled specifically for this task that is composed by a large volume of step-by-step instructions. We develop a Rule-based solver to extract information from instructions sentences and translate them into usable PDDL domain and problem specifications, followed by experiments with quantitative and qualitative results evaluation.

[1]Publicly available at https://github.com/pucrs-automated-planning/wikihow-planning-recipes-dataset

Method

In the following sections we detail the general structure and features of our dataset, including its acquisition and preprocessing tasks. Next, we discuss Rule-based solver, an algorithm that performs information extraction over instances in the WikiHow dataset in order to identify actions and objects based on predefined rule sets.

WikiHow Dataset

In order to evaluate our approach, we developed the WikiHow Dataset (Steinert and Meneguzzi 2020) for automated planning. The WikiHow Dataset comprises step-by-step instructions in natural language extracted from the WikiHow website[2]. These articles are organized in categories and contain step-by-step instructions on how to perform tasks. In our specific case, we work with the cooking recipes' category, all articles of which share a similar structure:

- The first sentence of each step summarizes the step and the following sentences usually refine it.
- More complex tasks are divided into sections, where each section is a set of independent steps. For example, pasta recipes have distinct sections on how to cook the pasta and to prepare the sauce in the same recipe.
- Sentences are written in imperative form, *i.e.* each sentence usually starts with a verb, followed by direct object and optional prepositions and object of preposition.

We generated this dataset by extracting data from the HTML files from WikiHow based on a predefined list of URLs of category pages. After acquiring all raw data, we preprocess them by removing HTML tags and identifying instructions steps within sentences. We keep only the first sentence of each step, given that it summarizes a whole step. We break sentences with conjunction *and* within the same sentence in order to facilitate identifying actions and objects as well as their correlation in further processing stages.

[2]https://www.wikihow.com/

Verb	Frequency	Noun	Frequency
add	8,353	water	4.101
heat	4,247	bowl	3,404
place	3,871	oven	2,973
serve	3,032	pan	2,877
mix	2,815	butter	2,583
cook	2,744	sugar	2,375
remove	2,666	oil	2,351
pour	2,466	salt	2,141
cut	2,337	dough	1,958
stir	2,228	cream	1,884

Table 1: Top-10 verbs and nouns dataset frequency

For example, the sentence *add salt and 1 pound (450 g) of pasta to the boiling water* is broken into two sentences *add salt* and *add 1 pound (450 g) of pasta to the boiling water*. The resulting dataset after preprocessing has a total of 5,518 recipes with an average of 13.75 steps per recipe, and each step with an average of 8.74 words. Since we use a rule-based solver that depends on manual parameters' setup, we evaluate the dataset about term frequency, specifically verbs that are used to identify actions and nouns that are used to identify objects. Table 1 reports the top-10 frequency of verbs and nouns found in dataset.

Rule-based Solver

The Rule-based Solver uses a predefined set of actions and objects to process sentences. These sets are manually generated based on the frequency of nouns and verbs in the dataset. Nouns define objects, and for each object it is possible to assign a type. These types are also used in the PDDL formalization, helping off-the-shelf solvers find a plan for a problem. Verbs define actions. To minimize the set of actions, we group actions with similar effects, where a set of keywords can trigger the same action. For example, the *cut*, *slice* and *chop* keywords trigger the action *cut*.

The Rule-based solver works by evaluating each sentence at a time, as we detail in Algorithm 1. For each sentence, we start by identifying actions and objects (Lines 2 and 3). Action and object identification use our predefined set of action keywords and object names, comparing these entries with each word in the sentences. For both cases, we convert each word to its canonical form before comparing to our sets. Using our previous identified actions and objects, we try to identify objects in the current sentence that fit the required parameters for the actions (Lines 4 to 6). These associations are established based on the syntactic function of each term – for example, a verb (action) requires a direct object (an object in which an action is applied). The same concept applies to prepositions and the objects of prepositions. If any parameter is not resolved, then the current action is discarded (Lines 7 and 8). If there are no unsolved parameters, then the current action is added into solved actions set and its effects added as a goal in problem formalization (Lines 9 to 11). Finally, after processing all sentences, both domain and problem PDDL descriptions are generated (Lines 12 and 13) using previously processed information. The last step sends the generated domain and problem PDDL formalization to be processed by an off-the-shelf planner in order to generate a plan (Line 14).

Algorithm 1: Rule-based pseudocode

```
1  for sentence in sentences do
2      actions = identify_actions(sentence);
3      objects = identify_objects(sentence);
4      for action in actions do
5          for parameter in get_parameters(action) do
6              object = fit_action(objects, parameter);
7              if object not found then
8                  unsolvable_parameters = True;
9          if not unsolvable_parameters then
10             add_action_solved_actions(action);
11             add_effect_to_goal(action, object);
12 generate_domain();
13 generate_problem();
14 generate_plan();
```

Besides grouping actions by their effects, we must infer the preconditions of the corresponding actions. Here we assume that all effects of previous actions are preconditions to perform the current action. A major drawback of this design choice is that all actions have the same set of parameters, even if they are not in practice required to perform the current action.

We developed the Rule-based solver using Python and spaCy[3] for part-of-speech tagging. To generate plans, we used Pyperplan (Alkhazraji et al. 2020), a lightweight STRIPS planner written in Python that we configured to perform a blind breadth-first search to solve problems. The Rule-based solver can operate automatically or interactively asking users input to fill in blanks related to action parameters.

Experiments and results

We performed our experiments over all 5.518 instances of the WikiHow dataset. Given that we operate over cooking recipes, we divide our objects into two broad types: vessels and ingredients.

The rule-based solver was capable of returning a valid plan, *i.e.* solving the problem, for 50% of them. This first experiment is performed without any interactive information input. Table 2 summarizes our statistical results, where we observed that the average plan length is really small (1.02 actions) and shows high variance between generated plans (1.54), with the longest plan containing 15 steps.

We also observed an average of 7.25 identified actions for each instance, approximately 52% of actions considering that all evaluated sentences contain valid actions. From these identified actions, we report an average of 0.97 solved actions, *i.e.* actions which parameters are available within evaluated sentence. We also observe that the solver iden-

[3]https://spacy.io/

	Avg	Std	Min	Max
Plans length	1.02	1.54	1	15
Identified actions	7.25	5.14	0	55
Solved actions	0.97	0.99	0	5

Table 2: Statistical analysis of results

Without manual input	With manual input
(squash egg)	(squash egg)
(put egg whisk)	(squash cheese)
(put egg skillet)	(put egg skillet)
(pour egg skillet)	(put egg whisk)
	(put cheese skillet)
	(pour egg skillet)

Table 3: Generated plans with and without manual information input comparison.

tified a maximum of 5 distinct actions for some generated domain formalization.

In order to evaluate how missing information affects generated results, we performed experiments by manually filling in blanks and comparing results. Table 3 compares generated plans with and without manual information input, where we observe that plans with manual information input are more detailed.

Related Work

Using unstructured raw data to define planning domains is an active area of research. Framer (Lindsay et al. 2017) receives sentences with instructions as input and acquires actions and objects using Named Entity Recognition systems, groups similar actions description using a similarity function to finally sends this compiled information for LOCM (Cresswell, McCluskey, and West 2009) to generate a formal domain description. Users must manually eliminate redundancies and fill in blanks.

StoryFramer (Hayton et al. 2017) devises planning domain models from input natural language plot synopses using a generation process similar to Framer, but in this case it generates a domain and problem definition capable of telling the original source story.

Instead of natural language input, LatPlan (Asai and Fukunaga 2018) uses images of states transition as input and, using State Auto-Encoders, generates a latent space representation of states, transitions and actions that can be computed to devise a plan. LatPlan returns either a PDDL model (Action Model Acquisition, or AMA 1) or a set of images that represents the order of actions that must be performed to achieve a goal (AMA 2).

LatPlan's Action Model Acquisition 1 generates large state transition models to be handled by off-the-shelf planners. To overcome this limitation, the *Cube-Space Autoencoder* (Asai and Muise 2020) translates unstructured raw data into PDDL model by jointly computing State Auto-Encoder and Action Auto-Encoder instead of computing them independently, which results in a smaller state transition model. An advantage of converting raw unstructured data to PDDL is that it allows using domain-independent heuristics to improve planning performance.

Conclusion

We developed a rule-based solver capable of generating formal domain and problem descriptions in PDDL format from natural language step-by-step instructions. We evaluated our solver using a cooking domain, but the method is domain-independent: adapting to other domains just requires adjusting actions and objects sets. It depends, however, on well-formed and informative sentences to yield good results.

In the future, we aim to improve the dataset preprocessing stage in order to better identify action parameters that may be implicit in current sentence or referred in previous ones. Given that more actions would be available during planning stage, this improvement should yield more detailed and accurate plans. Automating action identification within sentences not based on static keywords but evaluating words similarities by using Named Entity Recognition mechanisms, synonyms dictionaries or word embedding vectors (Mikolov et al. 2013) may reduce dependency of manually provided information by an user while designing rules.

Using all previous action effects as preconditions for an action requires that all actions have the same parameters, where most of the time they are not needed. This also generates another problem: either the sentence must provide all the (seldom available) information required by action parameters, or the solver must be capable of retrieving such information from previous sentences, which may be a complex task because of natural language peculiarities, or learn them automatically. Identifying specific action preconditions instead of using all previous actions effects would explore the full potential of automated planners by really exploring alternative plans in search for optimal solutions.

References

Alkhazraji, Y.; Frorath, M.; Grützner, M.; Helmert, M.; Liebetraut, T.; Mattmüller, R.; Ortlieb, M.; Seipp, J.; Springenberg, T.; Stahl, P.; and Wülfing, J. 2020. Pyperplan.

Amado, L.; Pereira, R. F.; Aires, J. P.; Magnaguagno, M.; Granada, R.; and Meneguzzi, F. 2018. Goal recognition in latent space. In *Proceedings of the 31st International Joint Conference on Neural Networks (IJCNN)*.

Asai, M., and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In McIlraith, S. A., and Weinberger, K. Q., eds., *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI)*, 6094–6101. AAAI Press.

Asai, M., and Muise, C. 2020. Learning neural-symbolic descriptive planning models via cube-space priors: The voyage home (to STRIPS). In Bessiere, C., ed., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, 2676–2682. ijcai.org.

Cresswell, S.; McCluskey, T. L.; and West, M. M. 2009. Acquisition of object-centred domain models from planning

examples. In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI.

Granada, R.; Pereira, R. F.; Monteiro, J.; Barros, R.; Ruiz, D.; and Meneguzzi, F. 2017. Hybrid activity and plan recognition for video streams. In *The AAAI 2017 Workshop on Plan, Activity, and Intent Recognition (PAIR@AAAI)*.

Hayton, T.; Porteous, J.; Ferreira, J.; Lindsay, A.; and Read, J. 2017. Storyframer: From input stories to output planning models. In *2017 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS@ICAPS)*.

Lindsay, A.; Read, J.; Ferreira, J.; Hayton, T.; Porteous, J.; and Gregory, P. 2017. Framer: Planning models from natural language action descriptions. In *27th International Conference on Automated Planning and Scheduling*.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. In Bengio, Y., and LeCun, Y., eds., *1st International Conference on Learning Representations (ICLR)*.

Steinert, M., and Meneguzzi, F. 2020. WikiHow Planning recipes Dataset: Code companion for KEPS 2020. https://doi.org/10.5281/zenodo.4056933.